Real-Time Motion Estimation and Visualization on Graphics Cards

Robert Strzodka*
caesar research center
53044 Bonn, Germany

Christoph S. Garbe †
Interdisciplinary Center for Scientific Computing
69120 Heidelberg,Germany

Abstract

We present a tool for real-time visualization of motion features in 2D image sequences. The motion is estimated through an eigenvector analysis of the spatiotemporal structure tensor at every pixel location. This approach is computationally demanding but allows reliable velocity estimates as well as quality indicators for the obtained results. We use a 2D color map and a region of interest selector for the visualization of the velocities. On the selected velocities we apply a hierarchical smoothing scheme which allows the choice of the desired scale of the motion field. We demonstrate several examples of test sequences in which some persons are moving with different velocities than others. These persons are visually marked in the real-time display of the image sequence. The tool is also applied to angiography sequences to emphasize the blood flow and its distribution.

An efficient processing of the data streams is achieved by mapping the operations onto the stream architecture of standard graphics cards. The card receives the images and performs both the motion estimation and visualization, taking advantage of the parallelism in the graphics processor and the superior memory bandwidth. The integration of data processing and visualization also saves on unnecessary data transfers and thus allows the real-time analysis of 320x240 images. We expect that on the upcoming generation of graphics hardware our tool will run in real time for the standard VGA format.

CR Categories: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Motion, Time-varying imagery; G.1.3 [Numerical Analysis]: Numerical Linear Algebra—Eigenvalues and eigenvectors; I.3.8 [Computer Graphics]: Applications;

Keywords: motion estimation, motion visualization, structure tensor, eigenvector analysis, real-time processing, graphics hardware

1 Introduction

For the estimation of motion from digital image sequences a number of different techniques has been proposed [Barron, J. L. et al. 1994; Beauchemin, S. S. and Barron, J. L. 1995; Haußecker, H. and Spies, H. 1999]. For real time applications, feature tracking algorithms are widely in use [Wu 1989; Camus 1997]. While these approaches offer real time performance, estimated velocity fields are sparse. Also, inherent to these techniques is a reduced accuracy [Liu, S.-Z. et al. 1998], not making them ideal candidates for applications in which the precise estimation of motion is required.

Estimating motion patterns from gradient based optical flow techniques offer a number of advantages. Generally, these techniques are highly accurate [Barron, J. L. et al. 1994] and provide dense estimates. Another important property is the computation of confidence measures and type measures, indicating the quality of the estimates and problematic regions. Both measures are given by gradient based techniques with almost no additional computational cost. Due to these advantages this type of estimator for optical flow was chosen in the context of this work.

The computation of dense motion fields for an image sequence requires high processing power. Parallel computers and different hardware architectures have been considered to accelerate these computations [Kohlberger et al. 2003; Zuloaga et al. 1998; Maya-Rueda and Arias-Estrada 2003]. We meet the real-time requirements by exploiting the stream architecture of graphics cards. Graphics cards are not a cure-all for performance critical applications. They have traditionally been optimized for high data throughput and subscribe to a different computing paradigm than microprocessors, resulting in an inherent advantage for operations on large data streams. The concept they follow is not new, but equivalent processing power has not been previously available in such relatively inexpensive standard hardware products. Consequently, our tool does not aim for the ultimate performance on the best suited architecture but wants to demonstrate that a simple camera and a PC with a powerful graphics card suffice for the real-time motion estimation and visualization of image sequences.

Because of its outstanding price-performance ratio, graphics hardware has already been considered for the implementation of various general computing problems. We refer to [GPGPU] for a comprehensive overview. We are the first to address motion estimation on graphics cards, but individual parts of our algorithm are related to other work in this area, such as filtering [Hopf and Ertl 1999; Hadwiger et al. 2002; Colantoni et al. 2003], linear algebra operations [Goodnight et al. 2003; Bolz et al. 2003; Krueger and Westermann 2003], visualization [Colantoni et al. 2003; van Wijk 2002; Weiskopf et al. 2003], adaptive hardware techniques [Lefohn et al. 2003; Strzodka and Telea 2004; Coombe et al. 2004].

Along with the increasing number of CCTV cameras literature on video surveillance has grown rapidly [Chellappa(Ed.) 2000; IEE 2003; Kittler and Nixon 2003]. In contrast to most other contributions we concentrate on the real-time visual emphasis of the motion field with standard hardware components, assuming a complex motion pattern in the scenes, which defeats simple tracking or classification of individual activities. This is also orthogonal to [Daniel and Chen 2003], where an efficient 3D visualization most suitable for a compact summary of isolated motion events has been presented. Concerning the angiography sequences research focuses mainly on the segmentation of the vascular system [Kirbas and Quek 2002]. We operate in real-time directly on the images similar to [Brodsky and Block 2003], whereas in a post-processing step a much more detailed analysis can be obtained [Watanabe et al. 2002].

*e-mail: strzodka@caesar.de
†e-mail: Christoph.Garbe@iwr.uni-heidelberg.de

2 Motion Estimation

We quickly review the gradient based optical flow method we use and describe on the algorithmic level the computations we perform.

2.1 Optical Flow

A very common assumption in computations of image velocity is the brightness change constraint equation (BCCE) [Horn, B. K. P. and Schunk, B. 1981]. It states that the image brightness $g(\vec{x},t)$ at the location $\vec{x} = (x_1, x_2)^\top$ should change only due to motion, i.e. the total derivative of its brightness has to vanish [Fennema, C. and Thompson, W. 1979]:

$$\frac{dg}{dt} = \frac{\partial g}{\partial t} + \frac{\partial g}{\partial x}\frac{dx}{dt} + \frac{\partial g}{\partial y}\frac{dy}{dt} = g_t + (\vec{f}\vec{\nabla})g = d^\top \cdot p = 0, \quad (1)$$

with the optical flow $\vec{f} = (dx/dt, dy/dt)^\top = (u,v)^\top$, the spacial gradient $\vec{\nabla}g$ and the partial time derivative $g_t = \partial g/\partial t$. The data vector $\vec{d}$ is given by $\vec{d} = [g_x, g_y, g_t]^\top$ and the parameter vector by $\vec{p} = [u,v,1]^\top$.

Equation (1) poses an under-determined system of equations, as there is only one constraint with the two unknowns of the optical flow vector $\vec{f}$. Assuming constant optical flow over a small spatio-temporal neighborhood surrounding the location of interest containing m pixels (for optical flow [Lucas, B. and Kanade, T. 1981] and [Campani, M. and Verri, A. 1990]), the problem consists of m equations of the form of Equation (1). With the data matrix $\vec{D} = (\vec{d_1}, \ldots, \vec{d_m})^\top$ the total least squares problem can be reformulated as the structure tensor [Knutsson 1989; Bigün, J. et al. 1991; Haußecker, H. and Spies, H. 1999], that is

$$\begin{aligned} ||\vec{D}\vec{p}||_2 &= \int_{-\infty}^{\infty} w(\vec{x}-\vec{x'}, t-t') \left(\vec{p}^\top \vec{D}^\top \vec{D}\vec{p}\right) \mathrm{d}\vec{x'}\mathrm{d}t' \\ &= \vec{p}^\top \vec{J}\vec{p} \longrightarrow \min \quad (2) \\ \vec{J} &:= \int_{-\infty}^{\infty} w(\vec{x}-\vec{x'}, t-t')\, \vec{D}^\top \vec{D}\, \mathrm{d}\vec{x'}\mathrm{d}t' \end{aligned}$$

with the boundary condition $\vec{p}^\top\vec{p} = 1$ to avoid the trivial solution $\vec{p} = 0$. Here $w(\vec{x}-\vec{x'}, t-t')$ represents a weighting function that defines the spatio-temporal neighborhood for which the parameters are to be estimated. On a discrete grid the integral is changed to a summation and the weight function $w(\vec{x}-\vec{x'}, t-t')$ to the individual weights w_i. A binomial filter has been proven to be a good choice for the weights w_i as it is both symmetric and leads to a decreasing influence of data terms with distance from the considered pixel. The parameter vector $\vec{p}$ was taken out of the integral as it is assumed to be locally constant.

After incorporating the boundary condition in a Lagrangian multiplier calculus the minimization problem of Equation (2) is reduced to an eigenvector problem of the symmetric matrix $\vec{J}$:

$$\vec{J}\vec{p} = \lambda\,\vec{p}. \quad (3)$$

Consequently, the eigenvector $\vec{e}_3$ to the smallest eigenvalue λ_3 of $\vec{J}$ is the solution of the minimization problem. The velocities are given after normalization

$$\vec{p} = [u,v,1]^\top = \vec{e}_3/\vec{e}_{3,3}\,, \quad (4)$$

where $\vec{e}_{3,3}$ is the last element of the eigenvector $\vec{e}_3$. The eigensystem of the symmetric matrix $\vec{J}$ can be computed with Jacobi rotations as described by [Press, W. et al. 1992] or more elaborately by the algorithm proposed in [Drmac 1997].

2.2 Computation

The structure tensor $\vec{J}$ can be computed quite efficiently. First of all the spatial-temporal gradients $\mathscr{D}_x$, $\mathscr{D}_y$ and $\mathscr{D}_t$ have to be estimated. Here an isotropy optimized Sobel operator is used [Jähne, B. et al. 1999]. The elements of the structure tensor $\vec{J}$ can then be computed from

$$\vec{J} = \mathscr{B}(\mathscr{D}_x \cdot \mathscr{D}_y),$$

with the smoothing operator $\mathscr{B}$ and the differential operator $\mathscr{D}_q$ in the direction of the coordinate q. We use a 3 or 5 tab optimized Sobel operator and the integration is also performed on a 3 or 5 tab with a binomial filter for smoothing. The computational cost can be further reduced by exploiting the separability of the involved filters.

The estimation of the full optical flow field $\vec{f}$ is only possible if no aperture problem is present [Hildreth 1984]. This is equivalent to requiring that the rank r of $\vec{J}$ be $r = \mathrm{rank}\vec{J} = 2$. By analyzing the eigenvalues of $\vec{J}$ a coherence measure c_e can be computed, indicating regions where full motion can be derived. This coherence measure is given by

$$c_e = \frac{\lambda_2 - \lambda_3}{\lambda_2 + \lambda_3}, \quad (5)$$

where λ_3 and λ_2 are the smallest and second smallest eigenvalues, respectively.

In natural image sequences large areas with negligible spatio-temporal gradients may be present. Since the trace of a matrix is invariant under rotation, trace$\vec{J}$ presents a good measure for these areas. By only computing the eigensystem of $\vec{J}$ at locations where the trace of the matrix is above a certain threshold τ, unnecessary computational cost is avoided. We refine this approach further by treating the diagonal elements of spatial and temporal gradients separately, i.e. we require

$$\vec{J}_{11} + \vec{J}_{22} > \tau_s \quad (6)$$

$$\vec{J}_{33} > \tau_t\,. \quad (7)$$

This condition is not fully rotationally invariant anymore, but allows a much better detection of motion irrelevant regions.

In our application concerned with the real-time presentation of selected motion features we can further reduce the computational load without significant loss of accuracy. First, we reduce the spatial resolution of the images with a down-sampling step. This is legitimate since in a real-time display the user is not able to draw any information from a single pixel anyway, and we often even apply a smoothing step for the visualization of the motion (Section 3.4). After the computation the images are scaled up again for display. The down-sampling is not critical as long as the texture information, which is crucial for the diversification of the structure tensor elements, is not lost. Typically we scale down the VGA format (640x480) to 320x240.

We also reduce the temporal resolution of the image sequence, if the frequency is higher than 25Hz. By performing the eigenvalue analysis for every other image, execution speed increases significantly. The problem of temporal aliasing can be counteracted by calculating the regularized gradients for all images of the sequence. Thus we obtain motion estimates of exactly the same quality as before but simply with a lower frequency, typically 25Hz.

3 Visualization

In this section we follow the visualization process from the raw velocities to the display of motion features in the image sequence. Figure 1 accompanies the explanation of the individual steps of this process.

3.1 Coloring

From the motion estimation we obtain an image with the estimated x and y velocities (Eq. 4). In Figure 1**a** we see the modulus of the velocity as intensity. Visual representation of vector fields is an extensive topic of its own. However, in real-time image sequences there is little time for computation and the user has only a fraction of a second to perceive and understand the images. A reliable method for conveying a qualitative picture of the motion is to use color, color is especially useful to catch the eye of the observer in an otherwise gray image [Travis 1991].

We use a 2D color map to represent the motion field. Theoretically each location in the color map is assigned a different color, such that all directions can be unambiguously distinguished, but it is illusionary to think that this information can correctly be interpreted in real-time. It is more advisable to adapt the color map to the application in mind. For the test sequences of walking people we use a map which helps to distinguish the differences in x velocity, with the y axis being poorly represented (Figure 2**a**). Figure 2**b** shows a map which represents all directions equally well. However, this color richness can be often more confusing than helpful and so for the medical data sets we use either a rainbow encoding of the velocity modulus (Figure 2**c**) or even a single color and rely on the fading explained below to better convey the motion. From the implementational point of view any texture with a color map could be used. Figure 1**b** is an image of the test sequence after the first coloring step.

3.2 Blending

The motion estimator works adaptively only on these regions which yield a sufficiently pronounced structure tensor (Eqs. 6,7). This saves a lot of computation time in typical sequences as can be seen in Figure 1**c**, where the uncomputed area is displayed in black. Despite the air irritations visible in 1**b**, most of the background is omitted upon Equation 7, while the homogeneous black in the trousers violates Equation 6. Areas with a strong aperture problem are also masked out (Eq. 5). For visualization purposes this empty area can be used to blend in the original image sequence (Figure 1**d**).

3.3 Region of Interest

In general the motion field contains velocities of various scales and in a given application we are usually only interested in a small subset of them. Also at spatial and temporal (very fast motion) discontinuities we can still obtain erroneous results despite the culling based on the quality measure (Eq. 5). Therefore, we allow to specify the region of interest on the velocity modulus or an axis through the center of the color map to select velocities upon the intensity in a certain direction, e.g. the x direction in the test sequence. In Figure 1**e** we have tried to pronounce the faster moving person in this way. But we see that the arms and legs of the others are moving at an even higher velocity. We need additional post-processing to distinguish among the velocity regions.

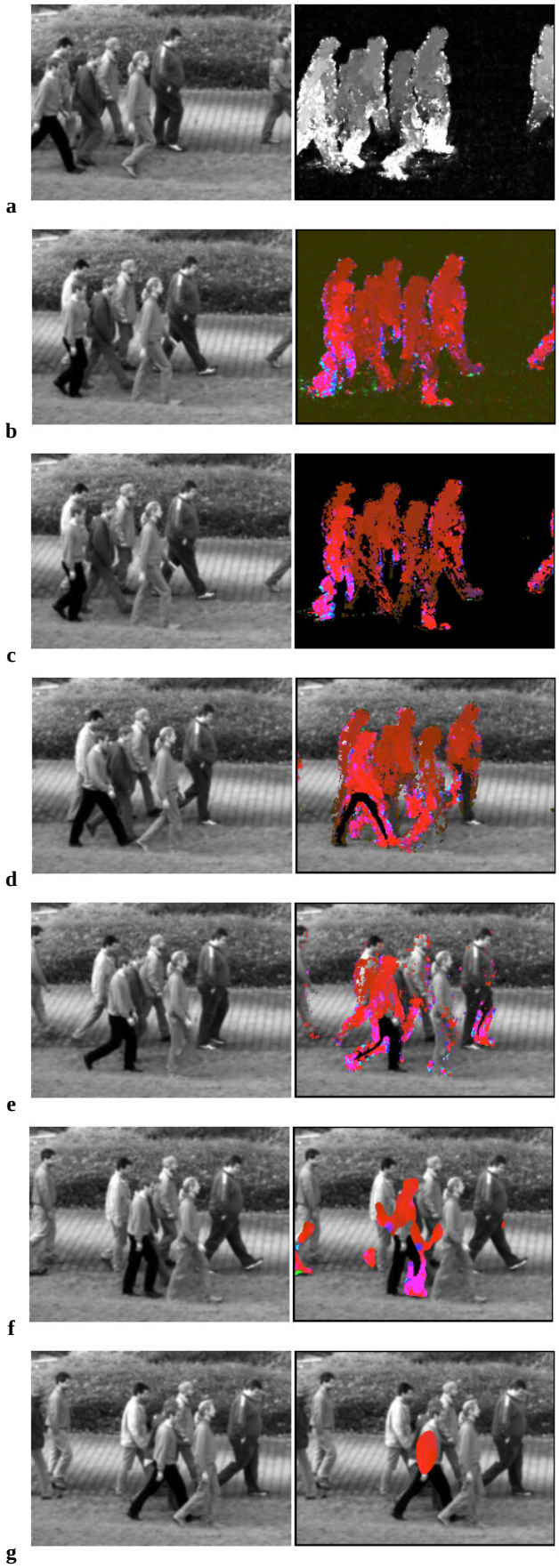

Figure 1: The steps of the visualization pipeline described in Section 3. Every fifth frame of the sequence is shown.

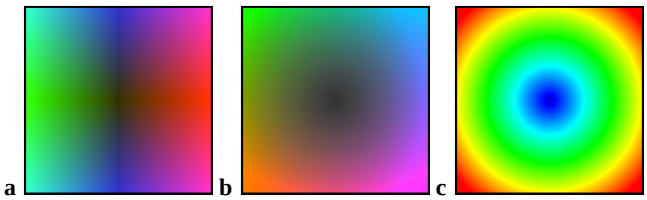

Figure 2: Color maps used for the coloring of the velocities

3.4 Smoothing

The previous selector determines the visible value range of the velocities. This produces regions of similar velocity but different size. We can use their size and form as a criteria to differentiate between them. For this purpose we apply a hierarchical smoothing scheme which on each level smooths the characteristic function of the selected regions. By thresholding the obtained values we can select the spatial scale of motion regions. The preference of this process for regions expanded in a certain direction can be influenced by changing the weights of the smoothing mask. The scheme applies the smoothing also to the velocities themselves to obtain a nicer visual representation. Figures 1**f** and **g** show the results after the application of 3 and 5 smoothing steps in the hierarchy respectively.

In the sequence from which Figure 1**g** has been extracted only the faster moving person is marked by the display of the motion region. All other motion regions, though similar or even higher in velocity, are masked out. This masking is very general and does not require any knowledge about the objects or type of motion. However, if this knowledge is present it could help to provide even finer feature distinctions. In future, we will therefore consider the integration of one of the many motion segmentation techniques which can incorporate such a-priori information.

3.5 Fading

In some cases we are not only interested in the display of the current motion field, but want also to visualize the regions already effected by previous motion. In angiography, for example, the flow of blood marked by a contrast agent is of great interest to the physician. But the motion estimator can only compute velocities at the front of the in- or outflowing agent. Without further processing the visualization of these velocities results in a confusingly fast rush of colors through the image sequence. Such sequences have also a lower temporal resolution, so that the motion estimates at any individual time point are not as reliable as their weighted integration.

During the streaming of the sequence we record for the each pixel location the point in time at which it represented a non zero velocity. This information is used to display a fading of the recorded motion. Figure 8 shows the benefit of this visualization method.

4 Hardware Implementation

Graphics cards are neither the most flexible nor powerful architectures for our application. But they perform better than typical micro-processors on such problems, because they use a different computing paradigm which is more suitable for the throughput of large data volumes. This means that they are the platform of choice for an inexpensive image sequence processing tool. To point out the architectural benefits of graphics cards we sketch their relation to the general idea of data-stream-based processing.

4.1 Data-Stream-Based Processing

In data-stream-based architectures the data streams rather than the instruction streams trigger the execution of operations. This computing paradigm deals much better with the memory gap [Wilkes 2000], the mismatch of memory and processor performance. In contrast to software for instruction-stream-based architectures, e.g. micro-processors, it requires two programming sources: *flowware*, which determines the assembly and direction of data streams, and *configware*, which contains the configuration of the processing elements. In FPGAs and some reconfigurable computing machines both sources can be executed on the same elements, but conceptually they are still different. Processor-in-Memory or stream architectures usually require two different sources explicitly.

The dual programming model has the great advantage that the individual elements of the data streams are assembled from memory before the actual processing. This allows the optimization of the memory access patterns, minimizing latencies and maximizing the sustained bandwidth. Unimodal software programs used for instruction-stream-based architectures allow only a limited prefetch of the input data, based on predictions of conditional jumps in the instruction stream. We recommend [Hartenstein 2003] as a starting point for further reading on this subject.

Graphics processors are a subclass of stream processors. Beside their unrivaled price-performance ratio, they have the great advantage that there exist widespread platform (DirectX API) and system (OpenGL API) independent Application Program Interfaces (APIs) for access to their functionality. The APIs are used both for the specification of flowware and configware, but the commands are clearly distinguished. In the following we describe first the control of the data-flow (flowware) and then the configuration of the processing elements in the graphics pipeline (configware) for our application.

4.2 Data-flow

First we assume that the individual images of the image sequence lie in main memory. The images are transported one by one to the graphics card with an asynchronous mechanism, which allows the card to continue the current computation during the transfer. For this we use several circular buffers on the card, and the image loads to a buffer position which is not needed in the concurrent computation. Each image is read only once, so that the AGP bus provides sufficient bandwidth in comparison to the number of on-card operations as not to decrease the overall performance. Because all steps of the algorithm are performed on the card, no additional memory transfers are needed. The final result is displayed directly from the graphics memory onto the screen.

On the graphics card the images are represented as pbuffers. These buffers can serve either as a source (texture) or a destination of data streams (see Figure 3). The operations of the algorithm are performed by streaming the texture operands through the appropriately configured graphics pipeline (Section 4.3) to a target pbuffer. The target pbuffer can then be used as a texture operand in the succeeding operation. Because several such passes are required by the algorithm, we use mainly floating point pbuffers to retain sufficient precision in intermediate computations.

As long as the same operation is applied to all image pixels, the entire images form the data streams, as the efficiency of the pipeline grows with the size of the streams. The handling of the adaptive exclusion of certain regions from computation, which requires the use of smaller streams, is described in Section 4.4.

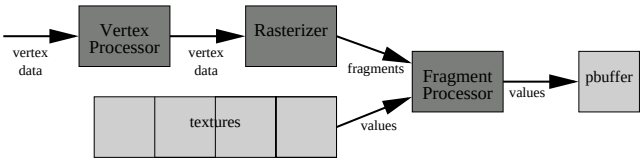

Figure 3: A simple diagram of the DX9 graphics pipeline. Light gray represents data containers, dark gray processing units. In each pass a different texture can serve as the target pbuffer for the output data stream.

Currently we assume that the image sequence is stored in the main memory. Since we need to read each image only once, the algorithm would work just the same if the images arrived from an external source at a certain memory address one by one. In fact, in a future version we plan to decode a video stream in real-time on the CPU, while the graphics processor works on the motion estimation and visualization.

4.3 Graphics Pipeline

The DX9 graphics pipeline contains two freely programmable parts, the vertex and the fragment processor (Figure 3). The vertex processor mainly manipulates the input vertex and texture coordinates and vertex color. The hard-wired rasterizer interpolates these values for each pixel in the primitive which is currently being drawn, e.g. a triangle. The interpolated values associated with one pixel location are called a fragment. They are manipulated by the fragment processor. The fragment processor combines the fragment data with possibly additional values from up to 16 textures to determine the output value for the current pixel.

Each vertex and each fragment is processed independently of the others in the same data stream. Therefore, they can be processed in parallel very quickly. The processing of streams achieves highest performance if the data streams are large and the texture access in the fragment processor uses only neighboring texture values, such that internally bandwidth-efficient memory burst-modes can be used and latency can be hidden with small caches.

The application's configware for the graphics pipeline consists mainly of short assembly programs for the vertex and fragment processor. Beyond, there are only minor parameter configurations of the fixed parts of the pipeline, e.g. the per-fragment tests which are executed after the fragment processor has completed. For the design of the configurations we use Cg [NVIDIA 2002], a C-like high level graphics programming language. A compiler generates from Cg the assembly code understood by the API.

In image based problems like ours the fragment processor bears most of the computational burden. We use the vertex processor only for the generation of texture coordinates to the neighboring values in a texture, whereas each step in the algorithm (Figure 4) requires a different configuration of the fragment processor.

The eigenvector analysis of the tensor is by far the longest and thus most demanding fragment program with almost 300 assembly operations for the 3 sweeps of the Jacobi method for matrix diagonalization. The main visualization program (coloring, blending, fading) is the next larger with approx. 50 operations, but most configurations have less than 10. Therefore, it makes sense to design an adaptive scheme which skips the eigenvector analysis for irrelevant data.

```
motion estimation {
  start loading of image for next iteration
  sample down current image
  compute optimized gradient with Sobel masks
  assemble the weighted structure tensor
  classify irrelevant regions for culling
  compute motion estimates through
    an eigenvector analysis of the tensor
}
visualization {
  select a range of velocities
  smooth the selected motion regions hierarchically
  record the selected motion areas
  visualize the result: coloring, blending, fading
  display the result
}
```

Figure 4: Overview of one iteration of the main algorithm. Apart from the loading, each line corresponds to the configuration of the fragment processor with the corresponding program and the streaming of the texture operands through the so configured graphics pipeline (see Figure 3). Some operations require several passes with slightly different configurations, e.g. smoothing in x and y direction.

4.4 Adaptivity

By analyzing the structure tensor (Eq. 2) we can save on the computation of the eigenvectors in areas which do not contribute significantly to the motion field (Eqs. 6,7). However, the introduction of efficient dynamic adaptive processing in graphics hardware is not straight forward. There exist per-fragment tests in the graphics pipeline which skip further processing depending on predefined masks and values of the fragments, but these are not very efficient, because they cannot exclude larger areas from processing at once. The fragment processor can also discard fragments, but in such a case the whole fragment program is still executed and only the final result is discarded. Significant speedup can currently be only obtained by culling areas on the vertex level.

The image is divided into tiles, each of which generates a data stream much smaller than the whole image. Smaller streams reduce the efficiency of the pipeline, but this effect is compensated to some extent by the graphics driver, which can efficiently catenate the individual data streams if their defining geometry is given in advance, ideally in a server sided vertex buffer object. A classification step determines which tiles need to be processed further and which can be skipped in the following. The classification step can be performed by combining the data of each tile to a single value and retrieving the values of all tiles with a single read-back to the main memory as in [Lefohn et al. 2003], where this technique has been introduced.

We use a different classification step which avoids the read-back by exploiting the occlusion test functionality. The test counts the number of passed fragments at a late stage in the graphics pipeline. The counters can asynchronously be retrieved from the graphics driver, i.e. they do not stall the ongoing computation. By discarding fragments upon the conditions in Equations 6,7, we thus easily obtain the number of motion relevant pixels in each tile, and can skip its subsequent processing if the number is below say 5%. The transition from Figure 1**b** to **c** demonstrates the savings. The tile structure becomes visible if one skips tiles with too many relevant pixels, e.g. 90% in Figure 5. For entire images the efficiency of the occlusion test has already been demonstrated in [Goodnight et al. 2003]. See also [Coombe et al. 2004] for a similar tile based testing.

Figure 5: The visible tiling in the adaptive scheme for much too aggressive culling. In Figure 1**c** the standard setting is used.

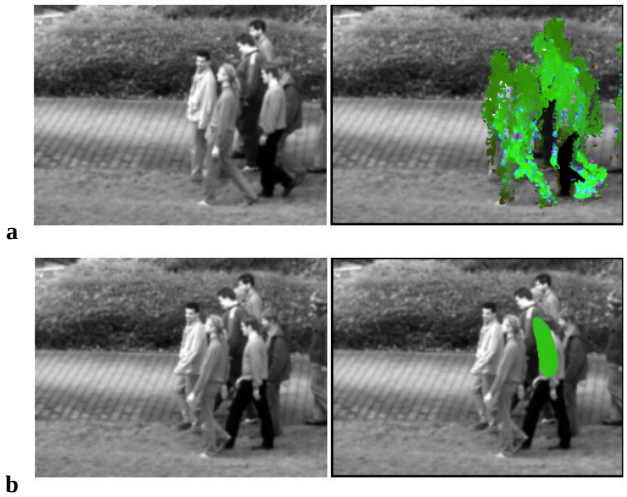

Figure 6: Visual emphasis of faster moving persons. In the upper row only the slightly brighter green conveys the qualitative velocity difference. Below the visual mark makes it much clearer.

5 Results

We use two types of image sequences as examples: test sequences of walking people to demonstrate the tool's ability to distinguish similar motion features, and angiography sequences for the enhancement of blood flow.

5.1 Motion Features

The sequences with walking people were recorded in VGA format at 100Hz. The computation takes place on 320x240 images. The eigenvector analysis runs on every fourth image resulting in a real-time requirement of 25Hz output frequency. Section 5.3 discusses the performance results.

Figure 1, discussed in Section 3, shows the individual steps which made it possible to visually extract the feature of the slightly faster moving person despite smaller regions (arms, legs) of higher velocity. Figure 6 shows another sequence of the same kind. In the above examples the parameters must be set carefully to obtain the visual distinction with such clarity. But it is obvious that a higher velocity difference requires only a rough selection of the visualization parameters. For example, for the task of marking persons who move in the wrong direction only the sign of admissible x velocities must be set correctly (Figure 7).

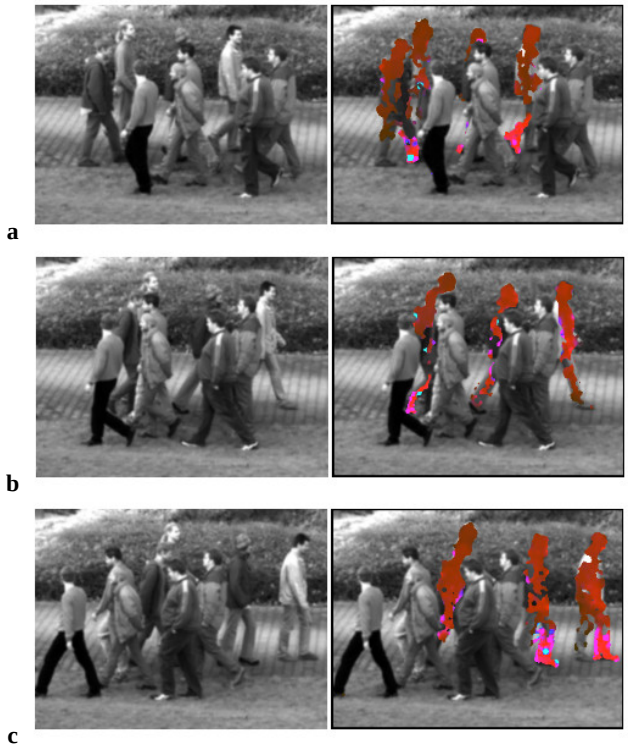

Figure 7: Marking of people who move in the wrong direction. Despite the occlusion the visual emphasis is very accurate.

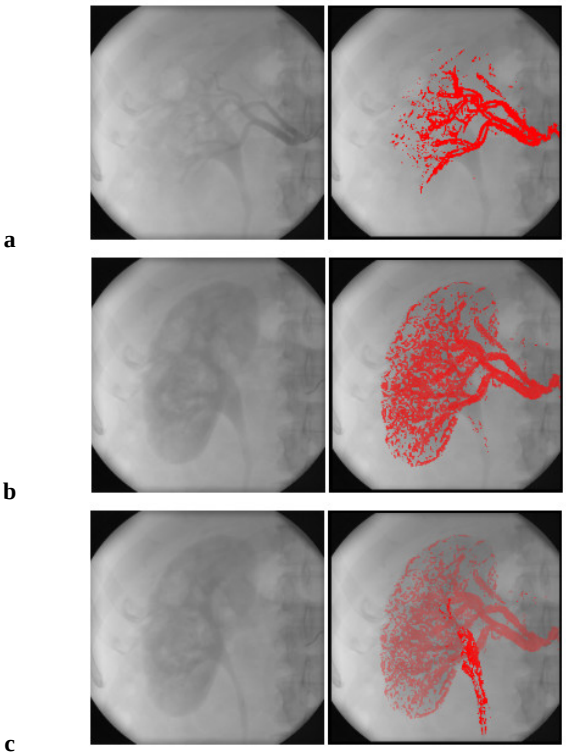

Figure 8: High velocities detected in the blood flow emphasized by a color fading. At the time point of frame **b** no velocities can be detected, such that without the fading the frame would not show any color at all.

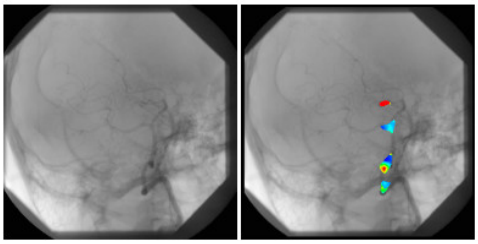

Figure 9: Detected regions of wide-stretched motion in the vascular system. Color indicates the modulus of velocity.

5.2 Flow Enhancement

The angiography sequences have a resolution of 1024x1024 at 20Hz. The computation takes place on 256x256 images, without a reduction of the temporal axis.

The first example shows the blood flow in a kidney (Figure 8). We see how the fading of the color helps to understand the distribution of the motion. In the second example we record the motion regions extending over a certain spatial scale to help in finding turbulent areas in the vascular system (Figure 9).

5.3 Performance

All results were computed with a GeForceFX 5800 Ultra 500MHz graphics processor. For comparison we have run a cache and SSE optimized version of the motion estimator on a Pentium 4 2GHz-FSB400 machine. Nowadays, the comparison is still in favor of the micro-processor, since the FX5800 is an old DX9 chip and the performance difference to the current graphics processors is larger than between the older and more recent Pentium 4 systems.

Sequence	Loops	P4	FX5800	Speedup	Est.+Vis.
Figure 1	403	22.4	5.0	4.5	6.0
Figure 6	353	19.8	4.3	4.6	5.2
Figure 7	303	17.7	3.8	4.7	4.6

The last column shows the graphics hardware timings for the whole process, i.e. motion estimation and visualization. We achieve a 4.5 speedup factor, which is a good value for this computationally intensive problem, in which the micro-processor benefits from its higher internal clock. Because of the long assembly program for the eigenvector analysis, our algorithm is bound by the fragment processor performance. In a software program the choice of sub-diagonal elements within a Jacobi sweep and the overall number of sweeps can be reduced dynamically depending on a user given tolerance. Similar our algorithm would also benefit from the proposed feature of dynamic branching in graphics hardware. However, we soon (May 2004) expect a quadrupled speedup simply from the fact that the upcoming generation of graphics cards presumably has four times the number of arithmetic units as the FX5800. In practice this would mean operating in real-time on full VGA image sequences.

6 Conclusions

We have presented a tool for the real-time motion estimation and visualization of image sequences. The precise, dense motion estimation allows to visually distinguish even very similar features through appropriate post-processing steps. The visualization pipeline contains several stages which can be easily controlled to serve the needs of different applications. Other hardware systems perform even more time consuming motion analysis in real-time but at a much higher price. For our tool a simple camera and a standard PC with a DX9 graphics card suffice, because we make efficient use of its data-stream-processing capabilities.

The current version implements the basic motion estimation based on the BCCE. This implies that gray values are modeled to remain constant on their trajectory. We want to incorporate further extensions which allow a gray value change as described by an appropriate partial differential equation [Haußecker, H. and Fleet, D. J. 2000]. In real world sequences another problem often encountered is multiple or transparent motion. The framework presented in this paper could also be extended to incorporate this type of motion [Mota, C. et al. 2001].

The unambiguous marking of objects with a certain motion feature suggests some sort of artificial intelligence in the algorithm. But currently the visual marks are based solely on the motion values. The inclusion of a-priori knowledge about the objects in the images could help to resolve even more difficult situations than those in the presented examples. From the implementational point of view we want to involve the CPU in the processing by decoding a camera's video stream and reusing its coarse motion estimators in real-time, while the graphics processor executes the precise motion estimation and visualization.

Acknowledgments

This work was partially funded by the DFG within the special research program on time sequence analysis and image processing. We want to thank the students of the research group digital image processing of the IWR, Heidelberg for help in creating the image sequences. Furthermore, we thank Stefan Böhm from Siemens Medical Solutions and Joachim Hornegger from the University Erlangen-Nürnberg for support with the medical data sets.

References

BARRON, J. L., FLEET, D. J., AND BEAUCHEMIN, S. 1994. Performance of optical flow techniques. *International Journal of Computer Vision 12*, 1, 43–77.

BEAUCHEMIN, S. S., AND BARRON, J. L. 1995. The computation of optical flow. *ACM Computing Surveys 27*, 3, 433–467.

BIGÜN, J., GRANLUND, G. H., AND WIKLUND, J. 1991. Multidimensional orientation estimation with application to texture analysis and optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 8, 775–790.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. In *Proceedings of SIGGRAPH 2003*.

BRODSKY, E., AND BLOCK, W. 2003. Interactive visualization of time-resolved contrast-enhanced magnetic resonance angiography (CE-MRA). In *IEEE Visualization 2003*.

CAMPANI, M., AND VERRI, A. 1990. Computing optical flow from an overconstrained system of linear algebraic equations. In *ICCV*, 22–26.

CAMUS, T. 1997. Real-time quantized optical flow. *The Journal of Real-Time Imaging 3*, 71–86. Special Issue on Real-Time Motion Analysis.

CHELLAPPA(ED.), R. 2000. Special section on video surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8, 745–746.

COLANTONI, P., BOUKALA, N., AND DA RUGNA, J. 2003. Fast and accurate color image processing using 3d graphics cards. In *Proceedings Vision, Modeling and Visualization 2003*.

COOMBE, G., HARRIS, M. J., AND LASTRA, A. 2004. Radiosity on graphics hardware. In *Proceedings Graphics Interface 2004*. to appear.

DANIEL, G., AND CHEN, M. 2003. Video visualization. In *IEEE Visualization 2003*, 409–416.

DRMAC, Z. 1997. Implementation of Jacobi rotations for accurate singular value computation in floating point arithmetic. *SIAM Journal of Scientific Computing 18*, 4, 1200–1222.

FENNEMA, C., AND THOMPSON, W. 1979. Velocity determination in scenes containing several moving objects. *Computer Graphics and Image Processing 9*, 301–315.

GOODNIGHT, N., WOOLLEY, C., LEWIN, G., LUEBKE, D., AND HUMPHREYS, G. 2003. A multigrid solver for boundary-value problems using programmable graphics hardware. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*.

GPGPU. GPGPU - general purpose computation using graphics hardware. http://www.gpgpu.org/. Mark J. Harris (Ed.).

HADWIGER, M., VIOLA, I., L, T. T., AND HAUSER, H. 2002. Fast and flexible high-quality texture filtering with tiled high-resolution filters. In *Proceedings of Vision, Modeling, and Visualization 2002*.

HARTENSTEIN, R. 2003. Data-stream-based computing: Models and architectural resources. In *International Conference on Microelectronics, Devices and Materials (MIDEM 2003)*.

HAUSSECKER, H., AND FLEET, D. J. 2000. Computing optical flow with physical models of brightness variation. In *CVPR'00*, vol. 2.

HAUSSECKER, H., AND SPIES, H. 1999. Motion. In *Handbook of Computer Vision and Applications*, Jähne, B., Haußecker, H., and Geißler, P., Eds., vol. 2. Academic Press, ch. 13.

HILDRETH, E. C. 1984. Computations underlying the measurement of visual motion. *Artificial Intelligence 23*, 309–354.

HOPF, M., AND ERTL, T. 1999. Accelerating 3d convolution using graphics hardware. In *Proc. Visualization '99*, IEEE, 471–474.

HORN, B. K. P., AND SCHUNK, B. 1981. Determining optical flow. *Artificial Intelligence 17*, 185–204.

IEEE COMPUTER SOCIETY. 2003. IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS 2003). ISBN 0-7695-1971-7.

JÄHNE, B., SCHARR, H., AND KÖRKEL, S. 1999. Principles of filter design. In *Handbook of Computer Vision and Applications*, Jähne, B., Haußecker, H., and Geißler, P., Eds., vol. 2. Academic Press, 125–151.

KIRBAS, C., AND QUEK, F. K. 2002. A review of vessel extraction techniques and algorithms. Tech. rep., Vision Interfaces and Systems Laboratory (VISLab), Wright State University, Dayton, Ohio, Nov. http://vislab.cs.wright.edu/review/extraction.html.

KITTLER, J., AND NIXON, M. S., Eds. 2003. Audio-and Video-Based Biometrie Person Authentication, 4th International Conference (AVBPA 2003). ISBN 3-540-40302-7.

KNUTSSON, H. 1989. Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis, Oulu, Finland, June 19-22, 1989*.

KOHLBERGER, T., SCHNÖRR, C., BRUHN, A., AND WEICKERT, J. 2003. Domain decomposition for parallel variational optical flow computation. In *Proceedings of DAGM-Symposium 2003*, 196–203.

KRUEGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG) 22*, 3, 908–916.

LEFOHN, A., KNISS, J., HANDEN, C., AND WHITAKER, R. 2003. Interactive visualization and deformation of level set surfaces using graphics hardware. In *Proc. Visualization*, IEEE CS Press, 73–82.

LIU, S.-Z., FU, C.-W., AND CHANG S. 1998. Statistical change detection with moments under time-varying illumination. *IEEE Transactions on Image Processing 7*, 9, 1258–1268.

LUCAS, B., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *DARPA Image Understanding Workshop*, 121–130.

MAYA-RUEDA, S., AND ARIAS-ESTRADA, M. 2003. FPGA processor for real-time optical flow computation. In *Proceedings FPL 2003*, 1103–1106.

MOTA, C., STUKE, I., AND BARTH, E. 2001. Analytic solutions for multiple motions. In *Proc. of International Conference on Image Processing*, vol. 2, 917–920.

NVIDIA, 2002. Cg programming language. http://developer.nvidia.com/view.asp?PAGE=cg_main.

PRESS, W., TEUKOLSKY, S., AND OTHERS. 1992. *Numerical Recipes in C*, 2 ed. Cambridge University Press, Cambridge, MA.

STRZODKA, R., AND TELEA, A. 2004. Generalized distance transforms and skeletons in graphics hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '04*.

TRAVIS, D. 1991. *Effective Color Displays*. Academic Press.

VAN WIJK, J. J. 2002. Image based flow visualization. *ACM Transactions on Graphics, special issue, Proceedings ACM SIGGRAPH 2002*.

WATANABE, M., KIKINIS, R., AND WESTIN, C.-F. 2002. Level set based integration of segmentation and computational fluid dynamics for flow correction in phase contrast angiography. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2002*.

WEISKOPF, D., ERLEBACHER, G., AND ERTL, T. 2003. A Texture-Based Framework for Spacetime-Coherent Visualization of Time-Dependent Vector Fields. In *Procceedings of IEEE Visualization '03*, 107–114.

WILKES, M. 2000. The memory gap (keynote). In *Solving the Memory Wall Problem Workshop*. http://www.ece.neu.edu/conf/wall2k/wilkes1.pdf.

WU, J. 1989. Mean square slope of the wind-disturbed water surface, their magnitude, directionality, and composition. *Radio Sience 25*, 2, 37–48.

ZULOAGA, A., MARTIN, J. L., AND EZQUERRA, J. 1998. Hardware architecture for optical flow estimation in real time. In *Proceedings ICIP 1998*, vol. 3, 972–976.