Dense Point Trajectories by GPU-Accelerated Large Displacement Optical Flow*

Narayanan Sundaram, Thomas Brox, and Kurt Keutzer

University of California at Berkeley
{narayans,brox,keutzer}@eecs.berkeley.edu

Abstract. Dense and accurate motion tracking is an important requirement for many video feature extraction algorithms. In this paper we provide a method for computing point trajectories based on a fast parallel implementation of a recent optical flow algorithm that tolerates fast motion. The parallel implementation of large displacement optical flow runs about 78× faster than the serial C++ version. This makes it practical to use in a variety of applications, among them point tracking. In the course of obtaining the fast implementation, we also proved that the fixed point matrix obtained in the optical flow technique is positive semi-definite. We compare the point tracking to the most commonly used motion tracker - the KLT tracker - on a number of sequences with ground truth motion. Our resulting technique tracks up to three orders of magnitude more points and is 46% more accurate than the KLT tracker. It also provides a tracking density of 48% and has an occlusion error of 3% compared to a density of 0.1% and occlusion error of 8% for the KLT tracker. Compared to the Particle Video tracker, we achieve 66% better accuracy while retaining the ability to handle large displacements while running an order of magnitude faster.

1 Introduction

When analyzing video data, motion is probably the most important cue, and the most common techniques to exploit this information are difference images, optical flow, and point tracking. Since difference images restrict us to static cameras and we want to extract rich and unrestricted motion information, we will focus only on the last two techniques. The goal here is to enable accurate motion tracking for a large set of points in the video in close to real time and in this paper, we make substantial progress towards that goal. The quality of both the estimated flow field and the set of point trajectories are very important as small differences in the quality of the input features can make a high level approach succeed or fail. To ensure accuracy, many methods only track a sparse set of points; however, dense motion tracking enables us to extract information at a much finer granularity compared to sparse feature correspondences. Hence, one

* This work was supported by the German Academic Exchange Service (DAAD) and the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

K. Daniilidis, P. Maragos, N. Paragios (Eds.): ECCV 2010, Part I, LNCS 6311, pp. 438–451, 2010.

wants to use the most recent motion estimation technique providing the most reliable motion features for a specific task. For dense and accurate tracking there are usually computational restrictions. Video data processing requires far more resources than the analysis of static images, as the amount of raw input data is significantly larger. This often hinders the use of high-quality motion estimation methods, which are usually quite slow [1] and require expensive computer clusters to run experiments efficiently. For this reason, ways to significantly speedup such methods on commodity hardware are an important contribution as they enable more efficient research in fields that build upon motion features. This is important as more processing is usually required to utilize this motion information for use in video processing applications. Fast implementations of the KLT tracker and optical flow [2,3] are examples that have certainly pushed research.

In this paper we present a fast GPU implementation of large displacement optical flow (LDOF) [4], a recent variational optical flow method that can deal with faster motion than previous optical flow techniques[1]. The numerical schemes used in [4] and most variational methods are based on a coarse-to-fine warping scheme, where each level provides an update by solving a nonlinear system given by the Euler-Lagrange equations followed by fixed point iterations and a linear solver, as described in [5]. However, the relaxation techniques used in the linear solver that work best for serial processors are not efficient on parallel processors. We investigate alternative solvers that run well on parallel hardware, in particular, red-black relaxations and the conjugate gradient method. We show that the conjugate gradient method is faster than red-black relaxations, especially on larger images. We also prove that the fixed point matrix is positive semi-definite, thus guaranteeing the convergence of the conjugate gradient algorithm. We obtain a speedup of about 78×, which allows us to compute high quality LDOF for 640×480 images in 1.8 seconds. Extrapolating the current progress in GPU technology, the same code will even run in real-time in only a few years. While additional speedups are often obtained at the cost of lower quality, we ensured in our implementation that the quality of the original method is preserved.

We also propose a method for dense point tracking by building upon the fast implementation of large displacement optical flow. Point trajectories are needed whenever an approach builds on long term motion analysis. The dominant method used for this task is the KLT tracker [6], which is a sparse technique that only tracks a very small number of designated feature points. While for many tasks like camera calibration such sparse point trajectories are totally sufficient, other tasks like motion segmentation or structure-from-motion would potentially benefit from higher densities. In [1] and [7], a method for point tracking based on dense variational optical flow has been suggested. The method proposed in [1] is computationally very expensive and impractical to use on large datasets without acceleration.The point tracking we propose uses a similar technique, as points are propagated by means of the optical flow field; however, we do not build upon another energy minimization procedure that detects occluded points mainly by appearance, but do the occlusion reasoning by

[1] Executables and mex functions can be found at the authors' websites.

a forward-backward consistency check of the optical flow. In a quantitative comparison on some sequences from [8], where close to ground truth optical flow has been established by manually annotating the objects in the scene, we show that we can establish much denser point trajectories with better quality than the KLT tracker. At the same time, our method is more accurate and runs an order of magnitude faster than the technique in [7]. Such fast, high quality tracking will enable new applications such as better video summarization or activity recognition through improved tracking of limbs and balls in sports videos.

2 Related Work

Finding efficient solutions to variational optical flow problems has been an active area of research. On serial hardware, multi-grid solvers based on Gauss-Seidel have been proposed in [9]. A GPU implementation of the formulation in [9] has been proposed using Jacobi solvers [10]. Compared to [10], our implementation handles large displacements through dense descriptor matching. Such extensions enable us to handle fast motion well [11], [4]. A multi-grid red-black relaxation has been suggested in a parallel implementation of the linear CLG method [12]. Very efficient GPU implementations of other variational optical flow models have been proposed in [3,13,14].

The conjugate gradient algorithm is a popular solver for convex problems and has been used for optical flow problems with convex quadratic optimization [15]. In order to theoretically justify the use of conjugate gradients, we prove that the system matrix of general variational optical flow methods is positive semi-definite and thus the conjugate gradient solver is guaranteed to converge. It was previously proven that the Horn-Schunck matrix is positive definite [16]. Our proof is more general and applicable to most variational formulations [9], [5], [17] and [11].

The most popular point tracker is the Kanade-Lucas-Tomasi (KLT) tracker [6], which constructs an image pyramid, chooses points that have sufficient structure and tracks them across frames. New features are periodically detected to make up for the loss of features because of occlusions and tracking errors. This is generally considered to be fast and accurate, but it tracks only a few points. Efficient GPU implementations of the KLT tracker have been released in [18] and [2]. While the KLT algorithm itself is quite old, the implementation in [2] compensates for changes in camera exposure to make it more robust. Non-local point trackers that use global information have also been proposed [19].

The more advanced point tracker in [1] and [7] tracks points by building on top of a variational technique. This comes with high computational costs. It takes more than 100 seconds to track points between a pair of 720×480 frames. Moreover, this technique cannot deal with large displacements of small structures like limbs, and it has never been shown whether tracking based on variational flow actually performs better than the classic KLT tracker.

3 Large Displacement Optical Flow on the GPU

Large displacement optical flow (LDOF) is a variational technique that integrates discrete point matches, namely the midpoints of regions, into the continuous energy formulation and optimizes this energy by a coarse-to-fine scheme to estimate large displacements also for small scale structures [11]. As pointed out in [4], region matching can be replaced with matching other features like densely sampled histograms of oriented gradients (HOG) [20]. These simpler features allow us to implement both the variational solver and the discrete matching efficiently on the GPU.

The considered energy functional that is minimized reads:

$$\begin{aligned} E(\mathbf{w}) = &\int_\Omega \Psi_1\big(|I_2(\mathbf{x}+\mathbf{w}(\mathbf{x})) - I_1(\mathbf{x})|^2\big) + \gamma\,\Psi_2\big(|\nabla I_2(\mathbf{x}+\mathbf{w}(\mathbf{x})) - \nabla I_1(\mathbf{x})|^2\big)d\mathbf{x} \\ &+\beta \int_\Omega \delta(\mathbf{x})\ \rho(\mathbf{x})\ \Psi_3(|\mathbf{w}(\mathbf{x}) - \mathbf{w_1}(\mathbf{x})|^2)d\mathbf{x} + \int_\Omega \delta(\mathbf{x})\ |\mathbf{f_2}(\mathbf{x}+\mathbf{w_1}(\mathbf{x})) - \mathbf{f_1}(\mathbf{x})|^2 d\mathbf{x} \quad (1) \\ &+\alpha \int_\Omega \Psi_S\big(|\nabla u(\mathbf{x})|^2 + |\nabla v(\mathbf{x})|^2\big)d\mathbf{x} \end{aligned}$$

where $\mathbf{w} = (u\ \ v)^T$ and $\Psi_*(s^2)$ is a general penalizer function with its derivative $\Psi'_*(s^2) > 0$. A popular choice in the literature is $\Psi_*(s^2) = \sqrt{s^2 + \epsilon^2}$ [4].

Since speed and accuracy are foremost in solving the optical flow problem, it is necessary to take advantage of the improvements in modern microprocessors to aid the solution. In particular, parallelism has emerged as a key to performance scaling. Hence, it is necessary to study and develop algorithms and techniques that best utilize multiple processing elements simultaneously.

A parallel implementation of the descriptor matching is relatively straightforward since several points are being searched for in parallel without any dependencies between them. It is important, however, to take advantage of coalesced memory accesses (vector loads/stores) in order to maximize the performance of the GPU. In the rest of the section, we will focus on the parallel implementation of the variational solver that considers these point correspondences.

3.1 Variational Solver on the GPU

We minimize (1) by writing the Euler-Lagrange equations and solving them through a coarse-to-fine scheme with fixed point iterations. This results in a sequence of linear systems to be solved, where each pixel corresponds to two coupled equations in the linear system:

$$\begin{aligned} &(\Psi'_1 {I_x^k}^2 + \gamma\Psi'_2({I_{xx}^k}^2 + {I_{xy}^k}^2) + \beta\rho\Psi'_3)du^{k,l+1} + (\Psi'_2 I_x^k I_y^k + \gamma\Psi'_2(I_{xx}^k I_{xy}^k + I_{xy}^k I_{yy}^k))dv^{k,l+1} \\ &- \alpha\ \mathrm{div}(\Psi'_S \nabla(u^k + du^{k,l+1})) = -\Psi'_1(I_x^k I_z^k) + \gamma\Psi'_2(I_{xx}^k I_{xz}^k + I_{xy}^k I_{yz}^k) - \beta\rho\Psi'_3(u^k - u_1) \end{aligned} \quad (2)$$

$$\begin{aligned} &(\Psi'_1 {I_y^k}^2 + \gamma\Psi'_2({I_{yy}^k}^2 + {I_{xy}^k}^2) + \beta\rho\Psi'_3)dv^{k,l+1} + (\Psi'_2 I_x^k I_y^k + \gamma\Psi'_2(I_{xx}^k I_{xy}^k + I_{xy}^k I_{yy}^k))du^{k,l+1} \\ &- \alpha\ \mathrm{div}(\Psi'_S \nabla(v^k + dv^{k,l+1})) = -\Psi'_1(I_y^k I_z^k) + \gamma\Psi'_2(I_{yy}^k I_{yz}^k + I_{xy}^k I_{xz}^k) - \beta\rho\Psi'_3(v^k - v_1) \end{aligned}$$

For details on the derivation of these equation we refer to [4]. From symmetry considerations, the discretization usually produces a symmetric block pentadiagonal matrix with 2×2 blocks (for a 5-point Laplacian stencil). From equation (2), it is clear that only the diagonal blocks are dense, while the off-diagonal blocks are diagonal matrices. In fact, for the isotropic functionals we consider here, they are scaled identity matrices.

Positive semi-definiteness of the fixed point matrix. We have proven in [21] that the fixed point matrix is symmetric positive semi-definite because (a) the diagonal blocks are positive definite and (b) the matrix is block diagonally dominant [22]. The detailed proof is provided in [21]. An interesting takeaway from the proof is that it is not restricted to convex penalty functions Ψ_*. The only restriction on Ψ_* is that it should be increasing. Moreover, the proof technique generalizes to most variational optical flow methods, e.g. [5], [9],[11] and [17].

Linear solvers. On the CPU, the linear system is usually solved using *Gauss-Seidel* relaxations, which have been empirically shown to be very efficient in this setting [23]. The Gauss-Seidel method is guaranteed to converge if the matrix is symmetric positive definite. Unfortunately, the Gauss-Seidel technique is inherently sequential as it updates the points in a serial fashion. It is hard to parallelize it efficiently on multi-core machines and even harder on GPUs.

It is possible to choose relaxation methods that have slightly worse convergence characteristics, but are easy to parallelize, such as *Red-black relaxation* [24]. A single red-black relaxation consists of two half iterations - each half iteration updates every alternate point (called red and black points). The updates to all the red points are inherently parallel as all the dependencies for updating a red point are the neighboring black pixels and vice versa. Usually, this method is used with successive overrelaxation. Since we have a set of coupled equations, each relaxation will update (u_i, v_i) using a 2×2 matrix solve. Red-black relaxations have been used in a previous parallel optical flow solver [12].

Besides red-black relaxation, we consider the *Conjugate gradient* method. This requires symmetric positive definiteness as a necessary and sufficient condition for convergence. The convergence of the conjugate gradient technique depends heavily on the condition number of the matrix $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$. The condition numbers of the matrices obtained in the optical flow problems are very large and hence, convergence is usually slow.

A standard technique for improving convergence for ill-conditioned matrices is preconditioning to reduce the condition number of the system matrix. The preconditioner must be symmetric and positive definite. The special structure of the matrix allows for several regular pre-conditioners that work well in practice. In particular, we know that the diagonal blocks of the matrix are positive definite. Hence, a block diagonal matrix with only the diagonal blocks of the matrix is symmetric and positive definite and forms a good pre-conditioner. This preconditioner is usually referred to as a *block Jacobi preconditioner*. From now on, unless specified, we use the term conjugate gradient solver to refer to the preconditioned conjugate gradient solver with a block Jacobi preconditioner.

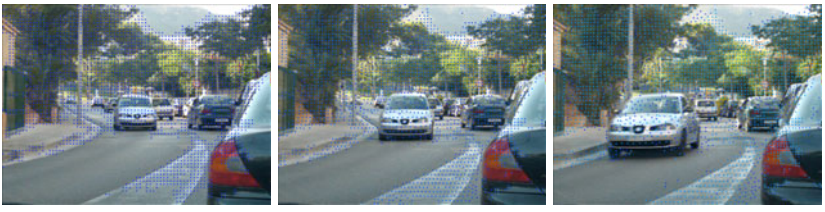


Fig. 1. Left: (a) Initial points in the first frame using a fixed subsampling grid. **Middle: (b)** Frame number 15 **Right: (c)** Frame number 30 of the cameramotion sequence. Figure best viewed in color.

Performing this algorithmic exploration is important as choosing the right algorithm for the right platform is essential for getting the best speed-accuracy tradeoff. This fast LDOF implementation can now be used to track points in video.

4 Point Tracking with Large Displacement Optical Flow

We demonstrate the utility of our LDOF implementation by suggesting a point tracker. In contrast to traditional local point trackers, like KLT [6], variational optical flow takes global smoothness constraints into account. This allows the tracking of far more points as the flow field is dense and tracking is not restricted to a few feature points. Moreover, large displacement optical flow enables tracking limbs or other fast objects more reliably than conventional trackers.

Our tracking algorithm works as follows: a set of points is initialized in the first frame of a video. In principle, we can initialize with every pixel, as the flow field is dense. However, areas without any structure are problematic for tracking with variational optical flow as well. For this reason, we remove points that do not show any structure in their vicinity as measured by the second eigenvalue λ_2 of the structure tensor

$$J_\rho = K_\rho * \sum_{k=1}^{3} \nabla I_k \nabla I_k^\top , \tag{3}$$

where K_ρ is a Gaussian kernel with standard deviation $\rho = 1$. We ignore all points where λ_2 is smaller than a certain portion of the average λ_2 in the image.

Depending on the application, one may actually be interested in fewer tracks that uniformly cover the image domain. This can be achieved by spatially subsampling the initial points. Fig. 1 shows a subsampling by factor 8. The coverage of the image is still much denser than with usual keypoint trackers.

Each of the points can be tracked to the next frame by using the optical flow field $\mathbf{w} := (u, v)^\top$:

$$(x_{t+1}, y_{t+1})^\top = (x_t, y_t)^\top + (u_t(x_t, y_t), v_t(x_t, y_t))^\top . \tag{4}$$

As the optical flow is subpixel accurate, x and y will usually end up between grid points. We use bilinear interpolation to infer the flow at these points.

The tracking has to be stopped as soon as a point gets occluded. This is extremely important, otherwise the point will share the motion of two differently moving objects. Usually occlusion is detected by comparing the appearance of points over time. In contrast, we detect occlusions by checking the consistency of the forward and the backward flow, which we found to be much more reliable. In a non-occlusion case, the backward flow vector points in the inverse direction as the forward flow vector: $u_t(x_t, y_t) = -\hat{u}_t(x_t + u_t, y_t + v_t)$ and $v_t(x_t, y_t) = -\hat{v}_t(x_t + u_t, y_t + v_t)$, where $\hat{\mathbf{w}}_t := (\hat{u}_t, \hat{v}_t)$ denotes the flow from frame $t + 1$ back to frame t. If this consistency requirement is not satisfied, the point is either getting occluded at $t+1$ or the flow was not correctly estimated. Both are good reasons to stop tracking this point at t. Since there are always some small estimation errors in the optical flow, we grant a tolerance interval that allows estimation errors to increase linearly with the motion magnitude:

$$|\mathbf{w} + \hat{\mathbf{w}}|^2 < 0.01 \left(|\mathbf{w}|^2 + |\hat{\mathbf{w}}|^2\right) + 0.5. \tag{5}$$

We also stop tracking points on motion boundaries. The exact location of the motion boundary, as estimated by the optical flow, fluctuates a little. This can lead to the same effect as with occlusions: a tracked point drifts to the other side of the boundary and partially shares the motion of two different objects. To avoid this effect we stop tracking a point if

$$|\nabla u|^2 + |\nabla v|^2 > 0.01\,|\mathbf{w}|^2 + 0.002. \tag{6}$$

In order to fill the empty areas caused by disocclusion or scaling, in each new frame we initialize new tracks in unoccupied areas using the same strategy as for the first frame.

5 Results

The implementation platform consists of an Intel Core2 Quad Q9550 processor running at 2.83GHz in conjunction with a Nvidia GTX 480 GPU. For the LDOF implementations almost all of the computation is done on the GPU and only minimal amount of data is transferred between the CPU and the GPU. We use Nvidia CUDA tools (v3.0) for programming the GPU. The CPU implementation is a well written hand coded serial C++ code that was vectorized using the Intel compiler with all the optimizations enabled.

For the tracking experiments, the KLT tracker used also runs on GPUs. A description of the algorithm is provided in [2]. The implementation in [2] also compensates for changes in camera exposure and provides real-time performance on the GPU considered. Default parameters were used unless otherwise specified.

5.1 GPU Accelerated Large Displacement Optical Flow

Runtime for large displacement optical flow has come down from 143 seconds for the previous serial implementation on CPU to 1.84 seconds for the parallel

implementation on GPU, a speedup of 78× for an image size of 640 × 480. This implementation searches for HOG matches in a neighborhood of ±80 pixels, uses $\eta = 0.95$, 5 fixed point iterations and 10 Conjugate gradient iterations to achieve the same overall AAE as the CPU version on the Middlebury dataset. It is also possible to run the optical flow algorithm at a slightly reduced accuracy (AAE increase of about 11%) at 1.1 seconds per frame. The performance of the linear solver is critical to the overall application runtime. Hence we look closely at the choice of the linear solver that enabled this speedup.

Performance of linear solvers. Figure 2 shows the convergence of different solvers for the optical flow problem. We measure convergence through the squared norm of the residual $||b - Ax^m||^2$. The rates of convergence are derived from 8 different matrices from images in the Middlebury dataset [25]. Red-black and Gauss-Seidel solvers use successive overrelaxation with $\omega = 1.85$. The matrices considered were of the largest scale (smaller scales show very similar results). The initial vector in all the methods was an all-zero vector. Using a better initialization procedure (the result of a previous fixed point iteration, for instance) also shows similar results.

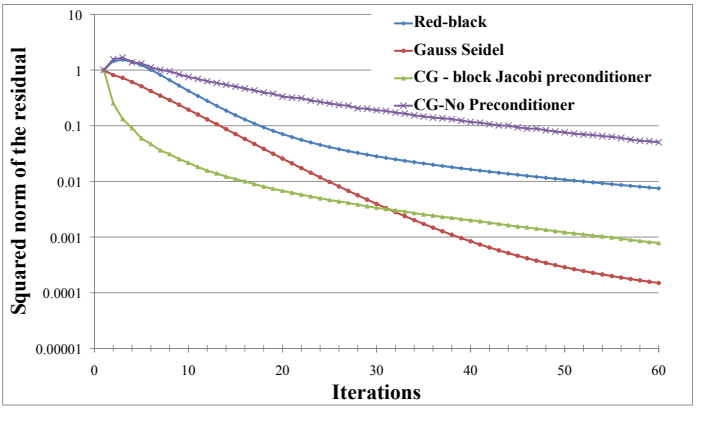
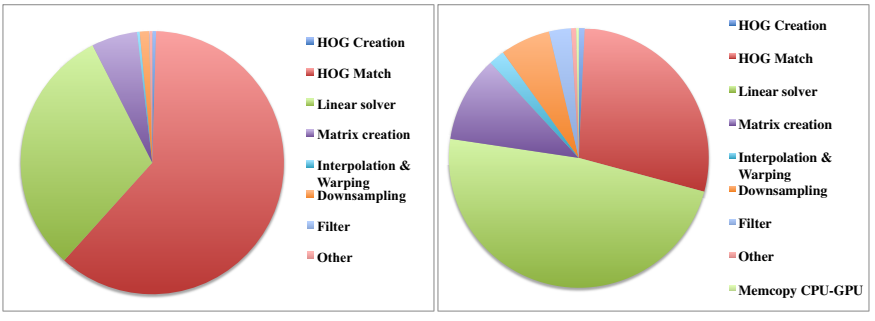


Fig. 2. Rates of convergence for different techniques considered. Y-axis shows the value of the residual normalized to the initial residual value averaged over 8 different matrices. Figure best viewed in color.

From Fig. 2, we can see why the Gauss-Seidel solver is the preferred choice for serial platforms. It converges well and is relatively simple to implement. In the numerical scheme at hand, however, we do not desire absolute convergence, as solving any one linear system completely is not important to the solution of the nonlinear system. It is more important to have a quick way of refining the solution and removing all the large errors. For a few iterations (30 or less), it is clear that the preconditioned conjugate gradient solver converges fastest. Non-preconditioned conjugate gradient is not as efficient because of the high condition number of the matrix.

Table 1. Average time taken by the linear solvers for achieving residual norm $< 10^{-2}$

Linear Solver	Time taken (in milliseconds)
Gauss-Seidel	395.13
Red-black	11.97
Conjugate Gradient	8.39



(a) Serial (Gauss Seidel on CPU) (b) Parallel (CG on GPU)

Fig. 3. Breakdown of execution times for serial and parallel variational optical flow solvers. Both solvers are run at a scale factor of 0.95, with 5 fixed point iterations and 25 Gauss-Seidel iterations/10 CG iterations to achieve similar AAE. Figure best viewed in color.

Although it is clear from Fig. 2 that conjugate gradient converges quickly in terms of the number of iterations required, a single iteration of conjugate gradient requires more computation than a Gauss-Seidel or a red-black iteration. Table 1 shows the runtimes of the solvers. Even though red-black relaxations are also parallel, we can see from Fig. 2 that we require roughly 3× as many red-black iterations as conjugate gradient iterations to achieve the same accuracy. Red-black iterations are 1.4× slower than CG overall. Gauss-Seidel iterations, running on the CPU, are 47× slower compared to conjugate gradient on the GPU.

Figure 3 shows the breakdown of the serial optical flow solver that uses Gauss-Seidel and the parallel solver that uses conjugate gradient. The solvers were run with $\eta = 0.95$, 5 fixed point iterations and 25 Gauss-Seidel iterations/10 Conjugate gradient iterations to achieve similar AAE on the Middlebury dataset. From both Figure 3(a) and 3(b), it is clear that the HOG matching and the linear solver are the most computation intensive components in the solvers. In both cases, they take more than 75% of the total runtime.

The major bottleneck in the conjugate gradient solver is the sparse matrix-vector multiply (SpMV). In order to optimize SpMV, the sparse matrix was laid out in memory as a set of images each corresponding to a particular non-zero diagonal. This, along with several other optimizations (caching in local scratch-pad memory, avoiding control overheads, ensuring vector loads/stores) enables the SpMV to run at 53 GFlops on the GPU. This is significant considering that

Table 2. Average Angular Error (in degrees) for images in the Middlebury dataset

Data	Dimetrodon	Grove2	Grove3	Hydrangea	RubberWhale	Urban2	Urban3	Venus	Average
AAE(CPU)	1.84	2.67	6.35	2.44	3.96	2.55	4.79	6.46	3.88
AAE(GPU)	1.84	2.51	5.94	2.37	3.91	2.47	5.43	6.38	3.86

the matrix is quite sparse (≤ 6 non-zeros per row). Under such conditions, most of the time in the kernel is spent fetching data to and from GPU main memory. Similar behavior is seen with the red-black relaxations, where 25% of the time is spent in floating point operations, while 75% of the time is spent in memory loads and stores. Red-black relaxations also have less computation to communication ratio (all the points are read, but only half the points are updated), which reduces their performance.

Accuracy. Table 2 shows the average angular error measured using our technique on the Middlebury dataset. These results have been achieved with the setting ($\gamma = 4$, β=30, $\alpha = 9$, $\eta = 0.95$, fixed point iterations = 5, Gauss-Seidel iterations = 25/CG iterations = 10). The data shows that the method provides similar accuracy to the CPU version while running fast on the GPU.

For faster computations, we use the parameter set ($\eta = 0.75$, 5 fixed point iterations, 10 linear solve iterations) to reduce the runtime by 38% with a degradation in AAE of 11%.

5.2 Tracking

We measure the accuracy of the tracking algorithms with the MIT sequences [8]. This dataset provides the ground truth optical flow for whole sequences and the sequences are much longer. This allows us to evaluate the accuracy of tracking algorithms. After obtaining the point trajectories from both KLT and LDOF, we track points using the given ground truth to predict their final destination. Tracking error is measured as the mean Euclidean distance between the final tracked position and the predicted position on the final frame according to the ground truth for all the tracked points. LDOF is run with $\eta = 0.95$, 5 fixed point iterations and 10 iterations for the linear solver in all the following experiments. Since the default parameters for the KLT tracker failed in tracking points in long sequences, we increased the threshold for positive identification of a feature from 1000 to 10000 (SSD_threshold parameter).

Accuracy. We compare the accuracy of the trackers for the entire length of the sequences. Since tracking algorithms should ideally track points over long times without losing points, we only consider those points that are tracked through all the frames. Trackers like KLT keep losing features and need to be constantly detecting new features every few frames to track well. From Table 3, it is clear that LDOF tracks almost three orders of magnitude more points than KLT with 46% improvement in overall accuracy. For tracking only the common points, the LDOF tracker is 32% better than KLT. These numbers exclude the fish sequence since it has transparent motion caused by dust particles moving in the water.

Table 3. Tracking accuracy of LDOF and KLT over the MIT sequences

Sequence name	Number of frames	All tracked points				Common points only			
		LDOF		KLT		LDOF		KLT	
		Mean error in pixels	Points tracked	Mean error in pixels	Points tracked	Mean error in pixels	Points tracked	Mean error in pixels	Points tracked
table	13	1.48	114651	3.78	363	1.04	293	1.39	293
camera	37	1.41	101662	3.78	278	1.01	185	2.64	185
fish	75	3.39	75907	35.62	106	3.12	53	5.9	53
hand	48	2.14	151018	3.11	480	1.87	429	2.39	429
toy	18	2.24	376701	2.88	866	1.70	712	1.89	712

Table 4. Tracking accuracy of LDOF and the Particle Video tracker over the 20 sequences used in [7]

Average number of frames	All tracked points				Common points only			
	LDOF		Particle Video		LDOF		Particle Video	
	Mean error in pixels	Points tracked	Mean error in pixels	Points tracked	Mean error in pixels	Points tracked	Mean error in pixels	Points tracked
61.5	0.83	109579	3.20	8967	0.84	3304	2.51	3304

Table 5. Occlusion handling by KLT and LDOF trackers based on region annotation from the MIT data set. Occluded tracks indicate tracks that are occluded according to the ground truth data, but not identified as such by the trackers.

Sequence	KLT		LDOF		
	Number of occluded tracks	Mean error with no occlusion	Number of occluded tracks	Mean error with no occlusion	Tracking Density (%)
table	11	2.73	853	1.41	39.6
camera	8	3.68	558	1.37	39.9
fish	30	31.79	8321	2.7	53.0
hand	10	2.90	2127	1.81	46.8
toy	31	2.58	5482	2.11	61.4

Although we were able to track this sequence well, performance on this sequence is sensitive to parameter changes.

Compared to the Particle Video point tracker in [7], our tracker is 66% more accurate for the common tracks. Since ground truth data does not exist for the sequences used in [7], it is not possible to have objective comparisons on metrics other than the average round trip error (The videos are mirrored temporally, so all unoccluded pixels should return to their starting positions). For comparison, we use only the full-length particle trajectories provided by the authors of [7] at http://rvsn.csail.mit.edu/pv/data/pv. The average statistics of both trackers over all the 20 sequences used in [7] are given in Table 4. More details on the comparison can be found in [21].

Occlusion handling. We use the region annotation data from the MIT dataset to measure the occlusion handling capabilities of the algorithms. The LDOF

Table 6. Tracking accuracy of LDOF and KLT for large displacements in the tennis sequence with manually marked correspondences. Numbers in parentheses indicate the number of annotated points that were tracked.

Frames	LDOF		KLT	
	Mean error in pixels	Points tracked on player	Mean error in pixels	Points tracked on player
490-495	2.55 (22)	8157	3.21 (19)	21
490-500	2.62 (8)	3690	4.12 (4)	4

Fig. 4. (Top) Frame 490 of the tennis sequence with (left) actual image, (middle) KLT points and (right) LDOF points. (Bottom) Frame 495 of the sequence with (left) actual image, (middle) KLT points and (right) LDOF points. Only points on the player are marked. KLT tracker points are marked larger for easy visual detection. Figure best viewed in color.

tracker has an occlusion error of 3% (tracks that drift between regions/objects) while the KLT tracker has an occlusion error of 8%. Given that KLT tracker is already very sparse, this amounts to a significant number of tracks that are not reliable (they do not reside on the same object for the entire time). After excluding all the tracks that were known to have occlusion errors, LDOF outperforms KLT by 44%. Since all the ground truth occlusions are known, we measure the tracking density (% of unoccluded points that the tracker was successful in tracking through the entire sequence without any occlusion errors). The LDOF tracker has an average tracking density of 48%, i.e., it tracks roughly half of the points that are not occluded for the entire length of the sequence, while KLT has a density of about 0.1%. Table 5 presents the data on occlusion handling.

Large displacements. The MIT sequences still mostly contain small displacements and hence KLT is able to track them well (if it does not lose the features); however, there are motion sequences with large displacements that are difficult for a tracker like KLT to capture. In the tennis sequence [11], there are frames

where the tennis player moves very fast producing motion that is hard to capture through simple optical flow techniques. Since ground truth data does not exist for this sequence we manually labeled the correspondences for 39 points on the player between frames 490, 495 and 500 [2]. These points were feature points identified by KLT in frame 490. The results for the points tracked on the player are shown in Table 6 and Figure 4. It is clear that the LDOF tracker tracks more points with better accuracy, while capturing the large displacement of the leg.

Runtime. The cameramotion sequence with 37 frames of size 640×480, requires 135 seconds. Out of this, 125 seconds were spent on LDOF (both forward and backward flow). Such runtimes allow for convenient processing of large video sequences on a single machine equipped with cheap GPU hardware.

6 Conclusion

Fast, accurate and dense motion tracking is possible with large displacement optical flow (LDOF). We have provided a parallel version of LDOF that achieves a speedup of 78× over the serial version. This has been possible through algorithmic exploration for the numerical solvers and an efficient parallel implementation of the large displacement optical flow algorithm on highly parallel processors (GPUs). Moreover, we have proposed a dense point tracker based on this fast LDOF implementation. Our experiments quantitatively show for the first time that tracking with dense motion estimation techniques provides better accuracy than KLT feature point tracking by 46% on long sequences and better occlusion handling. We also achieve 66% better accuracy than the Particle Video tracker. Our point tracker based on LDOF improves the density by up to three orders of magnitude compared to KLT and handles large displacements well, thus making it practical for use in a wide variety of video applications such as activity recognition or summarization in sports videos which require improved tracking of fast moving objects like balls and limbs.

References

1. Sand, P., Teller, S.: Particle video: Long-range motion estimation using point trajectories. International Journal of Computer Vision 80, 72–91 (2008)
2. Zach, C., Gallup, D., Frahm, J.M.: Fast gain-adaptive KLT tracking on the GPU. In: CVPR Workshop on Visual Computer Vision on GPU's, CVGPU (2008)
3. Zach, C., Pock, T., Bischof, H.: A duality based approach for realtime TV-L1 optical flow. In: Hamprecht, F.A., Schnörr, C., Jähne, B. (eds.) DAGM 2007. LNCS, vol. 4713, pp. 214–223. Springer, Heidelberg (2007)
4. Brox, T., Malik, J.: Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence (to appear, 2010)
5. Brox, T., Bruhn, A., Papenberg, N., Weickert, J.: High accuracy optical flow estimation based on a theory for warping. In: Pajdla, T., Matas, J(G.) (eds.) ECCV 2004. LNCS, vol. 3024, pp. 25–36. Springer, Heidelberg (2004)

[2] The manually labeled correspondence data can be found on the authors' website.

6. Shi, J., Tomasi, C.: Good features to track. In: CVPR, pp. 593–600 (1994)
7. Sand, P., Teller, S.: Particle video: Long-range motion estimation using point trajectories. In: CVPR (2006)
8. Liu, C., Freeman, W.T., Adelson, E.H., Weiss, Y.: Human-assisted motion annotation. In: CVPR (2008)
9. Bruhn, A., Weickert, J.: Towards ultimate motion estimation: Combining highest accuracy with real-time performance. In: ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05), Washington, DC, USA, vol. 1, pp. 749–755. IEEE Computer Society, Los Alamitos (2005)
10. Grossauer, H., Thoman, P.: GPU-based multigrid: Real-time performance in high resolution nonlinear image processing. In: Gasteratos, A., Vincze, M., Tsotsos, J.K. (eds.) ICVS 2008. LNCS, vol. 5008, pp. 141–150. Springer, Heidelberg (2008)
11. Brox, T., Bregler, C., Malik, J.: Large displacement optical flow. In: CVPR (2009)
12. Gwosdek, P., Bruhn, A., Weickert, J.: High performance parallel optical flow algorithms on the Sony Playstation 3. Vision, Modeling and Visualization, 253–262 (2008)
13. Wedel, A., Pock, T., Zach, C., Bischof, H., Cremers, D.: An improved algorithm for TV-L1 optical flow. In: Statistical and Geometrical Approaches to Visual Motion Analysis: International Dagstuhl Seminar, Dagstuhl Castle, Germany, July 13-18, pp. 23–45 (2009), Revised Papers
14. Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., Bischof, H.: Anisotropic Huber-L1 optical flow. In: Proc. of the British Machine Vision Conference, BMVC (2009)
15. Lai, S.H., Vemuri, B.C.: Reliable and efficient computation of optical flow. International Journal of Computer Vision 29 (1998)
16. Mitiche, A., Mansouri, A.R.: On convergence of the Horn and Schunck optical-flow estimation method. IEEE Transactions on Image Processing 13, 848–852 (2004)
17. Bruhn, A., Weickert, J., Schnörr, C.: Lucas/Kanade meets Horn/Schunck: combining local and global optic flow methods. Int. J. Comput. Vision 61, 211–231 (2005)
18. Sinha, S.N., Frahm, J.M., Pollefeys, M., Genc, Y.: Feature tracking and matching in video using programmable graphics hardware. Machine Vision and Applications (2007)
19. Birchfield, S.T., Pundlik, S.J.: Joint tracking of features and edges. In: CVPR (2008)
20. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR (2006)
21. Sundaram, N., Brox, T., Keutzer, K.: Dense point trajectories by GPU-accelerated large displacement optical flow. Technical Report UCB/EECS-2010-104, EECS Department, University of California, Berkeley (2010)
22. Feingold, D.G., Varga, R.S.: Block diagonally dominant matrices and generalizations of the Gerschgorin circle theorem. Pacific J. Math. 12, 1241–1250 (1962)
23. Bruhn, A.: Variational Optic Flow Computation: Accurate Modelling and Efficient Numerics. PhD thesis, Faculty of Mathematics and Computer Science, Saarland University, Germany (2006)
24. Stüben, K., Trottenberg, U.: Multigrid methods: Fundamental algorithms, model problem analysis and applications. Lecture Notes in Mathematics, vol. 960. Springer, Heidelberg (1982)
25. Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M., Szeliski, R.: A database and evaluation methodology for optical flow. In: ICCV (2007)