ESTIMATION OF OBJECT MOTION BY NORMALIZATION IN GRAPHICS CARDS

BY

ARAVINDTH SUDTHAN B(rvndsudthan@gmail.com,9842365667)
SUSHMITHA R (rsushmitharamesh@gmail.com,9789597272)

BANNARI AMMAN INSTITUTE OF TECHNOLOGY-SATHYAMANGALAM

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 spatio-temporal 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 newest generation of graphics hardware our tool could run in real time for the standard VGA format.

1 INTRODUCTION

For the estimation of motion from digital image sequences a number of different techniques has been proposed [1, 2, 19]. For real time applications, feature tracking algorithms are widely in use [7, 43]. While these approaches offer real time performance, estimated velocity fields are sparse. Also, inherent to these techniques is a reduced accuracy [31], 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 [1] 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 [28, 33, 44]. 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 micro-processors, 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 [15] 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 [9, 16, 21], linear algebra operations [4, 14, 29], visualization [9, 39, 41], adaptive hardware techniques [10, 30, 37].

Along with the increasing number of CCTV cameras literature on video surveillance has grown rapidly [8, 23, 26]. 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 [11], 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 [25]. We operate in real-time directly on the images similar to [5], whereas in a postprocessing step a much more detailed analysis can be obtained [40].

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) [22]. 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 [13]:

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

with the optical flow $\vec{f} := (dx/dt, dy/dt)^\top = (u,v)^\top$, the spacial gradient $\vec{\nabla}g = [g_x, g_y]^\top$ and the partial time derivative $g_t = \partial g/\partial t$. The data vector $\vec{d}$ is given by $\vec{d} := [g_x, g_y, g_t]$ 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 $\vec{f}$. Different approaches exist for solving this problem, such as introducing a global smoothness assumption. These techniques run into problems at motion discontinuities where measures have to be undertaken not to smooth over these boundaries. Therefore, in this work we chose a local approach that does not rely on any global constraints.

Assuming constant optical flow over a small spatio-temporal neighborhood surrounding the location of interest containing m pixels (for optical flow [32] and [6]), the problem consists of m equations of the form of Equation (1). With the data matrix

$$\vec{D} \quad := \quad (\vec{d}_1, \ldots, \vec{d}_m)^\top \tag{2}$$

the total least squares problem can be reformulated as the structure tensor [3, 19, 27], that is

$$
\begin{aligned}
||\vec{D}\vec{p}||_2 &= \int_{-\infty}^{\infty} w(\vec{x} - \vec{x}', t - t') \left(\vec{p}^\top \vec{D}^\top \vec{D} \vec{p} \right) \, d\vec{x}' dt' \\
&= \vec{p}^\top \vec{J} \vec{p} \longrightarrow \min \tag{3} \\
\vec{J} &:= \int_{-\infty}^{\infty} w(\vec{x} - \vec{x}', t - t') \vec{D}^\top \vec{D} \, d\vec{x}' dt'
\end{aligned}
$$

with the boundary condition $\vec{p}^\top \vec{p} = 1$ to avoid the trivial solution $\vec{p} = 0$. The parameter vector $\vec{p}$ was taken out of the integral as it is assumed to be locally constant. 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.

After incorporating the boundary condition in a Lagrangian multiplier calculus the minimization problem of Equation (3) is reduced to an eigenvector problem of the symmetric matrix $\vec{J}$:

$$\vec{J}\vec{p} = \lambda \vec{p}. \tag{4}$$

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

$$\vec{p} = [u, v, 1]^\top = \vec{e}_3 / \vec{e}_{3,3}, \tag{5}$$

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 [36] or more elaborately by the algorithm proposed in [12].

2.2 Computation

The structure tensor $\vec{J}$ can be assembled quite efficiently. First, the spatial-temporal gradients $\vec{d} = [g_x, g_y, g_t]$ have to be estimated. We apply a 3 or 5 tab isotropy optimized Sobel filter $\mathscr{D}$ in each direction $q \in \{x, y, t\}$: $\vec{d}_q = \mathscr{D}_q g$ [24]. Then, according to the continuous definition (Eq. 3), the elements of the structure tensor $\vec{J}$ can be computed from the data matrix $\vec{D}$ (Eq. 2):

$$\vec{J}_{pq} \quad = \quad \sum_{i=1}^{m} w_i \vec{D}_{ip} \vec{D}_{iq}. \tag{6}$$

Because the 3x3 tensor $\vec{J}$ is symmetric, we only need to compute 6 products $(\vec{d}_p \vec{d}_q)_{pq}, p \leq q$ at each pixel location. The weights w_i are usually chosen to be the binomial coefficients. Thus, we obtain $\vec{J}$ by applying a binomial filter $\mathscr{B}$ to each of the 6 products. The filter operates on a 3^2 or 5^2 stencil in the spatial domain, i.e. $m = 9$ or $m = 25$. The computational cost can be further reduced by exploiting the separability of the filters $\mathscr{D}$ and $\mathscr{B}$.

For the diagonalization of $\vec{J}$ we use the standard Jacobi method since the improved accuracy of the modified method from [12] is not directly transferable to graphics hardware, which implements division with reciprocals and uses only an approximate square root function. We perform the following iterations:

$$
\begin{aligned}
\vec{J}^0 &:= \vec{J}, & \vec{V}^0 &:= \mathbb{1}, \\
\vec{J}^{k+1} &:= \vec{G}^k{}^\top \vec{J}^k \vec{G}^k, & \vec{V}^{k+1} &:= \vec{V}^k \vec{G}^k, \\
\begin{pmatrix} \vec{G}^k_{p_k p_k} & \vec{G}^k_{p_k q_k} \\ \vec{G}^k_{q_k p_k} & \vec{G}^k_{q_k q_k} \end{pmatrix} &:= \begin{pmatrix} c_k \, s_k \\ -s_k \, c_k \end{pmatrix}, & \text{for other } pq: \ \vec{G}^k{}_{pq} &:= \mathbb{1}_{pq}, \tag{7} \\
c_k &:= (1 + t_k)^{-\frac{1}{2}}, & s_k &:= c_k \cdot t_k, \\
t_k &:= \frac{\text{sgn}(\tau_k)}{|\tau_k|\sqrt{1 + \tau_k^2}}, & \tau_k &:= \frac{\vec{J}^k_{q_k q_k} - \vec{J}^k_{p_k p_k}}{2\vec{J}^k_{p_k q_k}}.
\end{aligned}
$$

We use a cyclic pivot strategy, i.e. the matrix index (p_k, q_k) runs cyclically over the off-diagonal matrix indices $\{(p,q)|p < q\}$. With growing k the diagonal of $\vec{J}^k$ converges to the eigenvalues and $\vec{V}^k$ to the eigenvectors of $\vec{J}$.

The estimation of the full optical flow field $\vec{f}$ is only possible if no aperture problem is present [20]. This is equivalent to requiring that 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

$$c_e = \frac{\lambda_2 - \lambda_3}{\lambda_2 + \lambda_3}, \tag{8}$$

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 is above a certain threshold, unnecessary computational cost is avoided. We refine this approach by treating the diagonal elements of spatial and temporal gradients separately, i.e. we require

$$
\begin{aligned}
\vec{J}_{xx} + \vec{J}_{yy} &> \tau_s \tag{9} \\
\vec{J}_{tt} &> \tau_t. \tag{10}
\end{aligned}
$$

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 rather than ultimate precision in the estimation 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.

For frame rates higher than 25Hz the temporal resolution of the image sequences is reduced. To avoid temporal aliasing the regularized gradients still use the full temporal resolution and only the time intensive eigenvalue analysis skips intermediate images. Thus, artefacts that would be introduced by a mere sub-sampling are eliminated. At the same time a significant speed up is achieved.

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. 5). 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 [38].

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. 9,10). 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 (10), while the homogeneous black in the trousers violates Equation (9). Areas with a strong aperture problem are also masked out (Eq. 8). 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. 8). 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
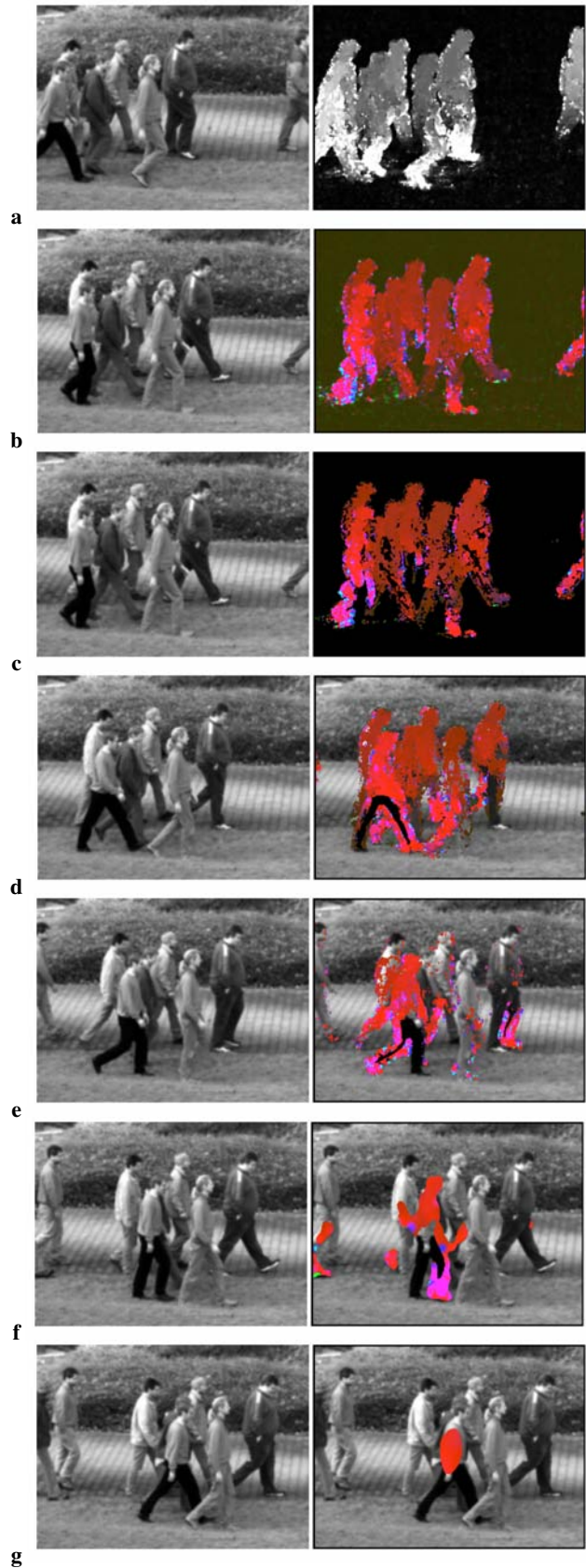

Figure 1: The steps of the visualization pipeline described in Section 3. Every fifth frame of the sequence is shown.
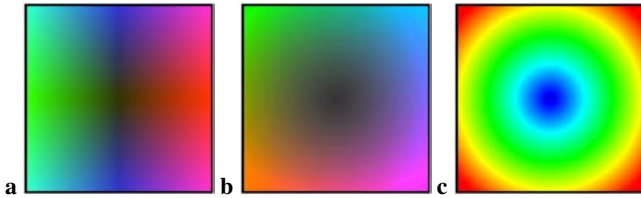
Figure 2: Color maps used for the coloring of the velocities

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.

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 smooth the characteristic function of the selected regions. By thresholding the obtained values we can select the spatial scale of the 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. For the elimination of small disjoint regions only the values of the smoothed characteristic function are relevant. But for a nicer visual representation the scheme applies the smoothing also to the velocities themselves. Figures 1**f** and **g** show the results after the application of 3 and 5 smoothing steps 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 processors achieve a high throughput for large data volumes by applying a data-stream-based computing paradigm [17]. In particular, this paradigm deals well with the memory gap [42], the mismatch of memory and processor performance. Among FPGAs, reconfigurable computing arrays, Processor-in-Memory or stream architectures, graphics processors are neither the most flexible nor

powerful devices exploiting data-stream-based processing, but they offer an unrivaled price-performance ratio and a comparably easy high level language access to their functionality. This means that they are the platform of choice for an inexpensive image sequence processing tool.

In the following we describe first the control of the data-flow and then the configuration of the processing elements in the graphics pipeline for our application.

4.1 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 (pixel buffer objects), 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 are 2D data arrays which 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 4.2) 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 pixels of the image sequence, the implementation is very fast, as the efficiency of the pipeline grows with the size of the streams. The handling of adaptive exclusion of certain regions from computation requires the use of smaller streams. This process is described in Section 4.3.

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.

4.2 Pipeline Configuration

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 additional values from up to 16 textures to determine the output value for the current pixel.

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. For the design of the configurations we use Cg [35], a C-like high level graphics programming language.

The motion estimation consists of two major tasks: the assembly of the structure tensor and its diagonalization. The assembly of $\vec{J}$
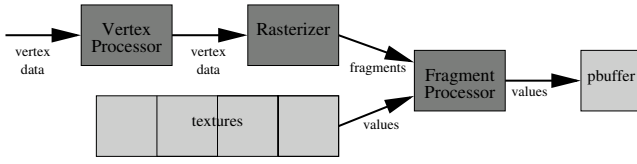
Figure 3: A simple diagram of the DX9 graphics pipeline. Light gray represents data containers, dark gray processing units. In each pass a different texture can serve as the target pbuffer for the output data stream.

```
motion estimation {
    start loading of the image for the next iteration
    sample down current image
    assemble the weighted structure tensor J⃗ (Eq. 6) {
        estimate the gradient with optimized Sobel filter 𝒟
        compute the products of gradient components (d⃗_p d⃗_q)_pq
        apply the binomial filter ℬ
    }
    classify irrelevant regions for culling (Eqs. 9,10)
    perform an eigenvector analysis of the tensor J⃗(Eq. 4) {
        diagonalize the tensor with Jacobi rotations (Eq. 7)
        compute the coherence measure c_e (Eq. 8)
        estimate the motion p⃗ (Eq. 5)
    }
}
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 at the desired scale
}
```

Figure 4: Overview of one iteration of the main algorithm. Basically each line corresponds to the configuration of the fragment processor with the corresponding fragment 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, while others can be executed in a single configuration (eigenvector analysis of the structure tensor).

is implemented according to Equation (6) as several passes with configurations for the binomial $\mathscr{B}$ and the optimized Sobel $\mathscr{D}$ filter, and the products $(\vec{d}_p \vec{d}_q)_{pq}, p \leq q$. Because the filters are separable, separate passes for the x, y and t direction save a lot of computation, e.g. the 5 tab optimized Sobel filter requires only 15 multiplications and additions although $5^3 = 125$ different values are involved. The savings are smaller for the binomial filter which operates on 2D stencils only in the spatial domain.

The eigenvector analysis executes in a single pass with a large configuration which performs a constant number (typically 9) of cyclic Jacobi rotations (Eq. 7) on $\vec{J}$. The approximate eigenvectors and eigenvalues are used to estimate the motion (Eq. 5) and the coherence measure (Eq. 8). We store the symmetric 3x3 matrices $\vec{J}^k$ as two and the transformation matrices $\vec{V}^k$ as three 3-vectors, such that the rotations can be vectorized into the internal 4-vector operations of graphics hardware.

The visualization pipeline contains also an iterative part (hierarchical smoothing) and a long configuration for the main visualization program (coloring, blending, fading). The hierarchical smoother is a series of filter applications. The separable filters have

the form $(\sigma_y, 1 - 2\sigma_y, \sigma_y)^\top (\sigma_x, 1 - 2\sigma_x, \sigma_x)$. By choosing $\sigma_x \neq \sigma_y$ we can favor the smoothing in one direction. To quickly incorporate information from farther regions in the smoothing process we use a multi-grid approach, but without actually generating smaller grids. Working on the next higher level means multiplying the offsets to the stencil positions with 2 such that we retrieve the values which would have been restricted to the next higher level in a standard multi-grid, e.g. the x mask $(\sigma_x^0, 1 - 2\sigma_x^0, \sigma_x^0)$ on level 0 becomes $(\sigma_x^1, 0, 1 - 2\sigma_x^1, 0, \sigma_x^1)$ on level 1. Certainly, we lose the computational savings of smaller grids, but because we operate on all pixels in the same manner, we have the advantage that the results are more stable under translation. This is especially important since the real-time requirements allow only very few iterations. In the standard setting we smooth only once on each level. So after 3 or 5 smoothing steps already a 15x15 or 63x63 neighborhood contributes to the smoothing of each pixel, respectively.

The eigenvector analysis of the tensor is by far the longest and thus most demanding fragment program with almost 300 assembly operations for 9 Jacobi rotations and the motion estimation. The main visualization program (coloring, blending, fading) is the next larger with approx. 50 operations. Most configurations have less than 10. Therefore, it makes sense to design an adaptive scheme which skips the eigenvector analysis for irrelevant data.

4.3 Adaptivity

By analyzing the structure tensor (Eq. 6) we can save on the computation of the eigenvectors in areas which do not contribute significantly to the motion field (Eqs. 9,10). 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 [30], 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 (9),(10), 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 [14]. See also [10] for a similar tile based testing.
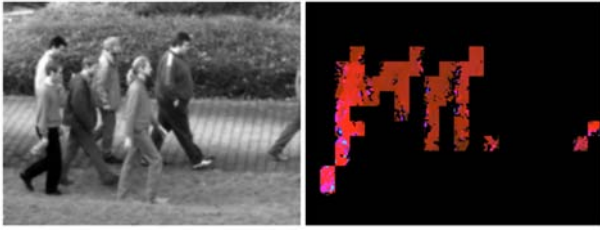
Figure 5: The visible tiling in the adaptive scheme for much too aggressive culling. In Figure 1c the standard setting is used.

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 (640x480) 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).
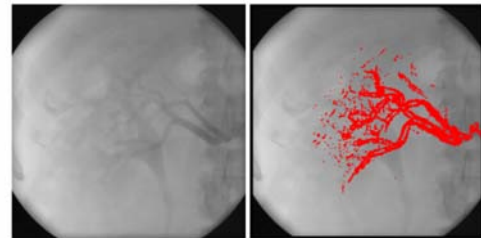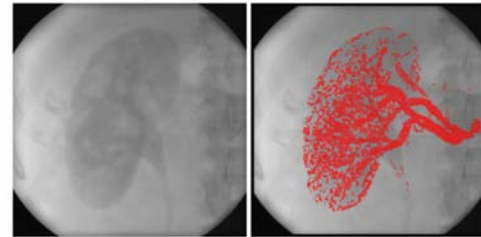

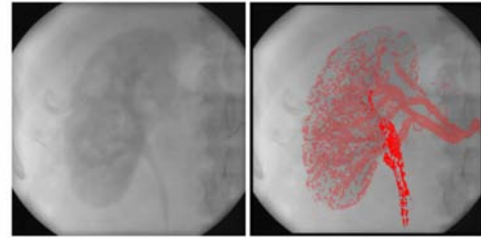Figure 7: Marking of people who move in the wrong direction. Despite the occlusion the visual emphasis is very accurate.

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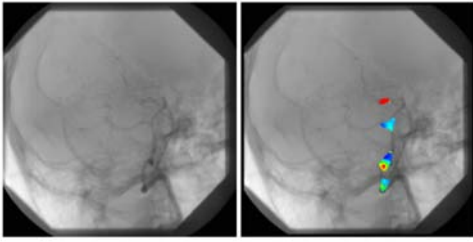
Figure 9: Detected regions of wide-stretched motion in the vascular system. Color indicates the modulus of velocity.

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

Figure 10 shows performance results computed with a GeForceFX 5800 Ultra (GF5800U) and different Pentium 4 (P4) processors. The software implementation was tuned to exploit cache coherency and the SSE operations. We achieve a 4.5 and 2.8 speedup factor against the older and newer P4 system respectively. We see that the newer P4 executes 1.6 times faster than the older one, which can be attributed both to the higher clock frequency and the faster FSB. Similarly the graphics processors would benefit from both a wider data bus and more or faster execution units. For our algorithm the latter is more important, because the performance of the very long assembly program for the eigenvector analysis is bound by the capability of the fragment processor. Therefore, we expect a quadrupled speedup on the newest generation of graphics hardware (Radeon X800XT, GeForce 6800 Ultra), which executes four to eight times more arithmetic operations in the fragment processor than the GF5800U. In practice this would mean operating in real-time on VGA image sequences.

The algorithm could also benefit from the new feature of dynamic branching in graphics hardware. For the performance comparison a constant number of cyclic Jacobi rotations has been executed in all cases. But in a software program the choice of the pivot elements and the overall number of rotations usually takes place dynamically depending on a user given tolerance. This allows to exploit the iterative nature of the Jacobi method (in contrast to QR decomposition) and iterate longer when the diagonalization of $\vec{J}$ is difficult or terminate faster for easy cases.

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
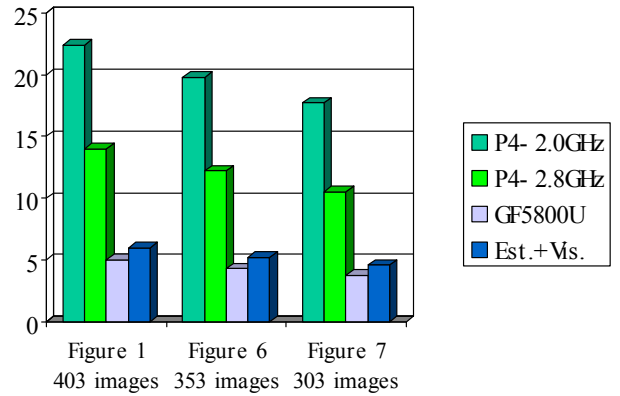
Execution times

Sequence	P4-2.0GHz	P4-2.8GHz	GF5800U	Est.+Vis.
Figure 1	22.4	14.0	5.0	6.0
Figure 6	19.8	12.2	4.3	5.2
Figure 7	17.7	10.5	3.8	4.6

Speedups

Sequence	GF5800/P4-2.0	GF5800/P4-2.8	P4-2.8/P4-2.0
Figure 1	4.5	2.8	1.6
Figure 6	4.6	2.8	1.6
Figure 7	4.7	2.8	1.7

Figure 10: Comparison of the motion estimation performance of a Pentium 4-2.0GHz FSB400, Pentium 4-2.8GHz FSB800 and the GeForceFX 5800 Ultra 500MHz / 500MHz (128bit DDR) graphics processor. The fourth bar shows the graphics timings for the entire process, i.e. motion estimation and visualization. All times are given in seconds.

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 [18]. 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 [34].

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.

REFERENCES

[1] Barron, J. L., Fleet, D. J., and Beauchemin, S. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[2] Beauchemin, S. S. and Barron, J. L. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–467, 1995.

[3] Bign, J., Granlund, G. H., and Wiklund, J. Multidimensional orientation estimation with application to texture analysis and optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):775–790, 1991.

[4] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. In *Proceedings of SIGGRAPH 2003*, 2003.

[5] Ethan Brodsky and Walter Block. Interactive visualization of time-resolved contrast-enhanced magnetic resonance angiography (CE-MRA). In *IEEE Visualization 2003*, 2003.

[6] Campani, M. and Verri, A. Computing optical flow from an overconstrained system of linear algebraic equations. In *ICCV*, pages 22–26, Osaka, 1990.

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

[8] Rama Chellappa(Ed.). Special section on video surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):745–746, 2000.

[9] Philippe Colantoni, Nabil Boukala, and Jérôme da Rugna. Fast and accurate color image processing using 3d graphics cards. In *Proceedings Vision, Modeling and Visualization 2003*, 2003.

[10] Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on graphics hardware. In *Proceedings Graphics Interface 2004*, 2004. to appear.

[11] Gareth Daniel and Min Chen. Video visualization. In *IEEE Visualization 2003*, pages 409–416, 2003.

[12] Z. Drmac. Implementation of Jacobi rotations for accurate singular value computation in floating point arithmetic. *SIAM Journal of Scientific Computing*, 18(4):1200–1222, 1997.

[13] Fennema, C. and Thompson, W. Velocity determination in scenes containing several moving objects. *Computer Graphics and Image Processing*, 9:301–315, 1979.

[14] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. A multigrid solver for boundary-value problems using programmable graphics hardware. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2003.

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

[16] Markus Hadwiger, Ivan Viola, Thomas Theußl, and Helwig Hauser. Fast and flexible high-quality texture filtering with tiled high-resolution filters. In *Proceedings of Vision, Modeling, and Visualization 2002*, 2002.

[17] Reiner Hartenstein. Data-stream-based computing: Models and architectural resources. In *International Conference on Microelectronics, Devices and Materials (MIDEM 2003)*, Ptuj, Slovenia, Oct. 2003.

[18] Hauecker, H. and Fleet, D. J. Computing optical flow with physical models of brightness variation. In *CVPR'00*, volume 2, 2000.

[19] Hauecker, H. and Spies, H. Motion. In Jhne, B., Hauecker, H., and Geiler, P., editors, *Handbook of Computer Vision and Applications*, volume 2, chapter 13. Academic Press, 1999.

[20] E. C. Hildreth. Computations underlying the measurement of visual motion. *Artificial Intelligence*, 23:309–354, 1984.

[21] M. Hopf and T. Ertl. Accelerating 3d convolution using graphics hardware. In *Proc. Visualization '99*, pages 471–474. IEEE, 1999.

[22] Horn, B. K. P. and Schunk, B. Determining optical flow. *Artificial Intelligence*, 17:185–204, 1981.

[23] IEEE Computer Society. *IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS 2003)*, 2003. ISBN 0-7695-1971-7.

[24] Jhne, B., Scharr, H., and Krkel, S. Principles of filter design. In Jhne, B., Hauecker, H., and Geiler, P., editors, *Handbook of Computer Vision and Applications*, volume 2, pages 125–151. Academic Press, 1999.

[25] Cemil Kirbas and Francis K.H. Quek. A review of vessel extraction techniques and algorithms. Technical report, Vision Interfaces and Systems Laboratory (VISLab), Wright State University, Dayton, Ohio, Nov. 2002. http://vislab.cs.wright.edu/review/extraction.html.

[26] Josef Kittler and Mark S. Nixon, editors. *Audio-and Video-Based Biometrie Person Authentication, 4th International Conference (AVBPA 2003)*, 2003. ISBN 3-540-40302-7.

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

[28] Timo Kohlberger, Christoph Schnörr, Andrés Bruhn, and Joachim Weickert. Domain decomposition for parallel variational optical flow computation. In *Proceedings of DAGM-Symposium 2003*, pages 196–203, 2003.

[29] Jens Krueger and Ruediger Westermann. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG)*, 22(3):908–916, 2003.

[30] A. Lefohn, J. Kniss, C. Handen, and R. Whitaker. Interactive visualization and deformation of level set surfaces using graphics hardware. In *Proc. Visualization*, pages 73–82. IEEE CS Press, 2003.

[31] Liu, S.-Z., Fu, C.-W., and Chang S. Statistical change detection with moments under time-varying illumination. *IEEE Transactions on Image Processing*, 7(9):1258–1268, 1998.

[32] Lucas, B. and Kanade, T. An iterative image registration technique with an application to stereo vision. In *DARPA Image Understanding Workshop*, pages 121–130, 1981.

[33] Selene Maya-Rueda and Miguel Arias-Estrada. FPGA processor for real-time optical flow computation. In *Proceedings FPL 2003*, pages 1103–1106, 2003.

[34] Mota, C., Stuke, I., and Barth, E. Analytic solutions for multiple motions. In *Proc. of International Conference on Image Processing*, volume 2, pages 917–920, 2001.

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