Towards Edge-Aware Spatio-Temporal Filtering in Real-Time

Michael Schaffner, Florian Scheidegger, Lukas Cavigelli, Hubert Kaeslin, Luca Benini, Aljosa Smolic

***Abstract*—Spatio-temporal edge-aware (STEA) filtering methods have recently received increased attention due to their ability to efficiently solve or approximate important image-domain problems in a temporally consistent manner - which is a crucial property for video-processing applications. However, existing STEA methods are currently unsuited for real-time, embedded stream-processing settings due to their high processing latency, large memory and bandwidth requirements, and the need for accurate optical flow to enable filtering along motion paths.**

To this end, we propose an efficient STEA filtering pipeline based on the recently proposed *permeability filter* (PF) which offers high quality and halo reduction capabilities. Using mathematical properties of the PF, we reformulate its temporal extension as a causal, non-linear *infinite impulse response* filter which can be efficiently evaluated due to its incremental nature. We bootstrap our own accurate flow using the PF and its temporal extension by interpolating a quasi-dense *nearest neighbour field* obtained with an improved *PatchMatch* algorithm, which employs *binarized octal orientation maps* (BOOM) descriptors to find correspondences among subsequent frames.

Our method is able to create temporally consistent results for a variety of applications such as optical flow estimation, sparse data upsampling, visual saliency computation and disparity estimation. We benchmark our optical flow estimation on the MPI Sintel dataset, where we currently achieve a Pareto optimal quality-efficiency tradeoff with an average endpoint error of 7.68 at 0.59 s single-core execution time on a recent desktop machine.

***Index Terms*—edge-aware filter, spatio-temporal filter, patchmatch, binary descriptor, optical flow**

1. Introduction

Edge-aware (EA) filters are an important tool in many image-domain applications such as *high-dynamic range* (HDR) tone mapping [1], stylization and detail manipulation [2]. In particular their close relation to anisotropic diffusion [3] allows to use them as efficient approximators for large, non-convex regularization problems such as optical flow, disparity estimation and colorization [4], [5]. This property becomes important considering that the size of such regularization problems typically increases rapidly with the inclusion of the temporal dimension - thereby rendering these problems often computationally impractical or even infeasible [6]. The temporal extension of EA filters is straightforward [1], [6], and computationally much more tractable than optimization-based approaches. Therefore, EA filters are well-suited for efficiently approximating many image-based regularization problems in a temporally consistent manner, which is a crucial property for video-processing applications [7], [8].

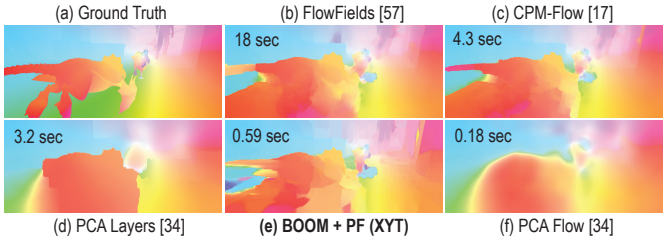

Fig. 1. Optical flow calculated with fast state-of-the-art methods (b,c,d,f) and our STEA pipeline (e). 'Sintel' images © Blender Foundation, www.sintel.org.

However, although existing *spatio-temporal, edge-aware* (STEA) methods [1], [6] are more efficient than solving complete optimization problems, they still have two major drawbacks which impede their deployment in embedded real-time settings where only limited computational and memory resources are available. First, they either operate iteratively on complete video volumes or on sliding windows, which in both cases incurs a high processing latency and requires access to a large memory with high bandwidth. Second, the temporal filtering extension requires accurate optical flow in order to align neighboring frames within the temporal window to be filtered, which is difficult to obtain efficiently. In this work, we address these issues by proposing an efficient STEA filtering pipeline based on the recently proposed *permeability filter* (PF) [1], [9]. In particular, we make the following contributions:

- We show that under certain conditions, the temporal PF can be formulated as an *infinite impulse response* (IIR) filter in time, which reduces computational complexity and bandwidth requirements compared to windowed approaches, since the filter can be incrementally evaluated over time. Also, this formulation exhibits constant memory complexity irrespective of the actual video length.
- In order to get fast and accurate flow estimates, we design a new binary feature descriptor termed *Binarized Octal Orientation Maps* (BOOM) which outperforms other state-of-the-art descriptors [10–16] in terms of *Receiver Operating Characteristic* (ROC) performance. We integrate it into the recently proposed *Coarse-to-fine PatchMatch* (CPM) method [17] in order to efficiently compute a quasi-dense *nearest neighbour field* (NNF) [18] which is used as an optical flow initialization. Dense flow is then obtained via interpolation with the PF.
- Combining the above two components, we design an efficient STEA filtering pipeline and provide an efficient implementation which is significantly faster than previous methods [1], [6]. Our single-core CPU implementation processes an MPI Sintel frame [19] in only 0.59 s (resulting in a throughput of 1.3 MPixel/s) and therefore

M. Schaffner is with ETH Zurich and Disney Research, Switzerland. F. Scheidegger, L. Cavigelli and H. Kaeslin are with ETH Zurich, Switzerland. L. Benini is with ETH Zurich, Switzerland, and University of Bologna, Italy. A. Smolic is with Disney Research, Switzerland and Trinity College, Ireland. Email: {schaffner,scheidegger,cavigelli,kaeslin,benini}@iis.ee.ethz.ch and {smolica}@scss.tcd.ie

This work has been jointly supported by Disney Research and the YINS RTD project (no. 20NA21 150939), evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

paves the way for embedded implementations in resource-constrained real-time settings.
- We present results for several image-based applications such as optical flow, disparity and saliency estimation. In particular, we demonstrate that our implementation significantly outperforms other state-of-the-art optical flow methods in terms of both speed and quality (see Figure 1).

After an introduction of related work in Section 2 we explain our STEA pipeline in Section 3 and the CPM+BOOM method in Section 4. The results and comparisons are presented in Section 5, and Section 6 finally concludes the paper.

2. Related Work

Edge-Aware Filtering Edge-aware (EA) filters are important basic building blocks in many image and video processing methods. Milanfar et al. [20] give an extensive overview of many filtering approaches. Since the establishment of the first EA methods such as the *bilateral filter* (BF) [21], a variety of EA filters have been proposed [1], [2], [22–27] for several image-based applications such as stylization, HDR tone mapping, detail editing and noise reduction. Notable instances for EA filtering are the following. The so-called *weighted least squares* (WLS) filter [22] is often used as reference method due to the high quality of its results and its ability to suppress halo artifacts by penalizing the distance between the original and filtered image. However, the drawback of this method is its high computational cost, since it requires the solution of a large linear system. *Edge-avoiding wavelets* (EAW) [23] are simpler to compute, but suffer from aliasing problems and irregular edges [27]. The local Laplacian filter is capable of producing high-quality results, but it also has the downside of being computationally demanding [27]. The authors of [26] provide a newer, more efficient implementation of the Laplacian filter, but it is not clear how to extend the method into the temporal domain. Other recent EA filtering techniques such as the *guided filter* (GF) [24], the *permeability filter* (PF) [1], [9], the *domain transform* (DT) [2] and its extension for high-order recursive filtering [25] are all efficient techniques which offer a good quality-performance tradeoff. However, as pointed out [26], the GF and DT still suffer from halo-artifacts, whereas the PF does not, since it has specifically been designed to mimic similar behaviour as the high-quality WLS filter - but with significantly lower computational complexity.

Optimization Problems A particularly interesting aspect of EA filters is their close relation to *anisotropic diffusion* (AD) [3], [28] which recently spurred a trend to use image filtering techniques to approximate optimization problems [4–6]. As explained in detail in [6], EA filters can be leveraged for a class of regularization problems which minimize energy functionals of the form $E(J) = E_{data}(J) + \lambda E_{smooth}(J)$, where E_{data} is the application specific error term, E_{smooth} enforces smoothness among neighbouring pixels and J is an unknown solution. Rather than imposing a regularization term E_{smooth} and solving for J, smoothness can be created by filtering application-specific initial conditions (which minimize E_{data} locally) with an efficient EA operation. By implicitly calculating optical flow, Lang et al. [6] also extend this concept into the temporal direction and propose an iterative *spatio-temporal edge-aware* (STEA) filtering method which is able to efficiently calculate *temporally stable* results for a variety of applications ranging from optical flow estimation itself to disparity and visual saliency estimation, sparse data upsampling and scribble propagation.

Temporal Consistency Temporal consistency is a significant problem in video processing since the frame-by-frame application of image-based methods can often produce visually disturbing temporal artifacts [1], [6–8]. As mentioned in [8], STEA filters [1], [6] are good at removing high-frequency temporal artifacts such as noise and flickering, but for low-frequency instabilities, more elaborate methods such as [7] and [8] have to be used. These algorithms can deliver better quality for certain applications - but at the cost of significantly increased complexity. Both [7] and [8] need to solve large optimization problems and typically require precomputed, accurate optical flow. Despite the mentioned limitation of STEA filtering methods, they provide efficient means for introducing temporal consistency and can be used to bootstrap accurate optical flow using sparse correspondences and interpolation.

Our work is conceptually similar to [6], but focuses on addressing the following two issues which are critical for embedded real-time settings, where only limited computational and memory resources are available. First, we formulate a temporal filtering extension which does not require access to a large memory with high bandwidth. In particular, instead of using the *domain transform* (DT) filtering kernel [2], we chose to use the *permeability filter* (PF) kernel instead, that has been originally introduced in [9] to filter disparity data, and which has been successfully extended to the temporal domain to filter HDR data in [1]. Under the simplifying assumption to only filter over all past video frames it can be reformulated as a non-linear IIR filter in time. This obviates the need to iterate over complete video volumes and results in constant computational complexity and memory requirements. The PF has similar spreading characteristics as the DT, which is important for propagation of sparse data (e.g., sparse data upsampling). In addition, the PF can be implemented with linear complexity $\mathcal{O}(N)$, where N is the amount of processed pixels, and the filter has good halo-reduction capabilities due to its close relation with the high-quality WLS filter [1].

Second, we propose a more accurate and efficient way to obtain optical flow estimates. Lang et al. [6] compute sparse point-correspondences using standard SIFT features which are then fed into their filtering framework to iteratively compute dense optical flow for the temporal filter. These standard features suffer from the fact that they are usually not regularly spaced, but rather occur in clusters in textured areas. In order to obtain sufficient coverage of the whole image, the interest point detectors therefore have to be tuned to yield many more points, which makes the matching process slow and inefficient. In this work, we also use the filtering framework to interpolate a sparse flow-initialization, but we propose to use a more accurate and efficient matching method, which returns a regular, quasi-dense correspondence field.

Optical Flow Estimation Optical flow estimation is an extensive field and therefore we only mention relevant methods and refer to [29], [30] for more details on standard algorithms.

Several recently proposed optical flow methods rely on sparse feature matching methods in order to gain efficiency. Sparse feature matches obtained between two temporally adjacent frames are typically used as initializations for variational refinements [17], [31], [32] or interpolation methods [6], [33], [34]. In particular, a recently proposed multi-level method called CPM [17] has been shown to provide a good speed/performance tradeoff (currently it is among the top ranking methods on the MPI Sintel dataset [19]). CPM calculates a quasi-dense NNF via a randomized nearest neighbor search. Compared to standard feature matching approaches which first employ an interest point detector, such NNFs achieve much more regularly distributed point-correspondences. However, standard NNFs such as computed by PatchMatch [18] originally suffered from many outliers. This issue has been addressed by CPM by performing the randomized search on an image pyramid and making use of SIFT descriptors [10] instead of a block matcher. In order to gain efficiency, the NNF is calculated on a subsampled grid – and hence it is quasi-dense. In order to obtain a dense flow estimate, the obtained NNF is interpolated using the geodesic interpolation and variational refinement method from EpicFlow [32]. CPM-Flow takes around 4.3 s on a single core machine, where around 1.3 s are spent calculating dense SIFT descriptors and performing CPM, and the remaining 3 s are spent in the variational refinement step of EpicFlow. Note that CPM-Flow has similarities with SIFT-Flow [35] and DAISY [15], which both use a dense scan formulation to efficiently compute a descriptor for each pixel. In fact, CPM-Flow uses the same dense formulation as SIFT-Flow for efficient operation.

In this work, we leverage the advantages of CPM, but without the costly SIFT descriptors and the variational refinement step. To this end, we propose to use CPM with an efficient, binary descriptor which we specifically developed for this task. After obtaining an NNF using our new version of CPM, we approximate the variational refinement step using our much faster edge-aware filtering pipeline.

Features Current state-of-the-art descriptors can be categorized into two classes, namely *floating-point* methods such as SIFT [10], SURF [11] or DAISY [15] which produce numeric feature vectors and *binary* methods which produce bit vectors. The latter class of descriptors has received increased attention over the last couple of years since they often can be calculated and matched more efficiently than their floating-point counterparts. Notable binary standard methods are BRIEF [12], BRISK [36], FREAK [13] and SKB [14], [16]. Zhou et. al. [37] propose a binary SIFT (B-SIFT) variant, which applies a post-processing step to standard SIFT descriptors, and hence is computationally more expensive than the other binary descriptors mentioned.

An interesting observation is that gradient-based descriptors like *histograms of oriented gradients* (HOG) [38] (and variants thereof) often exhibit superior ROC performance compared to intensity-based descriptors [39–41]. Moreover, HOG-based features have also proven to be useful for calculating optical flow with large displacements [42].

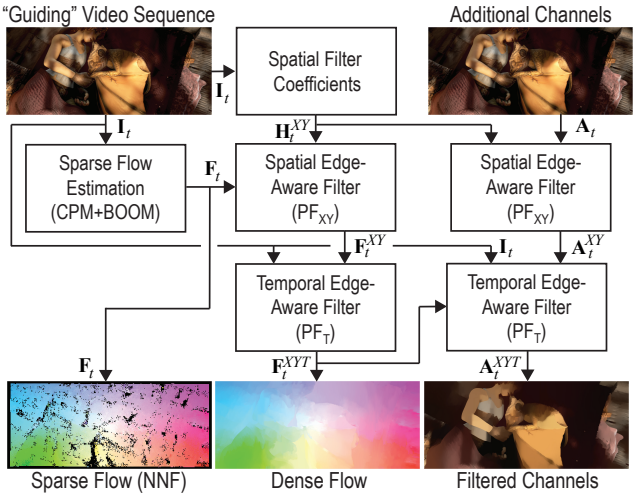

Fig. 2. Overview of the STEA Filtering Pipeline. Refer to the text for more details. 'Sintel' images copyright © Blender Foundation, www.sintel.org.

Machine learning has been successfully applied to optimize feature arrangements and reduce the dimensionality of the vectors [39–41], [43], [44]. In particular, methods employing *AdaBoost* have recently led to a variety of top-performing descriptors such as BinBoost [39], Bamboo [40] and LDDB [41]. However, learned descriptors typically forfeit some of the efficiency gain when extracting the descriptor, since they often have irregular computation patterns and employ costly dimensionality reducing projections.

For the NNF calculation using CPM, a good tradeoff among extraction speed, matching speed and ROC performance is important. On one hand we have state-of-the-art methods which provide fast extraction, but exhibit a larger memory footprint and fair ROC performance. On the other hand, there exist descriptors with small memory footprint and/or better ROC performance – but these are typically more costly to compute. We strive to seek a balance between these attributes and present a simple binary descriptor termed *Binarized Octal Orientation Maps* (BOOM), which can be efficiently calculated and matched, and exhibits similar performance as SIFT and DAISY. BOOM is inspired by HOG-like methods, but takes them a step further by providing an effective way to obtain a compact 256 bit representation. Note that BOOM has similarities with binarized HOG (BHOG) variants [45–47] from the domains of human detection and sketch-based image retrieval. However, as described in more detail in Section 4-A, BOOM employs a different cell layout and binarization strategy, and is based on orientation maps introduced by Tola et al. [15] instead of standard HOGs, leading to significantly better ROC performance as demonstrated in Section 5.

3. STEA Filtering Pipeline

An overview of our STEA filtering pipeline is shown in Figure 2. The inputs are a *guiding video sequence* $\mathbf{I}_t$ and *additional attribute channels* to be filtered $\mathbf{A}_t$, where t is the frame index. The guiding frames $\mathbf{I}_t$ are used to derive the edge-aware filter coefficients $\mathbf{H}_t^{XY}$ and to estimate sparse optical flow $\mathbf{F}_t$. The filter coefficients $\mathbf{H}_t^{XY}$ are then used to

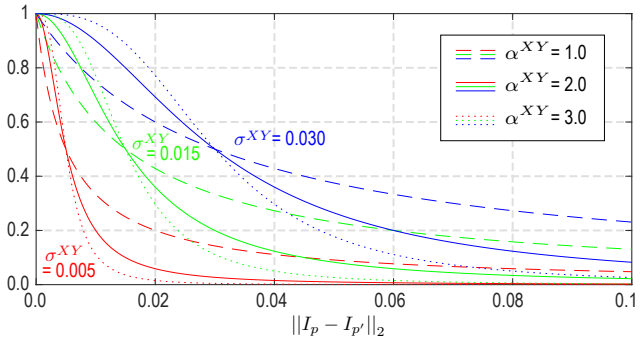

Fig. 3. Impact of σ^{XY} and α^{XY} on the shape of the edge-stopping function. σ^{XY} controls the transition point and α^{XY} the falloff rate.

perform spatial filtering operations in order to turn the sparse optical flow into a dense flow field $\mathbf{F}_t^{XY}$ and to filter the additional data channels in spatial directions to produce $\mathbf{A}_t^{XY}$. The optical flow estimate $\mathbf{F}_t^{XY}$ is then used to enable temporal filtering along motion paths. First, it is used to temporally filter itself to produce a stabilized version $\mathbf{F}_t^{XYT}$, and this temporally stabilized flow is then used to filter the additional channels in time. In this example we have $\mathbf{A}_t = \mathbf{I}_t$ in order to illustrate the impact of edge-aware filtering, but in practice $\mathbf{A}_t$ usually contains other feature maps such as disparity or saliency values. The pipeline outputs sparse and dense flow estimates $\mathbf{F}_t$, $\mathbf{F}_t^{XYT}$, as well as the filtered channels $\mathbf{A}_t^{XYT}$.

Similar to other recent flow estimation algorithms [17], [31–33], flow is estimated using sparse feature extraction, followed by an efficient sparse-to-dense conversion. The difference of our method w.r.t. related work is, that our conversion consists of a fast edge-aware interpolation without any variational refinement. We will see in Section 5, that this approach offers a competitive speed-quality trade-off. The employed feature matching method (CPM+BOOM) is an improved NNF method based on sparse features and will be explained in Section 4.

For the EA filtering operation we selected the recently proposed *permeability filter* (PF) [1], [9] due to its high efficiency and good quality. It can be formulated as an iterative application of ideally parallelizable 1D filtering passes with constant complexity per processed pixel. This property is shared by the *domain transform* (DT) filter [2], but the PF has the additional benefit of being closely related to the WLS filter, which is known to produce high-quality anisotropic filtering results with good halo artifact reduction [2], [22], [23]. The temporal filter employs the same PF, but as opposed to its original extension into the temporal domain [1], we formulate it as an incremental IIR filter which does not iterate over

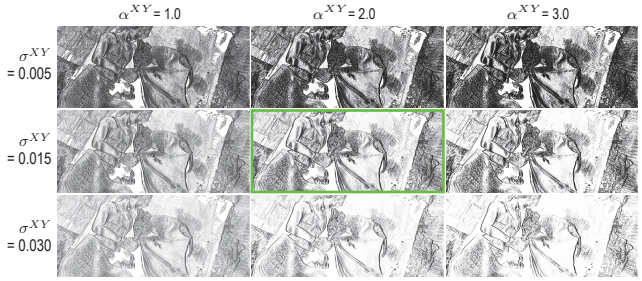

Fig. 4. Impact of σ^{XY} and α^{XY} on the permeability map $\tilde{\pi}^X$. The typically used configuration $\sigma^{XY} \approx 0.015$ and $\alpha^{XY} = 2$ is highlighted in green.

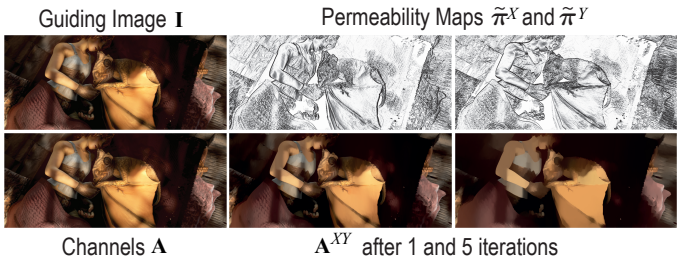

Fig. 5. Edge-aware spatial filtering using permeability maps $\tilde{\pi}^X$ and $\tilde{\pi}^Y$. Note that $\mathbf{A}$ is set to $\mathbf{I}$ for illustrative purposes in this example. 'Sintel' images copyright © Blender Foundation, www.sintel.org.

large temporal volumes. This enables efficient, low-latency implementations with constant memory complexity.

Note that temporal filtering as performed here and in related methods implicitly assumes the feature maps to be approximately constant over time (at least on a local scale) in order to produce valid results. This assumption does not always hold – especially for optical flow if the motion patterns change rapidly, or if the video has been captured at a low frame-rate. To this end, our temporal filter employs a flow gradient measure that is able to detect regions where the constancy assumption is violated in order to prevent errors.

In the following, we will describe the spatial and temporal filtering steps, and the employed NNF method for flow computation is explained in the subsequent section.

A. Spatial Filtering of Dense Data

The filter used in this work belongs to a class of filters which is defined by iterative application of the recurrence equation

$$J_p^{(k+1)} = \sum_{q \in \Omega} H_{pq}\, J_q^{(k)} + \lambda^{XY}\, H_{pp} \left(A_p - J_p^{(k)} \right), \quad (1)$$

where A_p denotes the input data to be filtered at position p at frame t, $J_p^{(k)}$ is the diffusion result at position p after k iterations. The set Ω contains all pixel positions of a frame, and H_{pq} are elements of the row stochastic [20] matrix $\mathbf{H}$ defining the filter. The iteration is initialized with $\mathbf{J}^{(0)} = \mathbf{A}$. Frame indices are omitted in this subsection since all operations are only applied to spatial dimensions. The first term of (1) is the actual shift-variant convolution and the second term is a fidelity term with $\lambda^{XY} \in [0, 1]$ which can be used to bias the iteration towards the input data $\mathbf{A}$. Aydın et al. [1] explain that the choice of $\lambda^{XY} = 1$ significantly reduces halo artifacts.

The PF is a specific instance of (1) with two separate filter matrices $\mathbf{H}^X$ and $\mathbf{H}^Y$ for filtering operations in horizontal and vertical direction, respectively. These operations are applied in alternating fashion, and the concatenation of one X and one Y pass constitutes one spatial filter iteration. The two matrices $\mathbf{H}^X$ and $\mathbf{H}^Y$ are defined via *permeability weights* π_{pq} between two pixels p and q which control the local diffusion strength. Below we summarize how to obtain the coefficients $\mathbf{H}^X$ for the horizontal filter. The derivation for $\mathbf{H}^Y$ is analogous.

The permeability between two neighboring pixels $p = (x, y)$ and $p' = (x + 1, y)$ is defined as

$$\tilde{\pi}_p^X = \left(1 + \left| \frac{||I_p - I_{p'}||_2}{\sqrt{3} \cdot \sigma^{XY}} \right|^{\alpha^{XY}} \right)^{-1}, \quad (2)$$

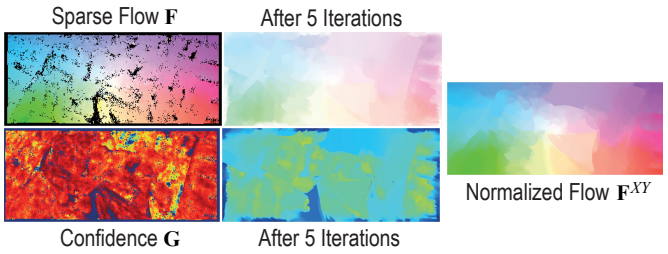

Fig. 6. Edge-aware spatial filtering of sparse data (in this case 2D flow vectors). 'Sintel' images copyright © Blender Foundation, www.sintel.org.

which is a variant of the Lorentzian edge-stopping function, applied to the color distance between p and p' *of the guiding image* $\mathbf{I}$. This function evaluates close to 0.0 if the color distance between the two pixels is high, and close to 1.0 if the distance is low. As illustrated in Figures 3 and 4, σ^{XY} controls the transition point and α^{XY} the falloff rate of the edge-stopping function. Typical values are $\sigma^{XY} \approx 0.015$ and $\alpha^{XY} = 2$. Permeabilities between arbitrary pixels are then defined as

$$\pi^X_{pq} = \begin{cases} 1 & \text{if } p = q, \\ \Pi_{n=p_x}^{q_x-1} \tilde{\pi}^X_{(n,p_y)} & \text{if } p_x < q_x, p_y = q_y \\ \Pi_{n=q_x}^{p_x-1} \tilde{\pi}^X_{(n,p_y)} & \text{if } p_x > q_x, p_y = q_y \\ 0 & \text{else.} \end{cases} \tag{3}$$

The final filter coefficients H_{pq} are then obtained by normalizing the pairwise permeabilities as

$$H_{pq} = \pi^X_{pq} \left(\sum_{n=1}^{w} \pi^X_{(n,p_y),q} \right)^{-1}, \tag{4}$$

where w is the image width. This filtering process is illustrated in Figure 5, where the guiding image is also subject to filtering. We observe that even with a low number of XY passes in the order of 5 iterations, the PF achieves strong edge-aware diffusion. Note that the permeabilities in (3) are defined such that the filtering operations reduce to 1D operations over image rows or columns. As will be shown later, this specific filter can be implemented with efficient scanline operations.

B. Spatial Filtering of Sparse Data

Although most EA filters are not strictly interpolating filters, they can also be used to efficiently spread sparse data $\mathbf{F}$, i.e., to perform an edge-aware, sparse-to-dense conversion. As shown in [6], [9], this can be conveniently achieved by introducing a normalization map $\mathbf{G}$, that contains nonzero values at sparse sample positions, and is zero otherwise. The map $\mathbf{G}$ is subject to the same filtering operation which is applied to the corresponding sparse data channels. After the desired amount of filtering iterations K, the map is used to normalize the filtered data element-wise as $\mathbf{F}^{XY} = \mathbf{F}^{(K)} ./ \mathbf{G}^{(K)}$. Figure 6 illustrates the sparse-to-dense conversion of optical flow. Note that the normalization map $\mathbf{G}$ can additionally be used to incorporate *data confidence* by assigning values between 0.0 and 1.0 at the sparse sampling positions in order to give more weight to those samples which are considered to be more accurate than others. For sparse flow-vectors, we use the matching confidence, normalized to the range $[0.0, 1.0]$.

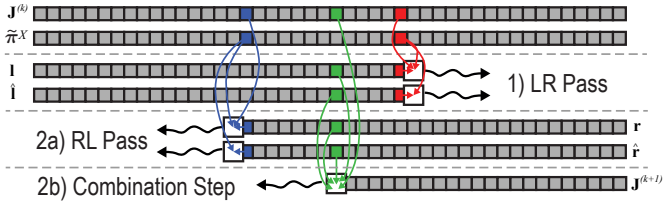

Fig. 7. The PF can be efficiently evaluated with only two scan line passes due to the multiplicative concatenation of the permeabilities. As explained in the text, left- and right-sided intermediate results can be formed, which allow to compute the final result without explicitly evaluating the full convolutions.

C. Efficient Formulation of the Spatial Filter

As shown by [9], the multiplicative concatenation of the permeabilities allows to formulate the filtering operation as an efficient two-pass scan line operation with constant computational complexity per pixel. The formulas are only given for the k-th horizontal iteration since their counterparts for the vertical iteration follow analogously.

As illustrated in Figure 7, the intermediate results l_p and the corresponding normalization values $\hat{l}_p$ are computed in a first left-right scan-line pass using the recurrences

$$\begin{aligned} l_p &= \tilde{\pi}^X_{(p_{x-1},p_y)} \left(l_{(p_{x-1},p_y)} + J^{(k)}_{(p_{x-1},p_y)} \right), \\ \hat{l}_p &= \tilde{\pi}^X_{(p_{x-1},p_y)} \left(\hat{l}_{(p_{x-1},p_y)} + 1.0 \right). \end{aligned} \tag{5}$$

In a second right-left pass (2a in Figure 7), the right-sided quantities r_p and $\hat{r}_p$ are then computed as

$$\begin{aligned} r_p &= \tilde{\pi}^X_{p} \left(r_{(p_{x+1},p_y)} + J^{(k)}_{(p_{x+1},p_y)} \right), \\ \hat{r}_p &= \tilde{\pi}^X_{p} \left(\hat{r}_{(p_{x+1},p_y)} + 1.0 \right). \end{aligned} \tag{6}$$

The result is finally calculated by combining and normalizing the intermediate results (step 2b in Figure 7), and adding the bias term $\lambda^{XY} \cdot \left(A_p - J^{(k)}_p \right)$ as

$$J^{(k+1)}_p = \frac{l_p + \left(1 - \lambda^{XY}\right) \cdot J^{(k)}_p + \lambda^{XY} \cdot A_p + r_p}{\hat{l}_p + 1.0 + \hat{r}_p}. \tag{7}$$

Note that this third step can be efficiently carried out on-the-fly during the right-left pass, since all intermediate results are available at position p at this point. Therefore, the whole procedure results in exactly two scan line passes. Individual scan lines of one X or Y iteration are independent and can be conveniently parallelized. The initial values $l_{(1,p_y)}$, $\hat{l}_{(1,p_y)}$, $r_{(w,p_y)}$, $\hat{r}_{(w,p_y)}$ are all set to zero (w is the image width).

D. Temporal Filtering

Existing Approaches In related STEA filtering methods [1], [6], the temporal extension of the EA filter has been achieved in two different ways, and both solutions can be problematic in terms of computational- and memory complexity when targeting efficient STEA embodiments capable of real-time operation. Lang et al. [6] iteratively apply XY and T iterations to the complete video volume by following the motion paths as illustrated in Figure 8a. Similarly to our work, they use their DT-based filter to bootstrap optical flow. Their method works on a large data structure of linked lists representing the motion paths, and requires a large amount of high-bandwidth memory in order to store the complete video volume. Also, the approach is limited to batch-wise processing, which incurs a high

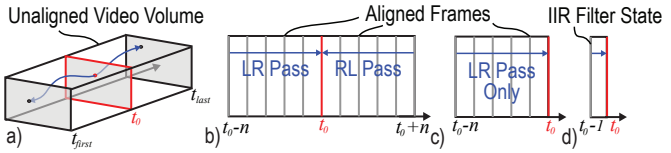

Fig. 8. Different temporal extensions of the filter: a) iterative filtering along motion paths within the complete video cube, b) single filtering step within aligned symmetric (noncausal) window, c) single filtering step within left-sided (causal) window, d) IIR formulation of c).

processing latency and therefore makes this approach unsuitable for real-time, stream processing settings. Aydın et al. [1] formulate their filter on a reduced temporal sliding window comprising in the order of ± 10 spatially aligned frames, which reduces the latency to about 10 frames. Also, they reduce the computational complexity by applying only one T iteration after the spatial XY iterations. This turned out to be sufficient to obtain temporally smooth results. However, the need to align all frames within the temporal window still incurs a non-negligible computational overhead. We therefore refrain from these approaches and seek to formulate the filter in an incremental fashion, which can be evaluated very efficiently and with constant memory complexity. As shown in more detail below, this is possible by leveraging the mathematical properties of the permeability filter.

Formulation of the Recursive Permeability Filter First, we make the following two assumptions:

- We assume that only one T iteration is applied after the XY iterations. This assumption is valid in practice, since already one T iteration improves temporal consistency [1]. In addition, using only one T iteration eliminates the 'chicken-and-egg' problem described in [6], which arises when the dense optical flow estimate used in the frame alignment is bootstrapped using the filter itself.
- We assume that all data to be filtered, $\mathbf{J}_t^{XY}$, has been aligned to the centering frame t_0 within a certain temporal neighbourhood $\mathcal{T} = [t_0 - n, ..., t_0, ...t_0 + n]$, $n \in \mathbb{N}^+$. For the moment we neglect the fact that this assumption implicitly requires the availability of optical flow for these frames in order to perform the alignment, as this does not pose a problem anymore in the incremental formulation.

With these assumptions, we can calculate one T filtering iteration with the same recurrence equations defined in (5) and (6), but using temporal permeabilities $\tilde{\boldsymbol{\pi}}_t^T$ which will be defined in Section 3-E. In other words, we can compute the intermediate results $\mathbf{l}_t$, $\hat{\mathbf{l}}_t$, $\mathbf{r}_t$, $\hat{\mathbf{r}}_t$, where a left-right pass corresponds to a forward pass in time and vice versa for the right-left pass (the bold symbols now represent full-frame matrices, indexed by the frame number t). The result of one T iteration for the frame at time t_0 is then given by

$$\mathbf{J}_{t_0}^{XYT} = \frac{\left(\mathbf{l}_{t_0} + \left(1 - \lambda^T\right) \cdot \mathbf{J}_{t_0}^{XY} + \lambda^T \cdot \mathbf{A}_{t0} + \mathbf{r}_{t_0}\right)}{\left(\hat{\mathbf{l}}_{t_0} + 1.0 + \hat{\mathbf{r}}_{t_0}\right)}. \quad (8)$$

We now simplify the problem by replacing the symmetric temporal neighborhood with a causal, one-sided time window $\mathcal{T} = [t_0 - n, ..., t_0]$, $n \in \mathbb{N}^+$. This is a valid simplification, especially for real-time settings where low-latency is required

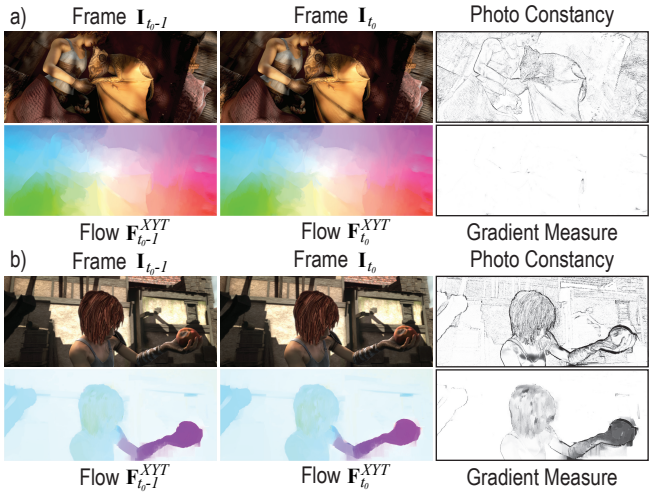

Fig. 9. Photo constancy and gradient measures for shots with slow and steady zoom (a), and fast-changing motion (b). 'Sintel' images © Blender Foundation, www.sintel.org.

and no information about future frames is available. We can observe that the recurrence equation then reduces to

$$\mathbf{J}_{t_0}^{XYT} = \frac{\left(\mathbf{l}_{t_0} + \left(1 - \lambda^T\right) \cdot \mathbf{J}_{t_0}^{XY} + \lambda^T \cdot \mathbf{A}_{t0}\right)}{\left(\hat{\mathbf{l}}_{t_0} + 1.0\right)}, \quad (9)$$

since r_{t_0} and $\hat{r}_{t_0}$ are zero in this case. This means that one temporal iteration can basically be calculated using just one left-right pass. If we now let $n \to \infty$, and by considering the fact that the left-right pass is defined as a recurrence, we see that it is possible to obtain a non-linear IIR filter in time which only requires one recurrence evaluation for each time step. The only missing part to consider is alignment. Recall that we assumed that all frames within $\mathcal{T}$ are aligned to the frame t_0 in the first place. If we drop this assumption and want to reuse the IIR filter state of the previous time-step $t_0 - 1$ to update the recurrence equation, we have to re-align it to the current frame t_0, which can be conveniently done using forward-warping based on the flow estimate of $\mathbf{F}_{t_0-1}^{XYT}$ which is already computed and available:

$$\begin{aligned} \mathbf{l}_{t_0} &= \tilde{\boldsymbol{\pi}}_{t_0}^T \operatorname{warp}_{\mathbf{F}_{t_0-1}^{XYT}} \left(\mathbf{l}_{t_0-1} + \mathbf{J}_{t_0-1}^{XYT}\right), \\ \hat{\mathbf{l}}_{t_0} &= \tilde{\boldsymbol{\pi}}_{t_0}^T \operatorname{warp}_{\mathbf{F}_{t_0-1}^{XYT}} \left(\hat{\mathbf{l}}_{t_0-1} + 1.0\right). \end{aligned} \quad (10)$$

Note that this recurrence step in time can be efficiently implemented with constant memory and low latency.

E. Temporal Permeabilities and Flow Gradient Measure

As proposed in [1], we use a combination of color constancy and a flow-gradient magnitude measure in order to calculate permeabilities in temporal direction. The photo constancy is a straightforward extension of the spatial permeabilities

$$\tilde{\boldsymbol{\pi}}_t^{photo} = \left(1 + \left| \frac{\left|\left| \mathbf{I}_t - \operatorname{warp}_{\mathbf{F}_{t-1}^{XYT}} \left(\mathbf{I}_{t-1}\right) \right|\right|_2}{\sqrt{3} \cdot \sigma^{photo}} \right|^{\alpha^{photo}}\right)^{-1}, \quad (11)$$

and allows filtering along motion paths with similar color values in the guiding image. The gradient-magnitude measure is calculated similarly as

$$\tilde{\boldsymbol{\pi}}_t^{grad} = \left(1 + \left| \frac{\left|\left| \mathbf{F}_t^{XY} - \mathrm{warp}_{\mathbf{F}_{t-1}^{XYT}} \left(\mathbf{F}_{t-1}^{XYT} \right) \right|\right|_2}{\sqrt{2} \cdot \sigma^{grad}} \right|^{\alpha^{grad}} \right)^{-1}, \tag{12}$$

and prevents temporal filtering with rapid motion changes where flow and warping artifacts are likely to occur. In other words, this measure introduces a bias towards the current frame in regions where the constancy assumption of the data to be filtered may not hold and where temporal filtering may hence introduce errors. Divisions and exponentiations are all element-wise in the two equations above. The two measures are then multiplied (element-wise) to get the final temporal permeabilities $\tilde{\boldsymbol{\pi}}_t^T = \tilde{\boldsymbol{\pi}}_t^{photo} \cdot \tilde{\boldsymbol{\pi}}_t^{grad}$. Figure 9 shows two examples, where the first shot exhibits slow and consistent camera zoom, and the second shot contains quickly moving parts (left arm of the main character). We can see that the gradient measure effectively detects regions where warping artifacts are likely to occur.

F. Forward Warping using EWA Splatting

For the implementation of the forward mapping operator $\mathrm{warp}_{\mathbf{F}_{t_0-1}^{XYT}}(.)$, we use the *elliptic-weighted average* (EWA) splatting framework [48], [49] which is an efficient high-quality resampling technique for non-linear, two-dimensional forward mapping transformations. In addition to the temporal permeabilities $\tilde{\boldsymbol{\pi}}_t^{photo}$ and the flow gradient measure $\tilde{\boldsymbol{\pi}}_t^{grad}$ introduced before, we additionally cull image regions that undergo a significant density change due to the transformation[1]. I.e., only image regions where the density stays within the range [0.25, 2.5] are accumulated in the forward mapping buffer. This can prevent stretching and compression artifacts (e.g., at motion boundaries). Regions that are empty after warping (e.g., disoccluded areas) are not filtered in time.

We use a forward mapping technique in this work since this allows us to simplify the overall pipeline and work with forward flow only. It should however be noted that backward flow could be used in addition to forward flow to perform stricter consistency checks in the warping step and resolve warping ambiguities more effectively.

4. CPM with Binary Descriptors

Our STEA pipeline relies on sparse flow vectors to obtain dense optical flow via EA filtering. These flow vectors are extracted using an improved version of the recently proposed CPM [17] method, which we will explain in the following.

CPM is a NNF method, which has been developed to provide accurate optical flow vectors on a coarse, but regular grid. As opposed to the original PatchMatch algorithm [18], the randomized search is formulated on subsampled grids over an image pyramid, where matching information is propagated from coarser to finer grids in a top-down fashion. Instead

[1]Local density changes due to the transformation can be estimated at all vertices of the warping grid using finite difference approximations of the Jacobian determinants.

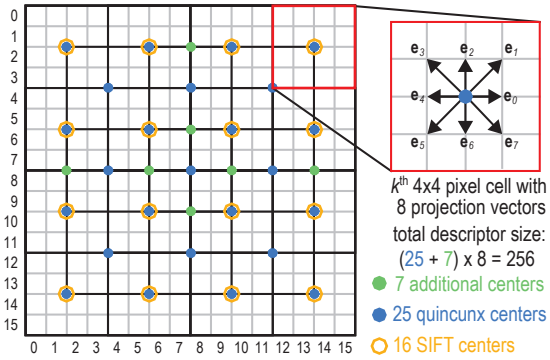

Fig. 10. The descriptor is applied to a 16×16 pixel patch and contains 32 4×4 pixel bins with 8 orientation map responses. This results in 256 values overall. The 4×4 spatial binning cells are arranged in a similar manner as in SIFT, but instead of 16 non-overlapping cells, 32 partially overlapping cells are employed (25 quincunx centers plus 7 additional centers).

of the simple block matching metric, CPM uses SIFT-Flow features [35] to compute costs, since this provides more robust matches. SIFT-Flow is a pixel-dense formulation of SIFT [10] and returns a pixel-dense *descriptor field* for a given image. It can be computed more efficiently than the sparse descriptor since the regular, dense setting makes it possible to share intermediate binning results among overlapping descriptors.

In this work, we found that the runtime of CPM can be even further decreased by switching from costly SIFT descriptors to binary descriptors. Such binary descriptors can be computed and matched more efficiently, and they have a smaller memory footprint. However, most binary descriptors with equivalent (or better) ROC performance than SIFT have been obtained with machine learning techniques such as [39–41], [43], [44] and often exhibit irregular computation patterns, making it difficult to share intermediate results among overlapping descriptors in a dense scan setting. We therefore introduce a new binary descriptor termed BOOM, which has a similar, regular binning cell layout and ROC performance as SIFT – while being computationally more efficient. The descriptor is explained below, followed by the modifications to CPM. Performance results and comparisons are presented later in Section 5.

A. Binarized Octal Orientation Maps

When looking at existing descriptors, we observe that gradient-based features consistently provide better performance than intensity-based ones [39–41]. Well performing descriptors (binary and non-binary) such as BinBoost [39], SIFT [10], DAISY [15] or SURF [11] therefore employ *Histograms of Oriented Gradients* (HOG) [38] or variations thereof as basic building blocks. Also, normalization of the descriptor vector is essential for good ROC performance [10], [39], [43]. The proposed BOOM descriptor combines HOG-like features with a normalized binarization scheme in order to get a descriptor with similar performance and regular layout as SIFT and DAISY, but with the computational efficiency of binary descriptors.

Support Region and Orientation Maps The input to our BOOM descriptor is a normalized 18×18 pixel patch $\mathbf{P}$, that is first preprocessed with Prewitt operators to get the gradients

$\mathbf{g}_j = [G_j^h, G_j^v]$ within a 16×16 pixel support $\mathcal{S}^{16\times16}$:

$$\mathbf{G}^h = \mathbf{P} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \mathbf{G}^v = \mathbf{P} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}. \quad (13)$$

BOOM is formulated as a collection of orientation maps [15] on 4×4 pixel cells (spatial bins) arranged in a quincunx pattern, as shown in Figure 10. In order to align the amount of responses with a power of 2, we additionally add 7 cells around the center region. The orientation maps are then built within these 32 spatial bins using eight directions as

$$b_{ki} = \sum_{j \in \mathcal{N}_k^{4\times4}} \max\left(0, \langle \mathbf{e}_i, \mathbf{g}_j \rangle\right), \quad (14)$$

where b_{ki} are the orientation map responses and $\mathcal{N}_k^{4\times4}$ denotes the 4×4 pixel neighbourhood of the k^{th} spatial bin. Overall this results in $32\times8 = 256$ orientation map responses b_{ki}. Note that we use projections onto the following direction vectors

$$\left[\mathbf{e}_0 \dots \mathbf{e}_7\right] = \begin{bmatrix} 1 & 1 & 0 & -1 & -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 \end{bmatrix} \quad (15)$$

in order to calculate the gradient contribution to a specific orientation map. Negative projections are clamped to zero. Compared to standard HOG binning approaches, this scheme has the advantage that angles do not have to be calculated explicitly. Note that the use of unnormalized vectors and Prewitt masks is intentional here as this allows to implement these operations with integer additions only.

Normalization and Binarization It is crucial to normalize gradient based descriptors in order to get good descriptor performance [39], [43]. However, straightforward normalization requires the computation of costly L_2 norms and divisions. As we aim to binarize the final responses, we use a slightly different approach. First, we compute a simple approximation (from [50], page 89) of the average gradient magnitude which can be carried out with integer arithmetic as

$$s = \sum_{j \in \mathcal{S}^{16\times16}} 5 \cdot \max\left(|g_j^0|, |g_j^1|\right) + 3 \cdot \left(|g_j^0| + |g_j^1|\right). \quad (16)$$

The normalization of this approximation with $(5 + 3) = 8$ is implicitly carried out in the binarization step

$$d_{ki} = \begin{cases} b_{ki} \cdot \theta_0 > s & \text{, if } i \text{ even,} \\ b_{ki} \cdot \theta_1 > s & \text{, if } i \text{ odd,} \end{cases} \quad (17)$$

which yields a binary 256 bit descriptor d_{ki}. The parameters θ_0 and θ_1 compensate for all normalizations omitted so-far. In order to determine the value of these parameters, we swept them over the range of interest on the Liberty 100k dataset of Brown et al. [43]. The minimal error is achieved for $\theta_0 = 980$ and $\theta_1 = 230$. We align these values to powers of two, $\theta_0 = 1024$ and $\theta_1 = 256$, since these can be implemented with simple bitshifts and the performance impact is negligible.

The BOOM descriptor can be efficiently implemented using only integer arithmetic. No divisions, trigonometric- and transcendental functions are required. As elaborated in more detail in Section 5, its ROC performance is similar to SIFT descriptors with 128 entries and to DAISY descriptors with 200 entries, while at the same time being more efficient than state-of-the-art binary descriptors in terms of processor execution time. Similar to SIFT-flow and DAISY, BOOM can be implemented as an efficient dense scan method.

Difference to Binarized HOG Descriptors Although the proposed descriptors has similarities with binarized HOG descriptors (BHOG) that have been proposed in the domain of human detection [45], [46] and sketch-based image retrieval [47], the following important differences should be noted:

1) Usually, HOG descriptors employ non-overlapping cell layouts, whereas our descriptor uses partially overlapping cells arranged in a quincunx pattern.
2) BHOG descriptors employ gradient binning – i.e., accumulation of the gradient magnitude into corresponding direction bins – instead of orientation maps.
3) BHOG descriptors use either fixed thresholds or the average bin response within a single 4×4 pixel cell to threshold the bin responses. In contrast, our BOOM descriptor uses the average gradient magnitude over the complete descriptor support.

As shown in Figure 11a in Section 5, these design decisions have a significant performance impact. In addition, the proposed normalization technique does not rely on the final HOG values. The binarization threshold can hence be accumulated in parallel to the orientation map responses, and does not require the evaluation of high-dimensional vector norms.

B. Modifications of CPM and Parametrization

Our CPM closely follows the implementation of [17], and therefore we refrain from repeating all details of the method here and only mention important differences:

- Instead of SIFT-Flow, we use a dense scan implementation of BOOM. In addition, we applied CLAHE [51] with threshold $\theta_{clahe} = 1$ to the input images in order to improve performance on low-contrast image regions.
- Instead of performing the forward-backward check on the two finest levels, we perform one check on the coarsest and one check on the finest level. Outliers on the coarsest level are re-initialized. In addition, we also threshold the matching costs in order to remove very bad matches. Currently, this threshold is set to $\theta_{desc} = 88$.
- We use the same amount of pyramid levels ($n_{levels} = 5$), propagation iterations ($n_{prop} = 6$), and a grid spacing of $d = 3$. However, we employ a slightly relaxed search radius r of 11 instead of 4, and reduce the forward-backward check threshold θ_{flow} from 3 to 1 pixel as we found that this improves the accuracy of the method.
- The original CPM method does not provide sub-pixel accurate results. In order to improve the precision for small flow-vectors, we perform a quadratic interpolation step by reusing the matching costs on 3×3 neighborhoods around matches returned by CPM.

5. Results

First, we provide implementation details, followed by an evaluation of BOOM and its combination with CPM. Then, we evaluate the optical flow obtained with our STEA pipeline, and we show results for additional image-based applications.

TABLE I
BOOM PERFORMANCE AND COMPARISON WITH OTHER STATE-OF-THE-ART DESCRIPTORS ON THE UBC DATASET [43].

Descriptor		Type	Size [Bytes]	95% Error (FPR for TPR=0.95) Liberty	Notredame	Yosemite	Time per Descriptor [μs]	Implementation	Processor Model
$SIFT_{128}$	[10]	FP	128	29.54	23.65	29.27	483.6	VLFeat‡	i7 3.1 GHz (1 Thread)
$SURF_{128}$	[11]		128	44.72	36.13	42.77	69.8	MATLAB	i7 3.1 GHz (1 Thread)
$SURF_{64}$	[11]		64	43.28	32.53	42.16	63.1	MATLAB	i7 3.1 GHz (1 Thread)
$DAISY_{200}$	[15]		200	28.33	22.66	31.71	-×	author	-
$Brown_{29}$	[43]		29	**17.56**	**11.98**	**13.55**	-	-	-
$BinBoost_{64}\{128\}$	[39]	Bin	**8**	21.08†	15.72†	20.93†	~1000§	author	Mobile i7 2.66 GHz
$LDA\text{-}Hash_{128}$	[44]		16	49.66	51.58	52.95	-	-	-
$Bamboo_{128}$	[40]		16	49*	-	-	~20§	author	-
$LDDB\text{-}U_{256}$	[41]		32	-	-	-	~154§	author	ARM Cortex A9 1GHz
$FREAK_{512}$	[13]		64	52.10	43.48	46.63	34.4	MATLAB	i7 3.1 GHz (1 Thread)
$BRISK_{512}$	[36]		64	73.74	70.24	67.55	38.2	MATLAB	i7 3.1 GHz (1 Thread)
$BRIEF_{256}$	[52]		32	53.47	46.51	51.12	29.9	author	i7 3.1 GHz (1 Thread)
SKB_{256}	[14], [16]		32	67.07	64.95	62.42	**10.9**	ours	i7 3.1 GHz (1 Thread)
$BOOM_{256}$	this work		32	27.27	22.04	27.23	**10.9**	ours	i7 3.1 GHz (1 Thread)

§ from original publication * 200k dataset † averaged training set numbers from [39] ‡ http://www.vlfeat.org/ × N/A as implemented as dense-scan.

A. Implementation and Choice of Parameters

We provide an efficient single core implementation of the complete filtering pipeline in plain C++. Apart from the `__popcnt64` SSE intrinsic for computing the descriptor matching cost, we do not make use of other SSE/AVX vector intrinsics in order to facilitate comparisons. The descriptor calculation and CPM parts only use integer arithmetic (with the exception of the subpixel interpolation). The filtering stages are implemented using single precision arithmetic. Timings for the different substeps of our pipeline have been measured on an i7-5557U machine (3.1 GHz) with 16 GB RAM. Detailed numerical results are given in the following subsections.

Note, that there is significant acceleration potential by leveraging vector intrinsics, multiple threads or GPU implementations – especially for high resolution video content where data-level parallelism becomes abundant (Nehab et al. [53] provide useful insights of how recursive filtering approaches can be parallelized). The only part of the current single-core implementation which cannot be trivially parallelized is the sequential search and propagation loop of CPM. For parallel implementations, this issue can be resolved by switching to more elaborate PatchMatch variants such as, e.g., those based on the *jump flood scheme* [18], [54] which use search patterns that are amenable to parallelization.

The filter parameters employed are listed in Table II, and have been tuned on a few frames of the MPI Sintel training set [19]. Parameters for spreading sparse data have been validated on the MPI Sintel test set by means of the optical flow evaluation in Section 5-C, and work well for a variety of different sequences and applications. We set the fidelity term λ to 0.0 since it is not intended to be used with sparse data and can lead to discontinuities at the sparse sampling locations. For applications involving a creative process with artistic intent (such as, e.g. *Visual Saliency* and *Base/Detail Layer*, see Section 5-E), the selected filter parameters represent a possible choice and may be changed in order to smooth the images at different scales. The employed parameterization of the temporal filter provides a good tradeoff between oversmoothing and no smoothing at all, as explained in more detail later in Section 5-E. For CPM, we use the parameters as described in Section 4-B. The only varying parameter is the grid spacing d which we set according to the image resolution such that CPM yields approximately the same amount of flow vectors (~30-40K). For *Sintel* images (1024×436) we use $d = 3$ and for *'Tears of Steel'* (ToS) images (1920×800) we use $d = 6$.

TABLE II
EMPLOYED FILTER PARAMETERS.

Filter Parameter	Flow	Disparity	Saliency	Base/Detail Layer
σ^{XY}	0.017	0.017	0.025	0.025 / -
σ^{photo}	0.3	0.3	0.3	0.3
σ^{grad}	1.0	1.0	1.0	1.0
K	5	5	5	5 / 0
$\lambda^{\{XY,T\}}$	0	0	1	1
$\alpha^{\{XY,grad,photo\}}$	2	2	2	2

B. Performance of the BOOM Descriptor and CPM

BOOM Descriptor We first evaluate the BOOM descriptor on the three 100k datasets *Liberty*, *Notredame* and *Yosemite* provided by Brown et. al [43], where each contains 100k pairs of rotation- and scale-normalized 64×64 grayscale image patches. The pairs come with annotated ground truth, which allows to calculate performance measures in terms of ROC.

As mentioned in Section 4-A, the usage of orientation maps instead of HOGs, and the complete descriptor support normalization have a significant impact on the ROC performance. Figure 11a shows a comparison of the proposed BOOM descriptor and three variants thereof that either employ standard HOG binning with 8 directions instead of the orientation maps, or that use standard per 4×4 pixel cell normalization as described in [46]. Note the significant performance gain due to the usage of orientation maps and the normalization method.

The ROC characteristic of our BOOM descriptor is shown in Figure 11b, along with several other standard descriptor methods [10–16], [36]. With a 95 % error rate of 22-27.3 %, we can observe that it performs slightly better than SIFT and DAISY, and it performs significantly better than many other standard binary descriptors such as BRIEF, BRISK, FREAK and SKB. The novelty of BOOM lies in the combination of orientation maps with a normalized thresholding scheme, which allows to retain this performance in spite of binarization.

More detailed numerical results are provided in Table I, together with results from advanced learning-based descriptors such as Brown et al. [43], LDA-Hash [44] and AdaBoost-

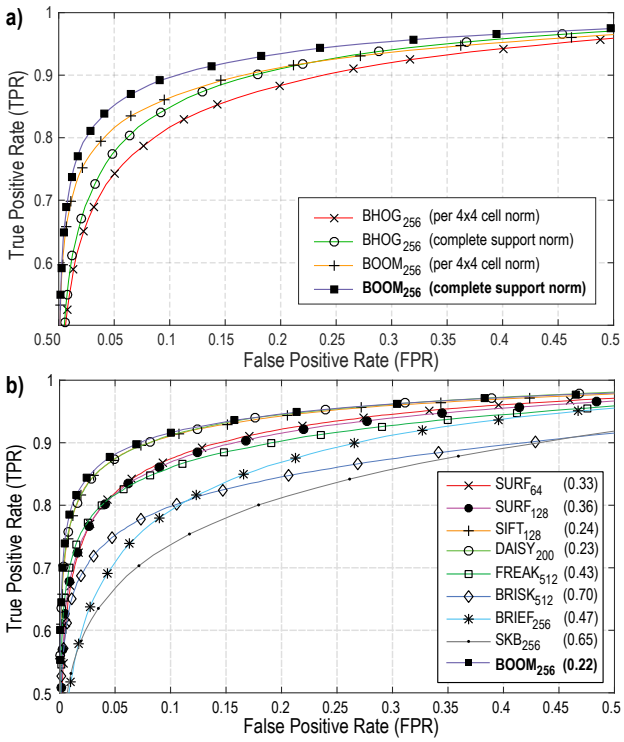

Fig. 11. ROC performance comparisons on the Notredame 100k UBC dataset [43]. a) comparison of BOOM and 3 variants that selectively use standard HOG binning instead of orientation maps, or standard per-cell HOG normalization instead of the proposed normalization (all variants use the same spatial layout). b) comparison of BOOM with several other standard methods.

based binary descriptors like BinBoost [39] and its simplified derivatives Bamboo [40] and LDDB-U [41]. Timings have been obtained by extracting 10k descriptors from the same image (including descriptor support scaling). We observe that in general, binary descriptors have much lower memory footprints with respect to the floating-point (FP) counterparts (assuming 1 B per dimension for FP methods [39]). Especially BinBoost achieves a very compact representation with just 8 bytes while ranking second in terms of 95 % error rates. However, it is very costly to compute (~1 ms per descriptor) as each effective descriptor dimension results in the computation of a linear superposition of 128 features. Therefore, this method is better suited for large-scale database retrieval where memory and matching efficiency matter most. As opposed to this, BOOM offers a competitive speed/performance tradeoff, since with an execution time of ~11 μs per descriptor, it is significantly more efficient than most other methods. At the same time it provides similar error rates as SIFT and DAISY. The run-time of the dense scan DAISY and SIFT-Flow methods (published MATLAB MEX C++ implementations) amounts to 1350 ms and 950 ms, respectively, for a 1024×436 pixel frame on a 3.1 GHz i7 machine. Compared to this, our unvectorized single-core dense scan implementation of BOOM only requires 111 ms, which is 12.3× faster than DAISY and 8.6× faster than SIFT-Flow.

CPM with BOOM Figure 12 shows a comparison of the sparse flow estimates delivered by the original CPM method (calculated using the code from [17] with standard parameters) and our CPM variant with BOOM. The corresponding

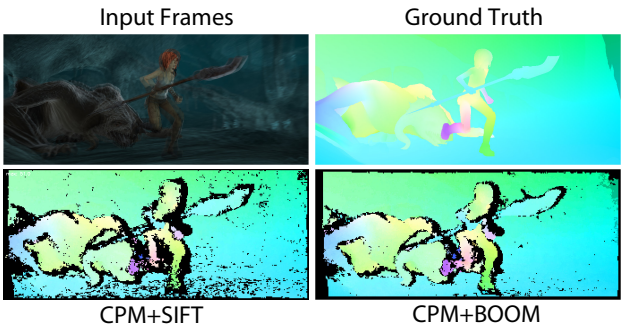

Fig. 12. Comparison of CPM+BOOM with CPM+SIFT from [17]. 'Sintel' images copyright © Blender Foundation, www.sintel.org.

numerical results using the metrics defined in [17] are shown in Table III. *SIFT-NN* are matches obtained with SIFT and FLANN [17], *Kd-tree PatchMatch* is a recent NNF method [55], *Deep Matching* is the matching method used in *DeepFlow* [33] and *EpicFlow* [32], and CPM+SIFT is the original CPM method [17]. We can see that CPM+BOOM performs similarly compared to CPM+SIFT in terms of density and precision. At the same time, it is more than twice as fast when run on a slower i7 machine (3.1 GHz) than the original (3.5 GHz). The decreased number of matches of CPM+BOOM when compared to CPM+SIFT is mainly due to the larger border region of our implementation and a stricter forward-backward consistency check. This decrease can be tolerated since the method provides sufficiently dense initializations.

C. Optical Flow and Comparison with Related Methods

Optical Flow Evaluation We evaluated the flow estimation part of our STEA pipeline on the MPI Sintel dataset [19], which is considered realistic and challenging and exhibits large motions and motion blur. It comprises 23 training and 12 test sequences with up to 49 frames, and since it is derived from an animation movie, accurate ground-truth is available for training. Table IV lists our evaluation results, together with results of the fastest methods (run times below 20 s per frame) published on the Sintel website. A visual comparison of selected methods is shown in Figure 14, and the run time vs. *average endpoint error* (AEE) tradeoff is shown in Figure 13.

As can be observed in Figure 14, our method provides visually similar quality as CPM-Flow [17], EpicFlow [32], and FlowFields [57] which however provide more accurate results in terms of AEE. The lowest AEE is achieved by CPM-Flow on the *clean pass* (3.56) and by FlowFields on the *final pass* (5.81). However, the run time of our unvectorized single-core implementation amounts to only 0.59 s for XYT filtered flow results and is therefore significantly faster than most other methods listed in Table IV. This fact is also reflected in the plot

TABLE III
COMPARISON OF CPM+BOOM WITH RELATED METHODS ON MPI SINTEL (FINAL TRAINING PASS). TIMINGS OF CPM+SIFT AND CPM+BOOM HAVE BEEN MEASURED, THE REMAINING ONES ARE FROM [17].

Method	Amount	Density	Precision	Time	Processor
SIFT-NN [10]	1K	0.175	0.851	0.5 s	i7 3.5GHz
KPM [55]	**446K**	**1.000**	0.595	**0.4 s**	i7 3.5GHz
DM [33]	5K	0.892	0.945	15 s	i7 3.5GHz
CPM+SIFT [17]	40K	0.886	0.975	2.1 s	i7 3.1GHz
CPM+BOOM	32.7K	0.834	**0.979**	0.56 s	i7 3.1GHz

TABLE IV
RESULTS AND COMPARISON WITH OTHER FAST STATE-OF-THE-ART METHODS ON THE MPI SINTEL DATASET [19]. TIMINGS TAKEN FROM [17].

Method	Clean Pass AEE	Clean Pass AEE Noc	Clean Pass AEE Occ	Final Pass AEE	Final Pass AEE Noc	Final Pass AEE Occ	Time [s]	Processor Model
DiscreteFlow [56]	3.57	1.11	23.63	6.08	2.94	31.69	~180 s	i7 3.5GHz (1 Thread)
SparseFlow [31]	6.20	2.36	37.46	7.85	3.86	40.40	10 s	?
DeepFlow [33]	5.38	1.77	34.75	7.21	3.34	38.78	19 s	i7 3.5GHz (1 Thread)
FlowFields [57]	3.75	**1.06**	25.70	**5.81**	**2.62**	31.80	18 s	i7 3.5GHz (1 Thread)
EpicFlow [32]	4.12	1.36	26.60	6.29	3.06	32.56	16.4 s	i7 3.5GHz (1 Thread)
CPM-Flow [17]	**3.56**	1.19	**22.89**	5.96	2.99	**30.18**	4.3 s	i7 3.5GHz (1 Thread)
PCA-Layers [34]	5.73	2.46	32.47	7.89	4.26	37.48	3.2 s	i7 3.5GHz (1 Thread)
PCA-Flow [34]	6.83	3.01	37.94	8.65	4.73	40.67	**0.18 s**	?
BOOM+Epic+Var	**3.56**	1.10	23.71	6.53	3.17	33.96	3.4 s	i7 3.1GHz (1 Thread)
BOOM+Epic	3.67	1.27	23.23	6.70	3.37	33.87	2.5 s	i7 3.1GHz (1 Thread)
BOOM+PF+Var (XY)	4.91	1.38	33.63	7.29	3.41	38.90	1.4 s	i7 3.1GHz (1 Thread)
BOOM+PF (XYT)	5.31	1.82	33.81	7.68	3.83	39.12	0.59 s	i7 3.1GHz (1 Thread)
BOOM+PF (XY)	5.20	1.70	33.83	7.60	3.73	39.14	0.54 s	i7 3.1GHz (1 Thread)

in Figure 13, which summarizes the quality/speed tradeoff of all compared methods. We can see that our BOOM+PF method provides a very competitive tradeoff, and shares the Pareto frontier with PCA-Flow, CPM-Flow and FlowFields which are optimal choices at different operating points. The only CPU-based method that is faster than BOOM+PF is PCA-Flow [34], which only requires 0.18 s. But as can be observed in Figure 14 at the bottom, PCA-Flow is not capable of capturing fine details and the resulting flow is blurry. It is interesting to note that all methods on the Pareto front (PCA-Flow, CPM-Flow and FlowFields) rely on similar concepts. They extract sparse point correspondences or NNFs and convert these to dense flow maps via interpolation. The superior accuracy of CPM-Flow and FlowFields is to some extent achieved by using the EpicFlow backend (interpolation and variational refinement). In addition, FlowFields employs a more elaborate matching technique than CPM-Flow that results in slightly lower AEE, at the price of a significantly longer execution time of 18 s.

To illustrate the quality-runtime tradeoffs due to the backend, we also integrated EpicFlow into our method, such that we can selectively switch the interpolation technique and/or add a variational refinement step. As can be seen in Table IV and Figure 13, the resulting variants (BOOM+...) are all on the Pareto front, offering a range of different quality/execution time tradeoffs. Note that these variants have not been tuned

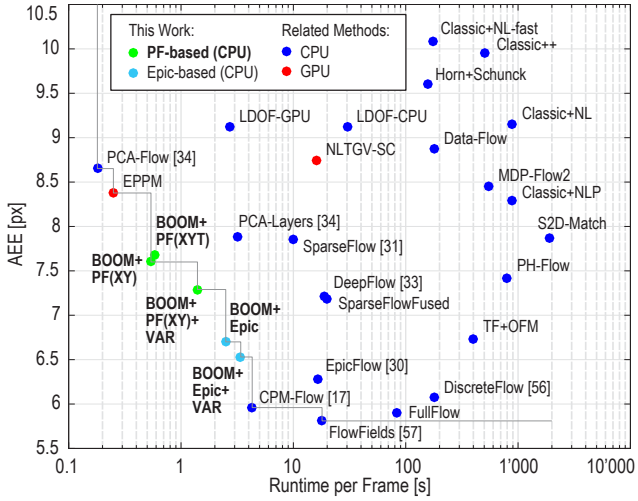

Fig. 13. Runtime vs. AEE plot of fast CPU implementations on the *final* MPI Sintel dataset. The Pareto front is indicated in gray.

and employ the default parameters. BOOM+EPIC+VAR has a similar AEE as CPM-Flow on the *clean pass*, as expected, and a slightly higher AEE on the *final pass*. Interestingly, the variational refinement step decreases the AEE by only around 0.4, when applied to BOOM+PF (XY). Larger AEE improvements in the order of 0.9 can be achieved by using EpicFlow interpolation instead. The reason for this is that EpicFlow employs local, affine transformations for interpolation, whereas our edge-aware filter performs an averaging operation. It should be noted, however, that the execution time of EpicFlow interpolation depends on the number of sparse matches, and amounts to around 2.1 s on average (including structured edge detection [58]). In contrast, the PF has constant runtime and is significantly faster (90 ms), which is an important property for real-time applications.

A visual comparison of all BOOM+... variants is given in Figure 14, and reveals that interpolation with the PF can preserve fine details a bit better than Epic interpolation, whereas Epic interpolation provides smoother object boundaries. This can be explained by the fact that the PF directly operates on pixel-wise permeabilities, whereas the Epic interpolation employs structured edges [58] that are smoother.

The AEE difference between the XY and the XYT filtered flows is a consequence of several small error sources that may result in a numerical offset. This includes warping artifacts, violations of the constant flow assumption, and the fact that an asymmetric causal filtering window is used, which can introduce a small temporal lag. Note however, that the numerical differences are small and that several of these artifacts are explicitly addressed (e.g., by using the flow gradient measure to stop filtering in regions with rapidly changing motion patterns). Further, the T step effectively removes temporal flickering as shown in more detail in Section 5-D.

Performance and Related STEA Methods Calculating optical flow for a Sintel frame (1024×436) takes 0.59 s (without file I/O), and most of the time is spent calculating the dense descriptor field (26 %) and performing CPM (50 %). The five spatial filtering iterations of the two flow components and the confidence map amount to 16 %, and the temporal step including warping to only 8 % of the total runtime. For this resolution, CPM requires a large fraction of the time. Note however, that the computational effort of CPM does not increase when scaling to higher resolutions, since we keep

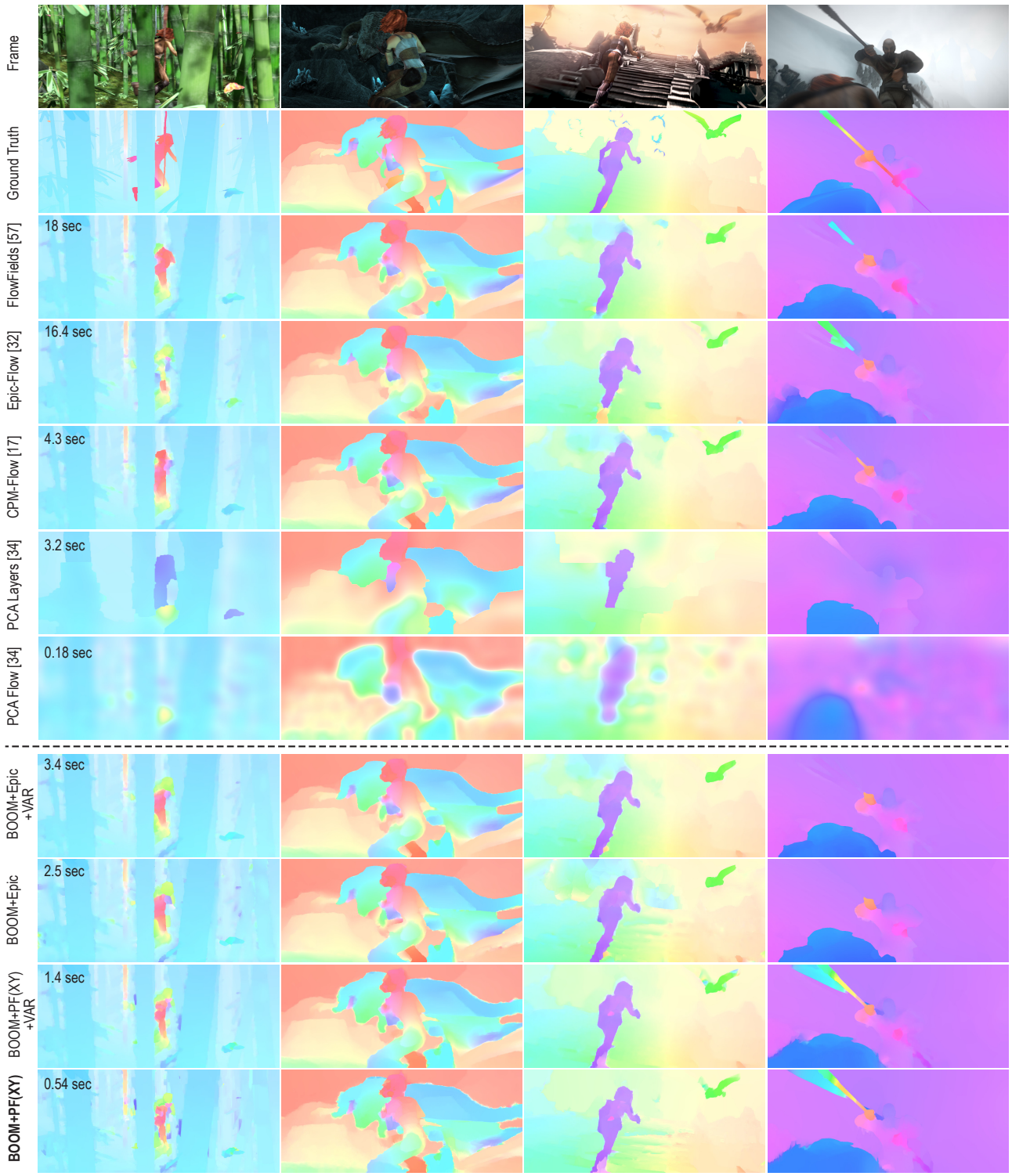

Fig. 14. Flow results and comparison of our method on the MPI Sintel dataset. 'Sintel' images copyright © Blender Foundation, www.sintel.org.

the amount of sparse grid sampling points constant. For larger frames such as in the ToS examples (1920×800) shown in the following subsections, the fraction spent in CPM reduces to 25 %. The runtime for one frame amounts to 1.41 s in that case, and the remaining fractions are 39 % for BOOM, 24 % for the XY iterations, and 11 % for the T step. The measured DRAM consumption of our method does not exceed 72 MB for Sintel and 245 MB for ToS frames. The generated DRAM traffic per frame has been profiled by observing last-level data cache misses, and amounts to ~500 MB and ~1.66 GB, respectively.[2]

The related STEA method by Aydın et al. [1] on which our spatial filter is based, has not been designed to calculate flow, and relies on a slow, high-quality variational method [30]. The STEA method by Lang et al. [6] is able to calculate optical flow, but has not been thoroughly evaluated on a large dataset like Sintel. In terms of run time, their optimized quad-core

[2]Typical embedded GPU platforms such as the Tegra X1 and X2 offer in the order of 25-50 GB/s of DRAM bandwidth [59].

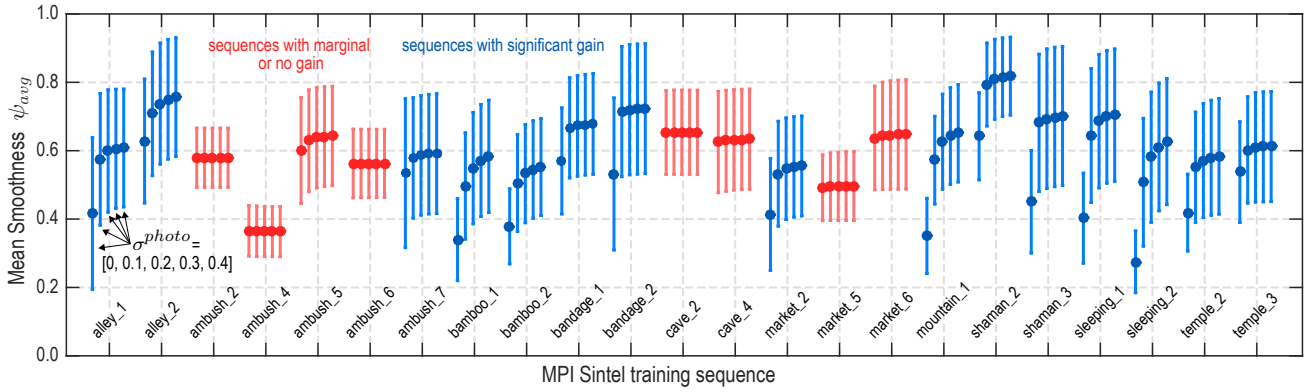

Fig. 15. Average smoothness score of all traces in the analyzed sequences for different photo constancy parameterizations σ^{photo} of the temporal filter (σ^{grad} is constantly set to 1.0 in this analysis.). The bars indicate the standard deviation. Note that there are several Sintel sequences, where the temporal filtering step improves the smoothness of the sequence (highlighted with blue). The sequences plotted in red color only show marginal improvements, which is due to the fact that they exhibit large and complex motion that inhibits filtering along motion paths.

C++ implementation requires 0.625 s on average to process a 640×480 pixel image including feature matching and 4 XYT iterations (measured on a 2.67 GHz i7 CPU). Our single-core implementation requires only 0.35 s for the same image including CPM+BOOM, 5 XY iterations and one T step. Note that the required amount of memory and the processing latency of the method by Lang et al. grows linearly with the number of frames, whereas our method has a constant latency and memory footprint. For example, their method requires 8.83 GB of DRAM to process 400 frames of 640×480 video, while our method works with less than 50 MB of DRAM.

D. Temporal Consistency

Smoothness Metric To assess the temporal filter performance, we define a *smoothness metric* ψ employing the lag-1 autocorrelation function to get a normalized smoothness score for a discrete value trace $\mathbf{tr}_p$ starting at pixel position p:

$$\psi\left(\mathbf{tr}_p\right) = 0.5 \cdot \mathrm{acf}\left(\mathrm{diff}\left(\mathbf{tr}_p\right), 1\right) + 0.5, \tag{18}$$

The differential operator diff acts as a low-pass filter and removes low-frequency components from the motion itself. The behavior of ψ is illustrated on a set of synthetic example traces in Figure 16, where we observe that indeed, smooth traces are assigned a value close to 1.0, and rough traces are assigned a value close to 0.0. To evaluate complete video sequences, we use the ground truth flow available in the MPI Sintel dataset to extract motion paths as illustrated in Figure 17. We start tracing at frame 1 and stop as soon as the trace is being occluded, resulting in traces with an average length of 21 samples. The average smoothness ψ_{avg} of a sequence is then computed as the average over all scores

$$\psi_{avg} = \frac{1}{|\Omega|} \cdot \sum_{p \in \Omega} \psi\left(\mathbf{tr}_p\right). \tag{19}$$

Evaluation We evaluate the impact of the temporal filtering step by assessing the smoothness of optical flow that has been filtered with different temporal filter settings. Figure 15 shows the mean and standard deviation of the smoothness metric ψ_{avg} evaluated on all sequences of the MPI Sintel training set. In this plot, the parameter σ^{photo} is varied from 0.0 to 0.4, whereas the parameter σ^{grad} is held constant at 1.0. As can be seen, the temporal filtering step leads to significant temporal smoothness gains in the majority of sequences (blue). However, there are a couple of sequences that do not show improvements (red). Note that these sequences are sequences with fast changing motion that is difficult to track. Our filter has been designed to avoid temporal filtering in regions with fast changing motion, since these regions are likely to contain motion artifacts, and they violate the constant flow assumption that is required for correct operation of our temporal filter.

The behavior of the temporal filter is analyzed further in Figure 18, where the mean smoothness improvement $\Delta\psi_{avg}$ has been plotted against the average flow gradient magnitude of each sequence. In this analysis, the parameter σ^{photo} is set to 0.3, and the parameter σ^{grad} is varied from 0.25 to 2.0. We can observe that the smoothness gain is well correlated

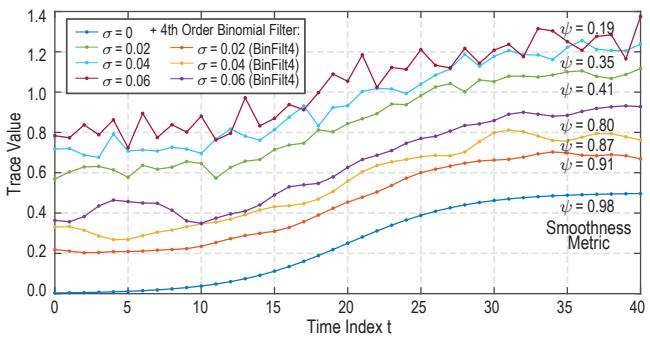

Fig. 16. Behavior of the smoothness metric ψ on a set of synthetic traces generated by superposition of a sigmoid with Gaussian noise with varying σ. Some traces have additionally been filtered using a 4th order Binomial filter. Note that ψ consistently assigns higher scores to smoother traces.

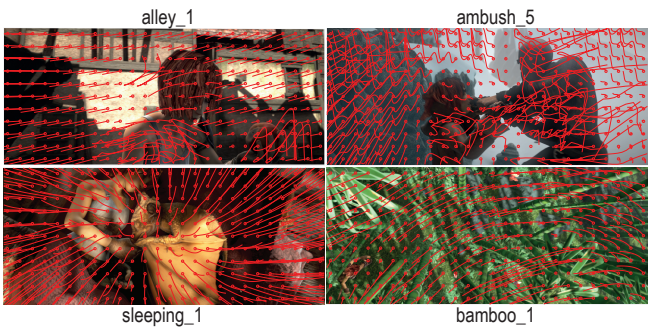

Fig. 17. Examples for motion paths extracted with the ground truth available in the Sintel training dataset. These paths are used to sample value traces from the filtered video cubes to assess their smoothness. Note that some scenes like *ambush_5* have complex motion patterns that are difficult to track.

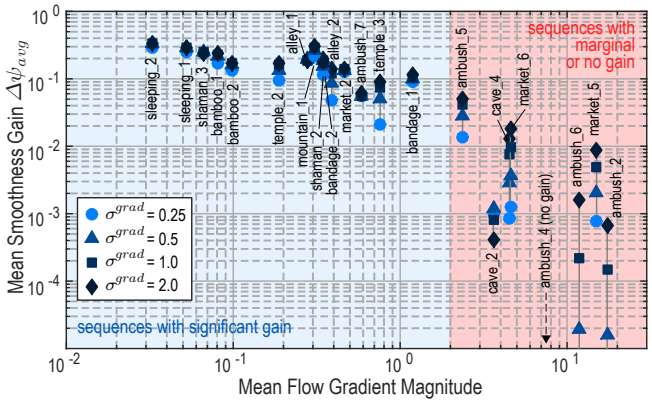

Fig. 18. This scatter plot shows the average smoothness gain vs. the mean flow gradient magnitude of the analyzed Sintel sequences. We can observe that the sequences with slowly changing motion show the highest smoothness improvements, which is expected due to the constant flow assumption.

with the average flow gradient magnitude. Sequences with fast changing motion are also more sensitive to the gradient measure parameter σ^{grad}, as expected.

In Figure 19, we show a subset of the value traces extracted from the *alley_1* and *sleeping_1* sequences (depicted in Figure 17) for qualitative assessment. As can be seen, temporal flickering noise is effectively removed, whereas large value changes are tracked accurately. It should be noted that the temporal filter shows a similar behavior when other feature maps such as saliency or disparity data is filtered.

E. Applications

In addition to optical flow, our STEA pipeline can be used for a variety of important image-based applications for which we present a selection of qualitative results in this section. See also http://iis.ee.ethz.ch/~michscha/stea/ for example videos.

Disparity Estimation Similarly to flow estimation, disparity estimation involves the computation of dense correspondences between image pairs. The main difference is the geometrically constrained stereoscopic setup, and the fact that both images are from the same time instant. Our STEA method can be used to produce temporally consistent disparity maps by initializing the additional channels with sparse point-correspondences, which are then first propagated and aligned with the input image, and then temporally smoothed in the T step. In Figure 21a and b we show two example sequences with video footage from the MPI Sintel stereo dataset. The sparse disparity initialization has been obtained with the same CPM+BOOM matching routine that has been used to calculate sparse optical flow vectors. The only modification is the restriction of the search space to horizontal displacements.

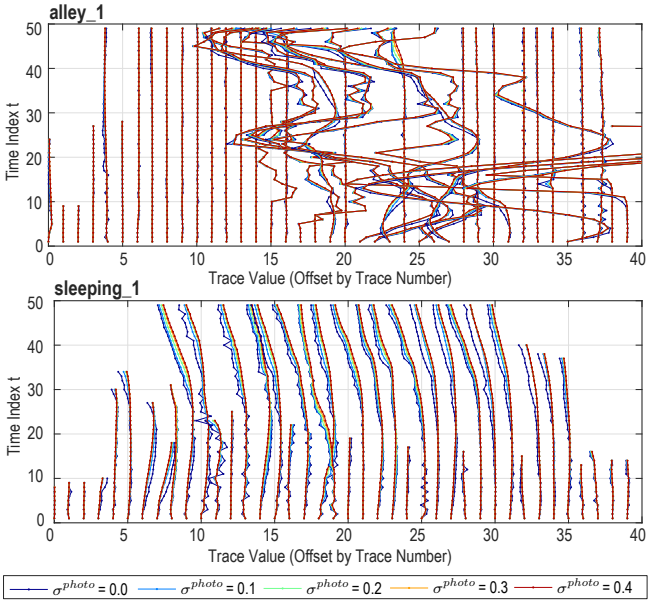

Fig. 19. The plot shows value traces extracted from the XYT filtered optical flow of the *alley_1* and *sleeping_1* sequences. The filter effectively removes temporal flickering noise, whereas large value changes are tracked accurately.

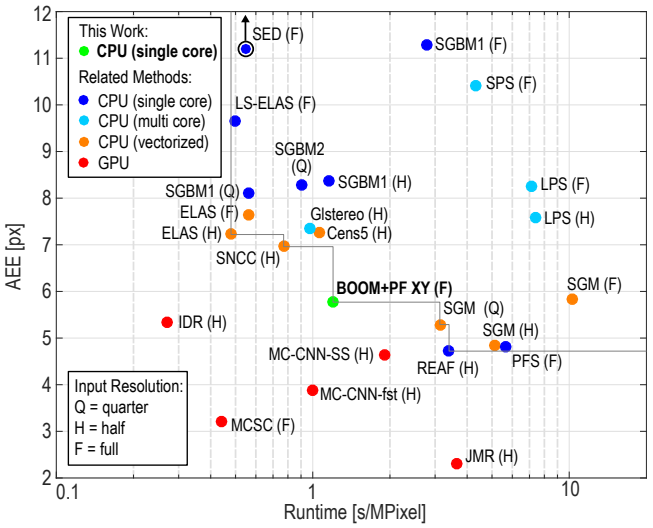

Fig. 20. AEE vs. runtime tradeoff on the *Middlebury3* stereo dataset (training). When excluding GPU accelerated disparity estimation methods, our BOOM+PF method is on the Pareto front.

Figure 20 shows a quantitative Pareto analysis of our method in terms of AEE and execution time on the *Middlebury3* stereo training set [60], including published results of the fastest methods (execution times below 10 s/MPixel). With an AEE of 5.77 px and an execution time of 1.3 s/MPixel, our BOOM+PF method offers a competitive quality/speed tradeoff also for disparity estimation.

Base/Detail Layer Decomposition Base/Detail layer decompositions obtained with edge-aware filters are powerful tools enabling HDR tone mapping methods [1], [3], [22] and detail manipulation methods [2], [22], [27]. In Figure 21c, we show one instance of such a method, where details of a video sequence are enhanced. Using our STEA filter, we compute the base layer by $\mathbf{B_t} = \mathbf{I_t^{XYT}}$, where $\mathbf{I_t}$ are the input video frames. A detail layer is then computed by subtracting this base layer from only temporally filtered input frames as $\mathbf{D_t} = \mathbf{I_t^T} - \mathbf{B_t}$. As described in [22], the detail layer is now boosted by multiplying each pixel by a factor (3 in the example shown) and applying a properly shifted and normalized sigmoid curve in order to avoid hard clipping. The enhanced image is then obtained by adding the boosted detail layer back to the base layer.

Visual Saliency A saliency map identifies the visually important regions in the image [61], and is an important feature for many image-based applications such as retargeting and multi-

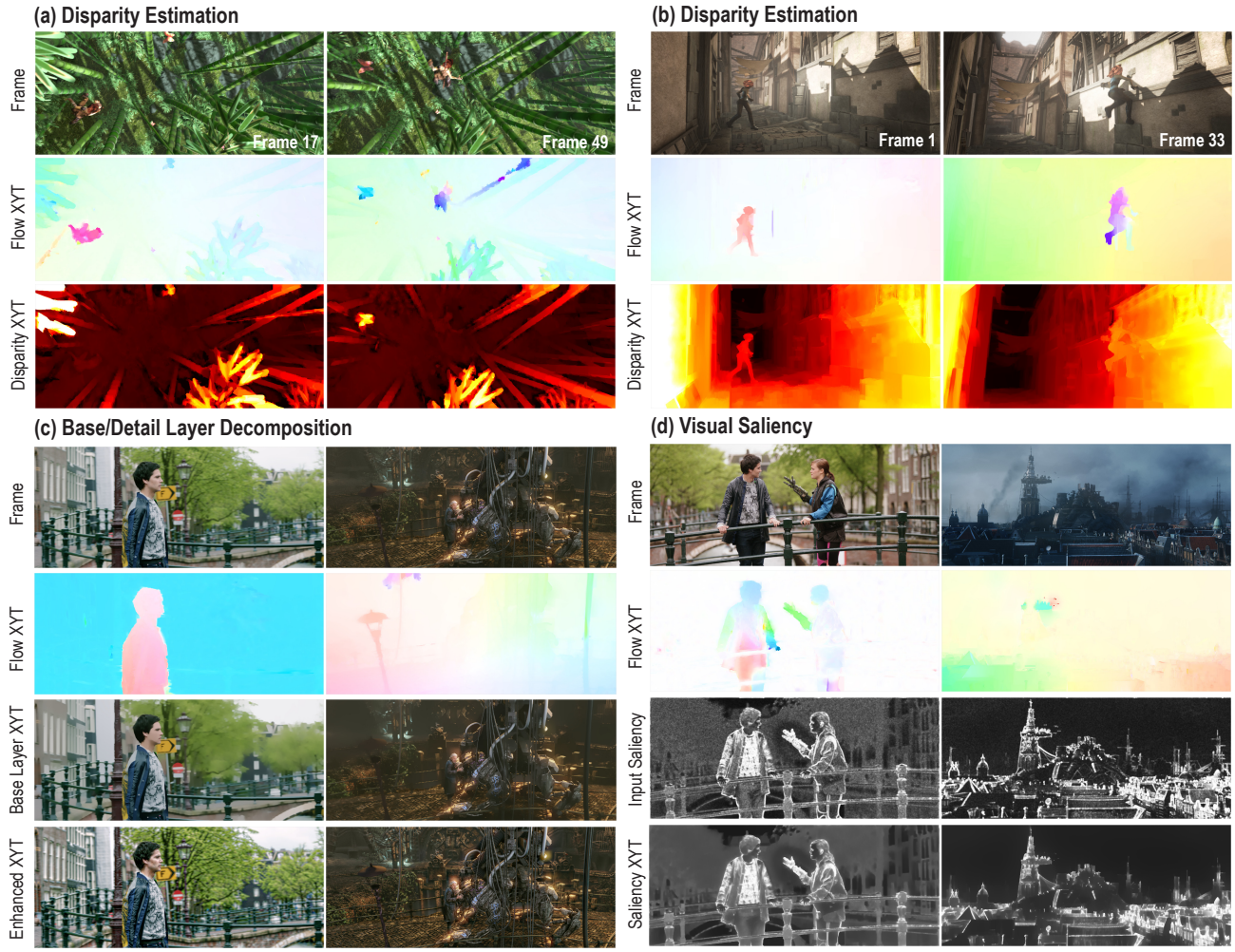

Fig. 21. a and b show disparity estimation results using our pipeline, and c show an example of a base-detail layer decomposition which is used to enhance image details in temporally consistent manner. See also http://iis.ee.ethz.ch/~michscha/stea/ for the corresponding videos. 'Sintel' images (a,b) copyright © Blender Foundation, www.sintel.org. 'Tears of Steel' images (c,d) copyright © Blender Foundation, mango.blender.org.

view rendering based in image domain warping (IDW) [62]. Efficient methods like [63] analyze the frequency spectrum of the image and since they operate on a per-frame basis, they often produce temporally noisy output. Our STEA pipeline can be used to stabilize and clean such noisy data, as show in Figure 21d, where the per-frame saliency Guo et al. [63] has been filtered with our STEA pipeline.

6. Conclusions

In summary, we have designed a STEA filtering pipeline which is more efficient than previous methods [1], [6] since the incremental PF formulation does not require large temporal neighbourhoods to be available in memory. Also, our pipeline does not depend on precomputed optical flow as, e.g., [1], since it is able to bootstrap its own optical flow on-the-fly by interpolating a quasi-dense NNF obtained with an improved patch match method. With a runtime of only 0.59 s per frame and an AEE of 7.683 px on the MPI Sintel dataset our method provides a competitive speed/quality tradeoff when compared to several other state-of-the-art optical flow methods.

Similar to existing STEA approaches [6], our method trades accuracy for computational efficiency, and is therefore not without limitations. The quality of the obtained results depends on the amount and quality of the input images and initial conditions. In case of sparse flow vectors, the proposed CPM+BOOM matching method alleviates this problem to a certain degree, since it provides accurate, quasi-dense initializations. However, similar to other matching methods it is dependent on the input image quality and the presence of noise and blur may lead to false correspondences. Our method can also fail when important object boundaries are not well represented by image edges – a limitation which applies to many other image-based applications and EA filters.

Despite these limitations, the presented pipeline offers an efficient way to apply EA filtering to video streams, without the requirement for large temporal windows. Apart from fast optical flow estimation, the pipeline can be used to create temporally consistent results for a variety of important image-based applications such as disparity and saliency estimation, and methods working on base/detail layer decompositions.

Compared to state-of-the-art methods, our STEA pipeline has low computational cost and constant memory requirements, making it suitable for embedded GPU or hardware implementations. E.g., it could be used to enhance temporal consistency of real-time video processing systems such as video retargeting and multiview synthesis engines [64], [65].

Acknowledgements

We would like to thank Tunç Aydın and Niko Stefanoski for helpful conversations and advice.

References

[1] T. O. Aydin, N. Stefanoski, S. Croci *et al.*, "Temporally Coherent Local Tone Mapping of HDR Video," *ACM TOG*, 2014.
[2] E. S. L. Gastal and M. M. Oliveira, "Domain transform for edge-aware image and video processing," *ACM TOG*, vol. 30, no. 4, 2011.
[3] F. Durand and J. Dorsey, "Fast Bilateral Filtering for the Display of High-dynamic-range Images," *ACM TOG*, pp. 257–266, Jul. 2002.
[4] A. Criminisi, T. Sharp, C. Rother *et al.*, "Geodesic Image and Video Editing," *ACM TOG*, vol. 29, no. 5, p. 134, 2010.
[5] C. Rhemann, A. Hosni, M. Bleyer *et al.*, "Fast Cost-Volume Filtering for Visual Correspondence and Beyond," in *IEEE CVPR*, 2011.
[6] M. Lang, O. Wang, T. Aydın *et al.*, "Practical Temporal Consistency for Image-Based Graphics Applications," *ACM TOG*, vol. 31, no. 4, 2012.
[7] G. Ye, E. Garces, Y. Liu *et al.*, "Intrinsic Video and Applications," *ACM TOG*, vol. 33, no. 4, pp. 80:1–80:11, Jul. 2014.
[8] N. Bonneel, J. Tompkin, K. Sunkavalli *et al.*, "Blind Video Temporal Consistency," *ACM TOG*, vol. 34, no. 6, 2015.
[9] C. Cigla and A. A. Alatan, "Information Permeability for Stereo Matching," *Signal Processing: Image Communication*, 2013.
[10] D. Lowe, "Distinctive Image Features From Scale-Invariant Keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.
[11] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up Robust Features," *ECCV*, pp. 404–417, 2006.
[12] M. Calonder, V. Lepetit, M. Ozuysal *et al.*, "BRIEF: Computing a Local Binary Descriptor Very Fast," in *IEEE TPAMI*, vol. 34, no. 7, 2012.
[13] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint," in *IEEE CVPR*, 2012, pp. 510–517.
[14] F. Zilly, C. Riechert, P. Eisert *et al.*, "Semantic Kernels Binarized - A Feature Descriptor for Fast and Robust Matching," in *CVMP*, 2011.
[15] E. Tola, V. Lepetit, and P. Fua, "A Fast Local Descriptor for Dense Matching," in *IEEE CVPR*, 2008, pp. 1–8.
[16] M. Schaffner, P. A. Hager, L. Cavigelli *et al.*, "A Complete Real-Time Feature Extraction and Matching System Based on Semantic Kernels Binarized," in *VLSI-SoC: At the Crossroads of Emerging Trends*. Springer Berlin Heidelberg, 2015.
[17] Y. Hu, R. Song, and Y. Li, "Efficient Coarse-to-Fine PatchMatch for Large Displacement Optical Flow," in *IEEE CVPR*, 2016.
[18] C. Barnes, E. Shechtman, A. Finkelstein *et al.*, "PatchMatch: a Randomized Correspondence Algorithm for Structural Image Editing," *ACM TOG*, vol. 28, no. 3, p. 24, 2009.
[19] D. J. Butler, J. Wulff, G. B. Stanley *et al.*, "A naturalistic open source movie for optical flow evaluation," in *ECCV*, 2012.
[20] P. Milanfar, "A tour of modern image filtering: New insights and methods, both practical and theoretical," *IEEE SPM*, Jan 2013.
[21] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV*, Jan 1998, pp. 839–846.
[22] Z. Farbman, R. Fattal, D. Lischinski *et al.*, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM TOG*, vol. 27, no. 3, pp. 67:1–67:10, Aug. 2008.
[23] R. Fattal, "Edge-avoiding wavelets and their applications," *ACM TOG*, vol. 28, no. 3, pp. 1–10, 2009.
[24] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE TPAMI*, vol. 35, no. 6, pp. 1397–1409, June 2013.
[25] E. S. Gastal and M. M. Oliveira, "High-Order Recursive Filtering of Non-Uniformly Sampled Signals for Image and Video Processing," in *Computer Graphics Forum*, vol. 34, no. 2, 2015, pp. 81–93.
[26] M. Aubry, S. Paris, S. W. Hasinoff *et al.*, "Fast Local Laplacian Filters: Theory and Applications," *ACM TOG*, vol. 33, no. 5, 2014.
[27] S. Paris, S. W. Hasinoff, and J. Kautz, "Local Laplacian Filters: Edge-Aware Image Processing With a Laplacian Pyramid," *ACM TOG*, vol. 30, no. 4, p. 68, 2011.
[28] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE TPAMI*, vol. 12, no. 7, Jul 1990.
[29] D. Sun, S. Roth, and M. J. Black, "Secrets of Optical Flow Estimation and their Principles," in *IEEE CVPR*, June 2010, pp. 2432–2439.
[30] H. Zimmer, A. Bruhn, and J. Weickert, "Optic Flow in Harmony," *IJCV*, vol. 93, no. 3, pp. 368–388, 2011.
[31] R. Timofte and L. Van Gool, "Sparse Flow: Sparse Matching for Small to Large Displacement Optical Flow," in *IEEE WACV*, 2015.
[32] J. Revaud, P. Weinzaepfel, Z. Harchaoui *et al.*, "EpicFlow: Edge-preserving Interpolation of Correspondences for Optical Flow," in *IEEE CVPR*, 2015, pp. 1164–1172.
[33] P. Weinzaepfel, J. Revaud, Z. Harchaoui *et al.*, "Deepflow: Large Displacement Optical Flow with Deep Matching," in *IEEE ICCV*, 2013.
[34] J. Wulff and M. J. Black, "Efficient Sparse-to-Dense Optical Flow Estimation Using a Learned Basis and Layers," in *IEEE CVPR*, 2015.
[35] C. Liu, J. Yuen, and A. Torralba, "SIFT Flow: Dense Correspondence across Scenes and its Applications," *IEEE TPAMI*, 2010.
[36] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary Robust Invariant Scalable Keypoints," in *IEEE ICCV*, 2011, pp. 2548–2555.
[37] W. Zhou, H. Li, M. Wang *et al.*, "Binary Sift: Towards Efficient Feature Matching Verification for Image Search," in *ICIMCS*, 2012.
[38] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *IEEE CVPR*, vol. 1, 2005, pp. 886–893.
[39] T. Trzcinski, M. Christoudias, and V. Lepetit, "Learning Image Descriptors with Boosting," *IEEE TPAMI*, vol. 37, no. 3, pp. 597–610, 2015.
[40] L. Baroffio, M. Cesana, A. Redondi *et al.*, "Bamboo: A Fast Descriptor Based on AsymMetric Pairwise BOOsting," in *IEEE ICIP*, 2014.
[41] X. Yang and K. T. Cheng, "Learning Optimized Local Difference Binaries for Scalable Augmented Reality on Mobile Devices," *IEEE TVCG*, vol. 20, no. 6, pp. 852–865, June 2014.
[42] T. Brox and J. Malik, "Large displacement optical flow: Descriptor matching in variational motion estimation," *PAMI*, vol. 33, no. 3, 2011.
[43] M. Brown, G. Hua, and S. Winder, "Discriminative Learning of Local Image Descriptors," *IEEE TPAMI*, vol. 33, no. 1, pp. 43–57, Jan 2011.
[44] C. Strecha, A. Bronstein, M. Bronstein *et al.*, "Ldahash: Improved matching with smaller descriptors," *IEEE TPAMI*, vol. 34, no. 1, 2012.
[45] Y. Yamauchi and H. Fujiyoshi, "Binary code-based Human Detection," in *CVIM*, 2012.
[46] B. Jun, I. Choi, and D. Kim, "Local transform features and hybridization for accurate face and human detection," *IEEE TPAMI*, vol. 35, no. 6, pp. 1423–1436, 2013.
[47] H. Fu, H. Zhao, X. Kong *et al.*, "BHoG: Binary Descriptor for Sketch-Based Image Retrieval," *Multimedia Systems*, vol. 22, no. 1, 2016.
[48] M. Zwicker, H. Pfister, J. V. Baar *et al.*, "EWA Splatting," *IEEE TVCG*, vol. 8, no. 3, pp. 223–238, 2002.
[49] P. Greisen, M. Schaffner, S. Heinzle *et al.*, "Analysis and VLSI Implementation of EWA Rendering for Real-Time HD Video Applications," *IEEE TCSVT*, vol. 22, no. 11, pp. 1577–1589, Nov 2012.
[50] H. Kaeslin, "Top-Down Digital VLSI Design, from VLSI Architectures to Gate-Level Circuits and FPGAs," *Morgan Kaufmann*, 2014.
[51] K. Zuiderveld, "Graphics gems iv," P. S. Heckbert, Ed. San Diego, CA, USA: Academic Press Professional, Inc., 1994, ch. Contrast Limited Adaptive Histogram Equalization, pp. 474–485.
[52] M. Calonder, V. Lepetit, C. Strecha *et al.*, "BRIEF: Binary Robust Independent Elementary Features," in *ECCV*, 2010.
[53] D. Nehab, A. Maximo, R. S. Lima *et al.*, "Gpu-efficient recursive filtering and summed-area tables," *ACM TOG*, vol. 30, no. 6, 2011.
[54] P. Yu, X. Yang, and L. Chen, "Parallel-friendly Patch Match Based on Jump Flooding," in *IFTC*, 2012, pp. 15–21.
[55] K. He and J. Sun, "Computing Nearest-Neighbor Fields via Propagation-Assisted kd-Trees," in *IEEE CVPR*. IEEE, 2012, pp. 111–118.
[56] M. Menze, C. Heipke, and A. Geiger, "Discrete Optimization for Optical Flow," in *GCPR*. Springer, 2015, pp. 16–28.
[57] C. Bailer, B. Taetz, and D. Stricker, "Flow Fields: Dense Correspondence Fields for Highly Accurate Large Displacement Optical Flow Estimation," in *IEEE ICCV*, 2015, pp. 4015–4023.
[58] P. Dollar and L. Zitnick, "Structured Forests for Fast Edge Detection," in *ICCV*, December 2013.
[59] A. Skende, "Introducing 'Parker' Next-Generation Tegra System-On-Chip," August, Hot Chips 2016.
[60] D. Scharstein, H. Hirschmüller, Y. Kitajima *et al.*, "High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth," in *GCPR*, 2014.
[61] L. Itti, C. Koch, and E. Niebur, "A Model of Saliency-based Visual Attention for Rapid Scene Analysis," *IEEE TPAMI*, vol. 20, 1998.
[62] N. Stefanoski, O. Wang, M. Lang *et al.*, "Automatic View Synthesis by Image-Domain-Warping," *IEEE TIP*, vol. 22, no. 9, 2013.
[63] C. Guo, Q. Ma, and L. Zhang, "Spatio-Temporal Saliency Detection Using Phase Spectrum of Quaternion Fourier Transform," in *IEEE CVPR*, June 2008, pp. 1–8.
[64] P. Greisen, M. Lang, S. Heinzle *et al.*, "Algorithm and VLSI Architecture for Real-time 1080P60 Video Retargeting," in *ACM EGGH-HPG*, 2012.
[65] M. Schaffner, F. K. Gürkaynak, P. Greisen *et al.*, "Hybrid ASIC/FPGA System for Fully Automatic Stereo-to-Multiview Conversion Using IDW," *IEEE TCSVT*, vol. 26, no. 11, Nov 2016.