Journal of Circuits, Systems, and Computers
Vol. 26, No. 2 (2017) 1750022 (20 pages)

DOI: 10.1142/S0218126617500220

GPU-Based Fluid Motion Estimation Using Energy Constraint*

Siyuan Xu†, Han Zhuang†, Xin Fu‡, Junlong Zhou†
and Mingsong Chen†,§

†*Shanghai Key Lab of Trustworthy Computing,*
East China Normal University, Shanghai 200062, China

‡*Department of Electrical and Computer Engineering,*
University of Houston, Houston, TX 77004, USA
§*mschen@sei.ecnu.edu.cn*

Received 1 March 2016
Accepted 21 July 2016
Published 15 September 2016

Although motion estimation (ME) approaches for fluid flows have been widely studied in computer vision domain, most existing ME algorithms cannot accurately deal with regions with both slight and drastic brightness changes. To address this issue, this paper introduces a novel data structure called brightness distribution matrix (BDM) which can be used to accurately model regional brightness. Based on our proposed consistency constraints and energy function, we can obtain motion vectors from image sequences with high accuracy. Since the BDM-based ME approach requires a large number of computations when dealing with complex fluid scenarios, to reduce the overall ME time, a parallelized version of our approach is developed based on graphics processing unit (GPU). Experimental results show that our GPU-based approach not only can be used to improve the ME quality for complex fluid images, but also can reduce the overall ME processing time (up to 7.06 times improvement).

Keywords: Fluid; motion estimation (ME); graphics processing unit (GPU); compute unified device architecture (CUDA).

1. Introduction

Motion estimation (ME) of fluid flows has been widely studied in many areas. For example, in the field of pattern recognition, the motion fields derived by ME techniques can be used to support facial expression recognition.[1,2] As a promising means, ME methods play an important role in the domain of environmental science. They not only can be employed to track ice floes,[3] but also can be utilized for the purpose of weather prediction and ocean circulation analysis.[4,5] Moreover, ME techniques have

*This paper was recommended by Regional Editor Tongquan Wei.
§Corresponding author.

been considered as the key components in the domain of fluid mechanics,[6] medical images processing,[7] and image recovery.[8]

When applying ME, the motion fields are captured by cameras, and the images of the perspective projection onto the image plane are saved for the analysis. However, in this way, the images generally are not properly represented, since they only provide the brightness information of fluid flows. It is common that brightness changes irregularly in image series, which can easily result in inaccurate ME results. Therefore, how to compute motion fields and how to recognize the image patterns have become major challenges in ME. Although in the past decade there are dozens of approaches proposed to address the above issues, most of them focus on the accuracy of motion fields rather than time-efficiency. Consequently, to achieve expected ME accuracy, existing approaches require a huge number of computation efforts to process complex images, which strongly limits the ME application in practice.

As a promising approach, graphics processing unit (GPU) is good at dealing with problems which can be expressed as data-level parallel computations. Therefore, it is becoming the mainstream computation platform to manipulate computer graphics and image processing.[20] Figure 1 illustrates the difference between CPU and GPU architectures. Unlike traditional multicore CPUs which are based on the multiple instruction, multiple data (MIMD) design, the single instruction, multiple data (SIMD)-based GPU has a highly parallel structure which enables more efficient computation than general-purpose CPUs when handling large blocks of data in parallel. To facilitate the GPU programming, compute unified device architecture (CUDA) provides a parallel computing platform and application programming interface (API) model, which can utilize GPU resources efficiently. CUDA offers a unified hardware and software solution for parallel computing on CUDA-enabled GPUs supporting the standard C programming language together with computing numerical libraries. To accelerate ME processing, our approach adopts the CUDA-based GPU as the parallel computing platform.

To enable accurate modeling and efficient calculation of motion fields for image sequences with the presence of both slight and drastic brightness changes, this paper

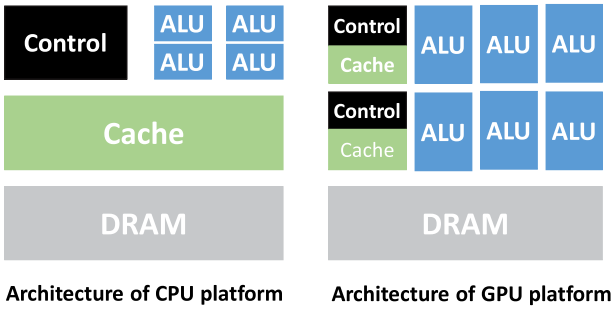


Fig. 1. Comparison between CPU and GPU architectures.

proposes a novel three-stage ME approach based on GPU. It makes three major contributions as follows:

(i) We propose a novel data structure named brightness distribution matrix (BDM), which can be used to accurately model the brightness of regions. During the initialization of motion fields, BDM can be used to effectively calculate neighboring pixels based on our defined BDM consistency constraint.
(ii) We introduce an efficient energy function which can be used to correct and optimize inaccurate motion vectors identified during the denoise of initial motion fields. By using this function, we can obtain motion vectors in a more accurate way.
(iii) We parallelize our proposed ME algorithm based on GPU, which can reduce the overall ME time.

Compared with existing ME approaches, our approach not only can enable the more accurate estimation of steady fluid flows with slight motion, but also can be used to estimate regions with abrupt brightness changes.

The rest of this paper is organized as follows. After introducing the related work in Sec. 2, Sec. 3 presents the major steps of our sequential ME algorithm including initialization, denoising, and optimization. To accelerate the processing time, Sec. 4 introduces the GPU version of our proposed ME approach in details. Based on the experimental results on a well-known benchmark, Sec. 5 shows the efficacy of our approach from the perspectives of accuracy and performance. Finally, Sec. 6 concludes the paper.

2. Related Work

Since motion estimation plays an important role in multimedia applications, various ME algorithms have been intensively investigated. For example, Horn and Schunck[9] introduced the first method for ME of objects among image sequences. Based on the brightness consistency-based constraints, their approach is widely used in the image sequences of solid. In Ref. 10, Lucas and Kanade presented an ME approach based on the pyramid model. The proposed approach not only can reduce the limitation of location window, but also can improve the accuracy of estimation. Although this approach is effective in dealing with drastic brightness changes, the accuracy of this method decreases significantly when there are only slight brightness changes.

To efficiently conduct ME of fluids with slight brightness changes, various continuity equation-based algorithms have been proposed.[11–18] For example, Tistarelli[11] and Uras *et al.*[12] proposed the gradient consistency, which can be used as the constraints in optical flow models. Their approach can enhance the ME robustness when processing little brightness variations. In Ref. 13, Corpetti *et al.* applied continuity equations to an optical flow model. Their approach can improve the ME accuracy when estimating fluids with slight brightness changes. In Ref. 14, Zhou *et al.* used the affine motion model to estimate the velocity of cloud motion. By using their

approach, the accuracy of estimating satellite images with slightest changes is largely increased. In Ref. 15, Nakajima *et al.*[15] adopted both the Navier–Stokes equation and a specific continuity equation as velocity constraints. The experimental results demonstrated the effectiveness of their approach when the flow is stable. However, all the above methods are based on continuity equation constraints, which cannot accurately handle regions with drastic brightness changes.

To further improve the accuracy of flow estimation, various approaches have been studied based on the work of Ref. 9. For example, Sun *et al.*[23] presented a novel ME method which introduces a nonlocal term that robustly integrates flow estimates over large spatial neighborhoods. Their approach can largely improve the accuracy of ME. In Ref. 24, Nilanjan proposed an effective flow computation framework to estimate the flow velocity vectors. A new data fidelity term is introduced to enable more accurate motion estimation. Although these approaches are promising in obtaining accurate ME results, there still exist unsatisfying ME results for some complex fluid scenarios. To the best of our knowledge, our approach is the first attempt based on GPU that tries to improve the ME accuracy and performance of fluid flows involving both drastic and slight brightness changes.

3. Sequential Motion Estimation Based on BDM

This section presents the sequential version of our ME approach in detail. In order to obtain motion fields for image sequences, we develop an ME approach which consists of three stages: (i) Based on BDM, the initialization stage calculates neighboring pixels based on our defined BDM *consistency constraint*. In this way, we can obtain primitive motion fields from images. (ii) The second stage (i.e., denoise stage) scans the previous obtained motion field to identify potential inaccurate vectors. (iii) Based on our proposed energy function, the third stage (optimization stage) corrects and optimizes the inaccurate motion vectors identified during the denoise stage. The following subsections will give the details of our three-stage ME method.

3.1. *Initialization of motion fields*

The motion vector of a pixel is calculated using the coordinate displacement between the pixel in the current frame and the corresponding pixel in the reference image. Since adjacent frames are almost the same and coordinate changes are mainly caused by the movements of objects or cameras, the brightness of one pixel should be similar to the regional brightness of its surrounding pixels. Particle image velocimetry (PIV) is widely adopted for flow visualization and analysis. In PIV, the brightness of a local region is described as

$$R(x,y,t) = \begin{pmatrix} I(x-w,y-w,t) & \cdots & I(x+w,y-w,t) \\ \vdots & \ddots & \vdots \\ I(x-w,y+w,t) & \cdots & I(x+w,y+w,t) \end{pmatrix}, \quad (1)$$

where w denotes the size of the sampling window and $I(x, y, t)$ denotes the brightness of pixel $p(x, y)$ at time t.

Generally, ME approaches assume that the regional brightness has a constant value. Nonetheless, slight brightness changes exist very frequently in real images. If we use formula (1) to present the regional brightness, the slight brightness may easily lead to the generation of incorrect motion vectors. In our approach, we calculate one BDM for each pixel in each frames. Assuming that the current pixel at time t is at location $p(x, y)$, we define the BDM of this pixel using a one-dimensional vector in the form of

$$\mathrm{BDM}(x, y, t) = (S^0(x, y, t), \ldots, S^{25}(x, y, t)), \tag{2}$$

where

$$S^L(x, y, t) = \begin{pmatrix} D^L_{r_0} & \cdots & D^L_{r_3} \\ \vdots & \ddots & \vdots \\ D^L_{r_{12}} & \cdots & D^L_{r_{15}} \end{pmatrix}$$

denotes the data structure that keeps the distance and brightness information of specific pixels. Here, L denotes the brightness level of pixels ranging from 0 to 25. Our approach maps the traditional 256 brightness levels (i.e., 0–255) into 26 levels (i.e., 0–25) by dividing them by 10. In this way, we can significantly reduce the chance of BDM consistency violations as defined in Eq. (7). This is because the larger granularity of brightness sampling will lead to smaller brightness changes between adjacent captured images.

To calculate the value of $D^L_{r_i}$ used in $S^L(x, y, t)$, we define r_i $(i = 0, \ldots, 15)$ which denotes one subblock of the pixel's surrounding area using a 4×4 matrix in the form of

$$r_i = \begin{cases} \begin{pmatrix} p(x+4m-8, y+4n-8) & \cdots & p(x+4m-5, y+4n-8) \\ \vdots & \ddots & \vdots \\ p(x+4m-8, y+4n-5) & \cdots & p(x+4m-5, y+4n-5) \end{pmatrix} & i = 0, 1, 4, 5, \\ \begin{pmatrix} p(x+4m-7, y+4n-8) & \cdots & p(x+4m-7, y+4n-8) \\ \vdots & \ddots & \vdots \\ p(x+4m-7, y+4n-5) & \cdots & p(x+4m-7, y+4n-5) \end{pmatrix} & i = 2, 3, 6, 7, \\ \begin{pmatrix} p(x+4m-8, y+4n-7) & \cdots & p(x+4m-8, y+4n-7) \\ \vdots & \ddots & \vdots \\ p(x+4m-8, y+4n-4) & \cdots & p(x+4m-5, y+4n-4) \end{pmatrix} & i = 8, 9, 12, 13, \\ \begin{pmatrix} p(x+4m-7, y+4n-7) & \cdots & p(x+4m-7, y+4n-7) \\ \vdots & \ddots & \vdots \\ p(x+4m-7, y+4n-4) & \cdots & p(x+4m-5, y+4n-4) \end{pmatrix} & i = 10, 11, 14, 15, \end{cases} \tag{3}$$

where $m = i\%4$ and $n = i/4$. For the pixel $p(x, y)$ at time t, we define a 17×17 searching matrix (SM) to denote the pixel's surrounding area, which is in the form of

$$\mathrm{SM}(x, y, t) = \begin{pmatrix} \boxed{r_0} & \boxed{r_1} & \vdots & \boxed{r_2} & \boxed{r_3} \\ \boxed{r_4} & \boxed{r_5} & \vdots & \boxed{r_6} & \boxed{r_7} \\ \cdots & \cdots & p(x, y) & \cdots & \cdots \\ \boxed{r_8} & \boxed{r_9} & \vdots & \boxed{r_{10}} & \boxed{r_{11}} \\ \boxed{r_{12}} & \boxed{r_{13}} & \vdots & \boxed{r_{14}} & \boxed{r_{15}} \end{pmatrix}. \tag{4}$$

Note that when processing SMs, we only consider the pixels located in $r_i \times (0 \leq i \leq 15)$. All the other pixels in the same row or same column as $p(x, y)$ are neglected. We introduce the function $\mathrm{check}(x_i, y_i, L)$ to check whether the brightness level of pixel $p(x_i, y_i)$ is L. The format of the function is as follows:

$$\mathrm{check}(x_i, y_i, L) = \begin{cases} 1 & \text{the brightness level of } p(x_i, y_i) \text{ is equal to } L, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Based on Eqs. (4) and (5), we can calculate the value of $D_{r_i}^L$ in $S^L(x, y, t)$:

$$D_{r_i}^L = \sum_{(x_i, y_i) \in r_i} \left(\mathrm{Dis}(P(x_i, y_i), P(x, y)) \cdot \mathrm{check}(x_i, y_i, L) \right), \tag{6}$$

where $P(x, y)$ denotes the coordinate of the pixel $p(x, y)$ in the plane and the function $\mathrm{Dis}(P(x_i, y_i), P(x, y))$ calculates the Euclidean distance between the pixels $p(x_i, y_i)$ and $p(x, y)$.

Since fluid motion in a region is continuous and the time interval between two images is very small, the BDM of the pixel $p(x, y)$ at time t and its peer pixel $p(x', y')$ at time $t + 1$ in the reference image can be assumed to be equal. Based on this observation, our approach adopts the *consistency constraint* to obtain motion vectors by finding the corresponding pixel in the reference image with the same BDM as the point in the original image. The constraint is in the following form:

$$\mathrm{BDM}(x, y, t) = \mathrm{BDM}(x', y', t + 1) = \mathrm{BDM}(x + u,\, y + v,\, t + 1), \tag{7}$$

where $\mathrm{BDM}(x, y, t)$ denotes the BDM of pixel $p(x, y)$ at time t, $\mathrm{BDM}(x + u, y + v, t + 1)$ denotes the BDM of pixel $p'(x', y')$ at time $t + 1$, and u and v denote the horizontal and vertical displacements of the same pixel in two different images, respectively. Note that pixel $p'(x', y')$ denotes the peer pixel in the reference image.

Since the *consistency constraint* can be easily violated in real images, a *distance function* can be used to approximate the effects of *consistency constraint*. The

distance function is in the following format:

$$D(x, y, u, v, t) = \sum_{L=0}^{25} \sum_{i=0}^{15} (S^{L,i}(x, y, t) - S^{L,i}(x + u, y + v, t + 1)), \tag{8}$$

where $S^{L,i}(x, y, t) = D_{r_i}$, $-8 \leq u \leq 7$ and $-8 \leq v \leq 7$. For all pixels $p(x, y)$ at time t, we use the above equation individually to calculate their distance. By identifying the pixel $p'(x', y')$ at time $t + 1$ with minimal distance defined in Eq. (8), we can obtain the motion vector $(u, v) = (x' - x, y' - y)$ based on Eq. (7).

3.2. *Denoising motion fields*

Due to the movement of cameras and light sources, the brightness changes of fluid flows are very common, which can inevitably result in inaccurate ME results. In the denoise stage, our approach tries to filter out inaccurate motion vectors from the motion field obtained in the first stage. Note that pixels should have similar motion directions within a small region of fluid flows. Therefore, we assume that the motion vector of a pixel is inaccurate if it is quite different from the motion vectors of other pixels in the same region. To improve the overall accuracy of motion fields, such kind of motion vectors should be identified for further improvements.

Algorithm 1 presents the detail of our approach for identifying whether the motion vector of $p(x, y)$ is a noise in a given motion field. This algorithm has three inputs: x and y denoting the x-component and y-component of pixel $p(x, y)$ to be processed and mf indicating the motion field of a specific image. Our approach assumes that the motion vector of a pixel has the same direction as the motion vectors of its neighboring pixels. For each target pixel $p(x, y)$, we designate a denoising window whose height and width are both 60 as indicated by the loop variables i and j in Algorithm 1, respectively. Note that both the inner and outer iterations have a step length of 10, and $\text{mf}(x, y)$.flag has an initial value of *false* indicating the motion vector of $p(x, y)$ is not denoised. In this algorithm, step 1 initializes the variable D_i $(i = 0, 1, 2, 3)$, which denotes the number of motion vectors located in the ith quadrant. We use the integers (i.e., 0, 1, 2, 3) to indicate the four different quadrants. Steps 2–5 count the numbers of motion vectors located in different quadrants. Step 6 figures out the quadrant that has the most motion vectors. We use the function Quad(mv) to obtain the quadrant index (i.e., 0, 1, 2, 3) of the motion vector mv. Assuming that the kth quadrant has the most motion vectors, step 7 checks whether $\text{Quad}(\text{mf}(x, y))$ equals to k, where $\text{mf}(x, y)$ indicates the motion vector of pixel $p(x, y)$. If they are not equal, we will set the flag of $\text{mf}(x, y)$ to *true*. The *true* flag indicates that the achieved motion vector $\text{mf}(x, y)$ is not accurate and it needs to be corrected and optimized in the optimization stage as described in Sec. 3.3.

Algorithm 1. Denoise Motion Fields

Input: (i) x, which is the x-component of pixel $p(x, y)$ to be denoised.
(ii) y, which is the y-component of pixel $p(x, y)$ to be denoised.
(iii) mf, which is the motion field of a specific image.
Output: A denoised motion vector for $p(x, y)$

```
1. MD0 = MD1 = MD2 = MD3 = 0;
for i from −60 to 60 step 10 do
    for j from −60 to 60 step 10 do
        if mf(x + i, y + j).u < 0 && mf(x + i, y + j).v > 0 then
            2. MD0 + +;
        end
        if mf(x + i, y + j).u > 0 && mf(x + i, y + j).v > 0 then
            3. MD1 + +;
        end
        if mf(x + i, y + j).u > 0 && mf(x + i, y + j).v < 0 then
            4. MD2 + +;
        end
        if mf(x + i, y + j).u < 0 && mf(x + i, y + j).v < 0 then
            5. MD3 + +;
        end
    end
end
6. find MDk (0 ≤ k ≤ 3) so that MDk ≥ MDi (i = 0, 1, 2, 3);
if Quad(mf(x,y)) ! = k then
    7. mf(x, y).flag = true;
end
end
```

3.3. *Optimization with energy function*

In order to get more accurate ME results while keeping motion consistency and movement continuity, proper ME optimization methods[23] should be adopted. For example, the active contour model has been adopted in many ME variants. Conforming to active contour models, energy functions[13,16,17,19] have been widely used in optical flows. Most of existing *energy functions* are in the following form:

$$E(u, v) = E_{\text{data}}(u, v) + \alpha E_{\text{regularization}}(u, v)\,, \tag{9}$$

where $E_{\text{data}}(u, v)$ is derived from a given continuity equation and $E_{\text{regularization}}$ is generated from a specific smoothness consistency constraint.[9] The parameter α balances the influences between two force terms in the function. Although the energy functions in this form are promising for ME, they cannot estimate regions with drastic brightness changes accurately in most cases. Since the pixels in the same region have similar motion vectors, under the guidance of neighboring motion vectors, the inaccurate motion vectors can be further improved. For the pixel $p(x, y)$ at

time t, we impose a novel term called *smoothness constraint*:

$$E_{\text{smooth}}(x, y, u, v, t) = \sum_{i=x-n}^{x+n} \sum_{j=y-n}^{y+n} (u - u_{ij})^2 + (v - v_{ij})^2 , \tag{10}$$

where u_{ij} and v_{ij} denote the x-component and y-component of the motion vector for pixel $p(x + i, y + j)$ at time t, respectively. In this equation, n denotes the searching window size.

Since a pixel in the reference image has similar brightness level as the original pixel, we impose *image constraints* based on Eq. (8), which is in the form of

$$E_{\text{image}}(x, y, u, v, t) = D(x, y, u, v, t) . \tag{11}$$

By combining all the above constraints, we can obtain our *energy function* in the form of

$$E(x, y, u, v, t) = E_{\text{image}}(x, y, u, v, t) + \alpha E_{\text{smooth}}(x, y, u, v, t) . \tag{12}$$

Note that motion vectors on the boundary of one subblock may have different directions compared with other within the same subblock. α should be set to a relatively small value in order to follow the motion trend in the subblock.

4. Parallelized Motion Estimation Using CUDA

Among all the three stages of our sequential approach as described in Sec. 3, there exist lots of independent calculation tasks which can be fully parallelized. For example, the motion vector calculations of pixels are independent which are quite time-consuming. To accelerate our ME algorithm, we choose CUDA[25] as our computing platform based on GPU. When applying GPU to accelerate our ME approach, we need to figure out how to efficiently allocate such subtasks to GPU cores to enable sufficient data-level parallelism. Since the initialization of motion fields and the optimization stage cost most of the calculation time, we only parallelized these two stages to achieve better ME performance. The following two subsections will give the GPU implementation details of these two stages.

4.1. *Initialization of motion fields*

In our sequential ME approach, we calculate the results of distance function between an original pixel and all the candidate pixels within the searching window of the reference image. The candidate pixel with the minimal result of distance function is reckoned as the desired pixel. Since each calculation process of each pixel is independent and the calculations share the same procedure, we can naturally parallelize this process using GPU.

Since motion estimation based on BDM involves huge number of independent distance function computations, we can resort to GPU platform to parallelize such computations in the initialization stage. We organize pixels of an image using

a two-dimensional matrix and assign each pixel in the image with a thread block. After the BDM calculation of pixels, a thread is allocated in the searching window of the second reference image for the purpose of distance function calculation. In this way, each thread conducts the calculations of distance function and the pixel with the minimal distance is reckoned as the counterpart. During the initialization stage, a CUDA kernel function is invoked to generate motion fields for each section. Limited by the searching method itself, the boundary motion vectors of each section cannot be decided until the optimization stage. Based on Eq. (8), we can calculate the BDM of every pixel.

Note that we do not parallelize the BDM calculation using GPU, since there are lots of branches in the BDM calculation. The BDM construction is used for distance-based motion vector calculation as defined in Eqs. (7) and (8). Algorithm 2 presents the details of our parallelized method for distance-based motion vector calculation. In this algorithm, there are three inputs: x and y denoting the x-component and y-component of pixel $p(x, y)$ to be processed and mf denoting the motion field of a specific image. Assume that the counterpart pixel of $p(x, y)$ in the reference image is located within the searching window. We can obtain the distance value for each pixel in the searching window and select the pixel with minimal value as the counterpart pixel. In this algorithm, step 1 launches the kernel function and loads the required

Algorithm 2. Distance-Based Motion Vector Calculation Using GPU

Input: (i) x, which is the x-component of pixel $p(x, y)$ to be initialized.
(ii) y, which is the y-component of pixel $p(x, y)$ to be initialized.
(iii) mf, which is the motion field of a specific image.

Output: An initialized motion vector for $p(x, y)$.

Initial(x, y, mf)**begin**
1. Launch a kernel for the first image;
2. assign a thread block for pixel $p(x, y)$;
for i from -8 to 7 **do**
for j from -8 to 7 **do**
/* Calculating distance using Eq. (8) */
$D_{i,j} = 0$;
for L from 0 to 25 **do**
for k from 0 to 15 **do**
3. thread$(i + 8, j + 8)$ compute
$D_{i,j} = D_{i,j} + S^{L,k}(x, y, t) - S^{L,k}(x + i, y + j, t + 1)$;
end
end
end
end
4. $D_{m,n} = \text{MIN}(\{D_{i,j} \mid -8 \le i, j \le 7\})$;
5. $\text{mf}(x, y).u = m$;
6. $\text{mf}(x, y).v = n$;
return $\text{mf}(x, y)$;
end

BDMs for both the pixels $p(x, y)$ and $p'(x', y')$ into the GPU global memory. Step 2 assigns a thread block of 256 threads for computing BDM(x, y). Note that the searching window of ME in the initialization is different from the SM introduced in Eq. (3). The size of searching window is a two-dimensional 16×16 neighborhood. Step 3 computes the distances between the pixel $p(x, y)$ and the pixels in the searching window of the reference image. Step 4 identifies the pixel in the reference image with minimal distance. Since the process of minimal value finding involves lots of control flow branches which can easily cause branch divergence, our approach uses CPU rather than GPU to find the pixel with minimal distance in the reference image. Steps 5 and 6 set the motion vector for $p(x, y)$. Finally, the algorithm returns an initialized motion vector for pixel $p(x, y)$.

Algorithm 3. Optimization for Denoised Motion Vector

Input: (i) x, which is the x-component of pixel $p(x, y)$ to be optimized.
(ii) y, which is the y-component of pixel $p(x, y)$ to be optimized.
(iii) mf, which is the motion field of a specific image.

Output: An optimized motion vector for $p(x, y)$.

```
Optimize(x, y,mf)begin
    if mf is denoised then
        for i from −6 to 6 do
            for j from −6 to 6 do
                1. compute the image_energy;
                for a from −10 to 10 step 5 do
                    for b from −10 to 10 step 5 do
                        if mf(x + b, y + a) is not denoised then
                            /* Calculate smooth energy using Eq. (10) */
                            2. smooth_energy =
                            smooth_energy+smooth_energy(x, y, i, j);
                        end
                    end
                end
                /* Calculate the total energy using Eq. (12) */
                3. total_energy = deviation + smooth_energy;
                if total_energy < min_energy then
                    4. mf(x, y).u = i;
                    5. mf(x, y).v = j;
                    6. min_energy = total_energy;
                end
            end
        end
    end
    return mf(x, y);
end
```

4.2. *Parallelized optimization based on energy constraint*

In the optimization stage, we introduce an efficient energy function which can be used to correct and optimize inaccurate motion vectors identified during the denoise stage. By using this function, we can obtain motion vectors in a more accurate way. In an image, different sections may have different motion directions. Note that it is difficult to find the motion vectors on the boundary of sections since the BDMs of some pixels in different regions can affect the motion vector calculation on the boundary of current section. Unlike the section division in the initialization stage, in the optimization stage we divide the images in a different way, which can effectively improve the motion vector calculations on the boundary of divided regions.

Algorithm 3 presents the details of our parallelized optimization method. Here, we only optimize the pixels that have been denoised previously due to their potential vulnerability. Step 1 calculates the image energy between the pixel $p(x, y)$ and those in the range of searching window using Eq. (11). Note that our approach adopts a searching window with a size of 13×13 here. Step 2 uses the *smoothness constraint* defined in Eq. (10) to calculate the smooth energy. Step 3 calculates the sum of

Algorithm 4. Our GPU-Based Motion Estimation Algorithm

Input: (i) img1, which is the original image.
(ii) img2, which is the reference image.
(iii) mf, which is the motion field of img1.
Output: The final motion field of img1.
ME(img1, img2, mf)**begin**
/* CPU-based BDM initialization */
for each $p(x, y) \in$ img1 and each $p'(x', y') \in$ img2 **do**
1. Calculate BDM(x, y) and BDM(x', y');
end
/* GPU-based motion fields initialization */
for each $p(x, y) \in$ img1 **do**
2. mf$(x, y) =$ Initial$(x, y,$ mf$)$;
end
/* The CPU-based denoise stage */
for each $p(x, y) \in$ img1 **do**
3. Denoise$(x, y,$ mf$)$
end
/* The GPU-based optimization stage */
for each $p(x, y) \in$ img1 **do**
if mf(x, y).flag $==$ *true* **then**
4. mf$(x, y) =$ Optimize$(x, y,$ mf$)$;
end
end
return mf;
end

image_energy obtained in step 1 and smooth_energy in step 2, which equals to total_energy as defined in Eq. (12). If total_energy is less than the currently obtained min_energy, we will set the motion vector of $p(x, y)$ using the current values of i and j, and reset the min_energy. Finally, the algorithm reports an optimized motion vector for pixel $p(x, y)$.

4.3. *GPU-based motion estimation algorithm*

Algorithm 4 shows the procedure for our GPU-based ME algorithm. Note that the BDMs for both original and reference images are defined as global data structures. Their elements can be accessed during the following ME stages. In this algorithm, step 1 calculates the BDMs of each pixel for both the original image (i.e., img1) and reference image (i.e., img2). Step 2 initializes the motion field using the distance-based approach as described in Algorithm 2. Step 3 identifies inaccurate motion vectors from the whole motion field. Step 4 optimizes the motion vectors that are identified during the denoise stage. Finally, the ME algorithm reports an optimized motion field for the given images.

5. Experimental Results

To evaluate the efficacy of our proposed approach, we conduct experiments on multiple series of real images. All the benchmarks are collected from a well-known dynamic texture library named DynTex,[22] which is a large database of high-quality videos. We developed a CUDA-based ME tool which implements all the three proposed stages based on C++ programming language. Our experiment was carried out on a Windows machine with 3.30 GHz Intel i5 CPU and NVIDIA GTX 645 GPU (with 576 CUDA cores). To show that our approach outperforms existing methods in terms of ME quality, Sec. 5.1 makes comparisons with two state-of-the-art methods.[23,24] Section 5.2 discusses the performance issues in order to demonstrate the performance of our GPU-based ME approach.

5.1. *Accuracy analysis*

To show the efficacy of our ME approach, we conducted our ME algorithm on four typical scenarios, namely motions of ocean wave (i.e., video "649dd10"), river stream (i.e., video "649i720"), waving flags (i.e., video "646a210"), and pumping fountain (i.e., "6484d10"). The video "649dd10" presents a scenario of ocean wave movement. This case involves a large number of drastic brightness changes due to the quick movement of sea water. The video "649i720" shows the movement of river stream which consists of slight brightness change. The video "646a210" presents the movement of waving flag which is irregular due to strong winds. The video "6484d10" shows a scenario of pumping fountain. This case consists of lots of irregular fluid movements. The reason why we chose these four cases is because the

sequences involve various irregular fluid movements and brightness changes, which can reflect the efficacy of our approach within most fluid motion cases. In Fig. 2, the images on the left side are original images and the images on the right side are the ones obtained using our GPU-based approach. Since the difference between the original image and reference image is small, we do not present the reference image in Fig. 2. In the right images, red arrows denote the motion vectors obtained from the initialization stage, whereas green arrows indicate the motion vectors generated in the optimization stage. From Fig. 2, we can find that our GPU-based method can achieve ME results with higher accuracy than the other two approaches[23,24] under different kinds of fluid scenarios. For example, the result of "649dd10" demonstrates that our GPU-based method is capable of estimating motion of regions with drastic brightness changes. The result of "649i720" shows the efficacy of our method when estimating regions with slight brightness changes.

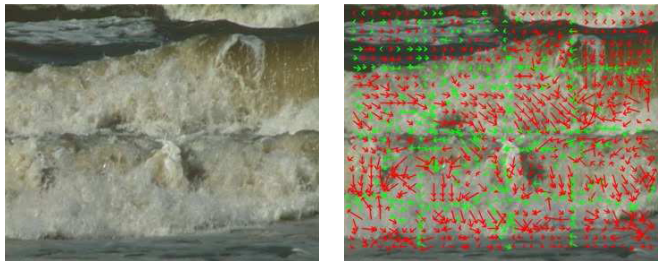
To further show the advantages of our approach, we compared our approach with two other state-of-the-art ME approaches.[23,24] Figure 3 shows the comparison results. We can observe that the ME approaches[23,24] cannot accurately detect the varying brightness changes. From examples shown in Figs. 3(a)–3(c), the ME results generated by Refs. 23 and 24 indicate that large parts of the images are motionless, which do not reflect the real fluid motions. However, our approach can accurately capture such complex scenarios. It is less likely to be affected by large displacements and drastic brightness variations. In Fig. 3(d), there is a rock at the left-bottom corner of the image which causes irregular fluid motions in that area. Unlike the reference methods (Refs. 23 and 24), our approach can accurately express such irregular fluid motions.

5.2. *Performance analysis*

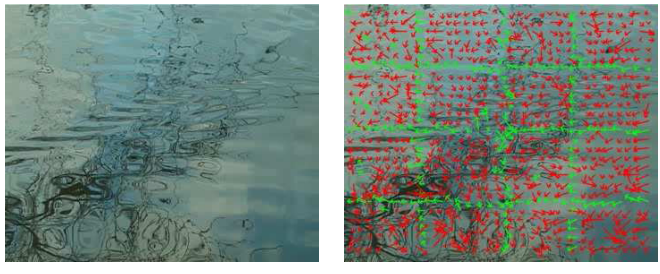
Although our approach is more accurate than existing approaches in dealing with images involving both drastic and slight brightness changes, it is based on BDM which requires much more computations. To improve the ME performance, we propose GPU-based implementation of our ME method. To fully utilize the potential of GPU to enable the data-level parallelism, our approach divides images into small sections. Therefore, the performance of our tool mainly depends on the image division strategy and the subtask parallelization strategy. This subsection investigates the performance of our approach under different strategies.

5.2.1. *ME results using different division strategies*

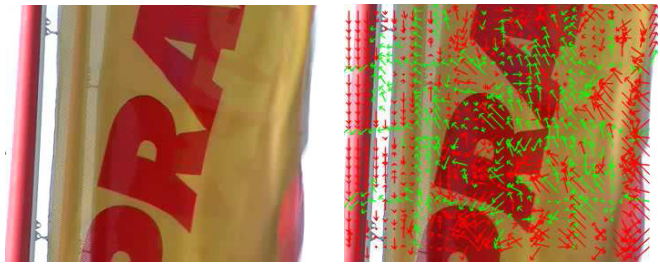
When images are not divided, the calculation workload of BDMs could be enormous. To enable parallel ME processing, our approach divides images into multiple sections before transferring them into device memory during the initialization stage. One key issue in our approach is the granularity of the divided sections. Since the size of the search window in our approach is 16×16, images should be divided into no more



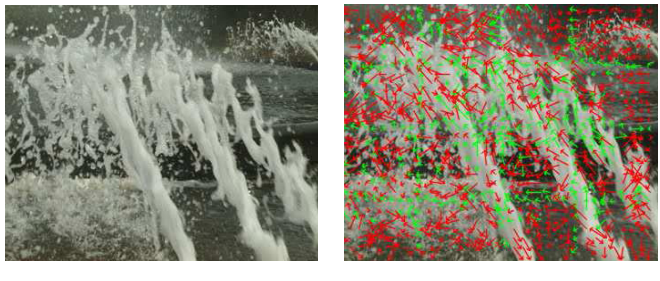
(a)



(b)



(c)



(d)

Fig. 2. Experimental results using ME approach. (a) The 33rd frame in video "649dd10", (b) The 0th frame in video "649i720", (c) The 0th frame in video "646a210" and (d) The 0th frame in video "6484d10".

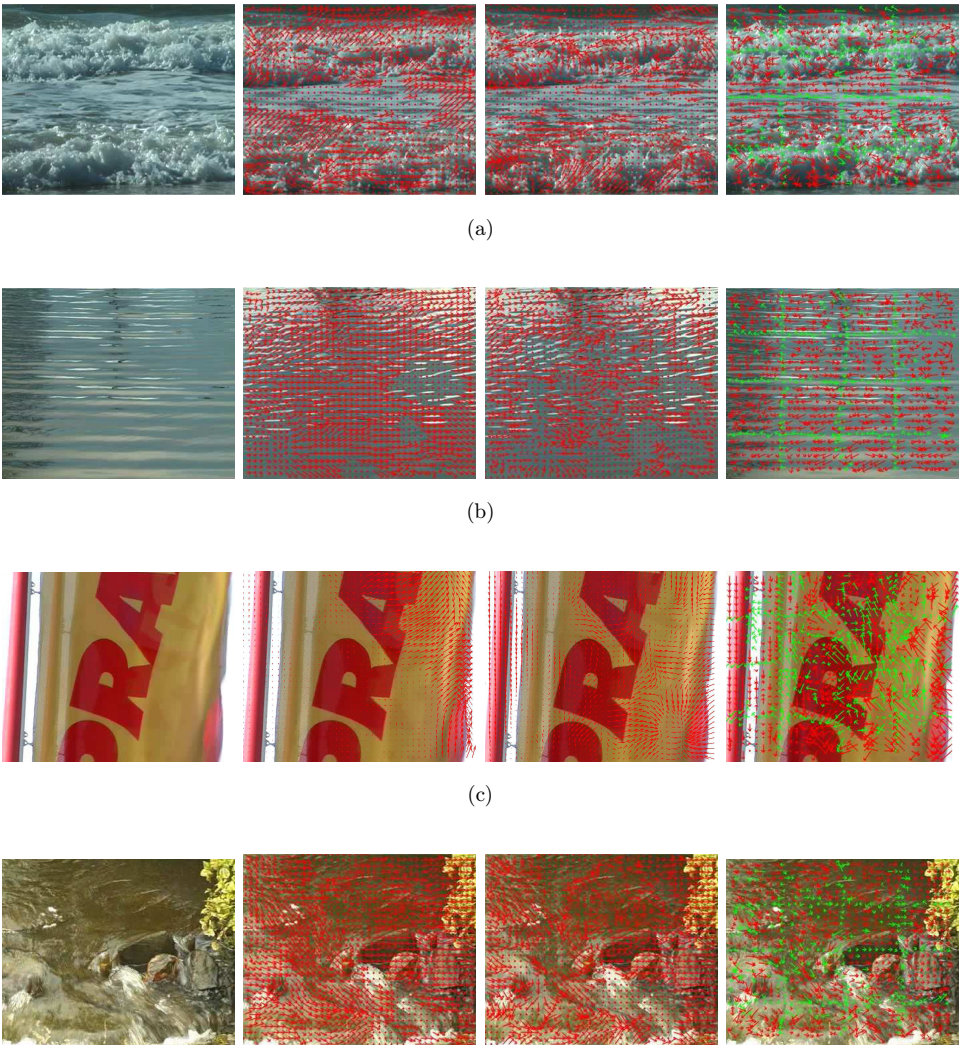


Fig. 3. ME results using different approaches. The images of each subfigure from left to right denote the original image, the ME results using Refs. 23 and 24, and our approach, respectively. (a) The 0th frame in video "54ab110", (b) The 0th frame in video "649ib10", (c) The 0th frame in video "646a210" and (d) The 218th frame in video "6482910".

than 16 sections. This is because the size of each section is smaller than the search window if we divide images into more than 16 sections, which can lead to a large deviation of boundary motion vectors. During the optimization stage, images should be divided into more than four sections, which can facilitate the computations of energy function. Moreover, the number of sections should be in the form of a square

Table 1. Performance comparison results of two division strategies.

	Video 649dd10	Video 649i720	Video 54ab110	Video 649ib10	Video 6482910	Video 646a210
D1.init (s)	43.24	41.52	41.00	40.95	41.21	41.29
D1.den (s)	0.01	0.01	0.00	0.00	0.00	0.00
D1.opt (s)	7.84	7.78	7.82	7.63	7.59	7.43
D1.total (s)	51.09	49.31	48.82	48.58	49.80	48.72
D2.init (s)	38.89	37.31	37.52	37.74	39.1	38.95
D2.den (s)	0.00	0.01	0.01	0.01	0.00	0.00
D2.opt (s)	8.44	8.39	8.34	8.45	8.49	8.42
D2.total (s)	47.33	45.71	45.87	46.20	47.59	47.37

number. In other words, we only have two division choices for our approach, i.e., 9 and 16.

Table 1 presents the performance comparison results under different division strategies. Strategy D1 denotes the case of dividing images into nine sections, while the strategy D2 denotes the case of dividing images into 16 sections. The first four rows of the table show the execution time details using the strategy D1, and the last four rows present the execution time details using the strategy D2. For each strategy, we investigate both the execution time for each stage and the overall ME time. From this table, we can find that strategy D2 outperforms strategy D1, since ME using strategy D2 consumes less time. For example, strategy D2 only needs 47.33 s to get the ME results of video "649dd10" while strategy D1 takes 51.09 s. This is because both strategies do not calculate the boundary motion vectors during the initialization stage. Since strategy D2 divides images into more sections, the initialization calculation workload can be largely reduced. Furthermore, the boundary motion vectors that are not initialized can be obtained rapidly in the optimization stage. Therefore, the strategy D2 is a better choice for our approach. Note that all the experimental results are obtained based on strategy D2.

5.2.2. *ME results using different parallel strategies*

Parallel strategies for GPU threads play an important role in GPU-based ME performance optimization. Since the initialization stage involves a large number of independent computations, proper parallel strategies should be applied to improve ME performance. The following are two parallel strategy alternatives which assign subtasks of different granularities to GPU threads.

Strategy P1: For each pixel, we assign a thread. In other words, each thread computes all the distances (using *distance function*) between the specified pixel and all the pixels in the corresponding searching window.

Strategy P2: For each pixel, we assign a thread block with 256 threads. In order to calculate the distance function, each thread is responsible for the computation of distance function for one of the pixels within the searching window.

Table 2. Performance comparison results of two parallel strategies.

	Video 649dd10	Video 649i720	Video 54ab110	Video 649ib10	Video 6482910	Video 646a210
P1 (s)	66.36	65.74	67.83	66.71	68.74	67.39
P2 (s)	47.33	45.71	45.87	46.19	47.59	47.37

Table 3. Performance comparison results of four ME approaches.

	Video 649dd10	Video 649i720	Video 54ab110	Video 649ib10	Video 6482910	Video 646a210
Heuristic-based[23] (s)	28.2	30.6	30.6	30.0	30.0	30.2
ABD-based[24] (s)	33.91	33.93	34.11	34.68	34.85	34.93
CPU-based (s)	322.73	322.55	323.95	321.92	308.14	328.47
GPU-based (s)	47.31	45.70	45.86	46.19	47.59	47.37
Speedup	6.82	7.06	7.06	6.97	6.96	6.93

Table 2 presents the experimental results using the two proposed different parallel strategies. From this table, we can find that the execution times of strategy P2 are much smaller than those of strategy P1. This is mainly because solution P1 uses GPU to accomplish the comparison tasks which consist of many branch instructions. In this case, the GPU performance will be degraded due to branch divergence caused by branch instruction.

5.2.3. *Comparison of overall performances*

To provide the objective performance of our GPU-based approach, Table 3 shows the total execution times comparison among four ME algorithms. The first two rows of the table show the execution times using two state-of-the-art methods proposed in Refs. 23 and 24, respectively. The third row shows the execution times using the sequential version of our ME approach which is based on CPU. Compared to the works in Refs. 23 and 24, the total execution time of our sequential ME method is quite long, though it can achieve ME results with better accuracy. To reduce the overall ME time of our approach, the fourth row gives the ME execution time results using the parallel version of our ME approach based on GPU. From this table, we can find that our GPU-based approach can reduce the overall ME time. Compared to its sequential counterpart, the GPU-based approach can achieve an improvement of up to 7.06 times.

6. Conclusion

As a hot topic in computer vision, motion estimation for fluid flows has been widely investigated. However, so far there is a lack of approaches that can accurately

estimate fluid image regions with both slight and drastic brightness changes. Furthermore, due to the enormous computation workload in accurate motion estimation, how to reduce the ME time while guaranteeing expected ME accuracy is becoming a major bottleneck in ME research. To address the above issues, this paper presented a three-stage GPU-based method for efficient motion estimation of fluid images. In our approach, the regional brightness of images is modeled using our proposed brightness distribution matrix. By denoising initial motion fields, our approach enables the early identification of inaccurate motion vectors. Based on our proposed energy function, these inaccurate motion vectors can be tuned and optimized to reflect the real fluid motions. Since our approach requires more computation efforts than the existing ME methods, we developed a parallelized version of our approach based on GPU to accelerate the above ME steps. Experimental results show that our GPU-based approach not only can obtain better motion estimation results for complex fluid scenarios than the existing ME methods, but also can be executed efficiently in terms of runtime.

Acknowledgments

This work was partially supported by Grants from the Natural Science Foundation of China (Grant No. 91418203 and 61672230), Innovation Program of Shanghai Municipal Education Commission (14ZZ047), NSF Grants (CCF-1351054 (CAREER)), Shanghai Municipal NSF (16ZR1409000), and East China Normal University Outstanding Doctoral Dissertation Cultivation Plan of Action under the Grant PY2015047. A preliminary version[21] of this paper appeared in the *Proceedings of Asia Simulation Conference 2012*.

References

1. R. Lorentz and D. B. Benson, Deterministic and nondeterministic flowchart interpretations, *Int. J. Comput. Syst. Sci.* **27** (1983) 400–433.
2. Y. Yacoob and L. Davis, Recognizing human facial expressions from long image sequences using optical flow, *IEEE Trans. Pattern Anal. Mach. Intell.* **18** (1996) 636–642.
3. S. D. Peddada and R. McDevitt, Least average residual algorithm (LARA) for tracking the motion of Arctic sea ice, *IEEE Trans. Geosci. Remote Sens.* **34** (1996) 915–926.
4. A. Ottenbacher, M. Tomasini, K. Holmund and J. Schmetz, Low-Level cloud motion winds from Meteosat high-resolution visible imagery, *Weather Forecast.* **12** (1997) 175–184.
5. I. Cohen and I. Herlin, Nonuniform multiresolution method for optical flow and phase portrait models: Environmental applications, *Int. J. Comput. Vis.* **33** (1999) 29–49.
6. Z. Lu, Q. Liao and J. Pei, A PIV approach based on nonlinear filtering, *J. Electron. Inf. Technol.* **32** (2010) 400–404.
7. X. Shu, K. Shen and Y. Long, Method of medical image registration based on optical flow field, *Comput. Eng. Appl.* **44** (2006) 191–193.

8. H. Nogawa, Y. Nakajima and Y. Sato, Acquisition of symbolic description from flow fields: A new approach based on a fluid model, *IEEE Trans. Pattern Anal. Mach. Intell.* **19** (1997) 58–63.
9. B. Horn and B. Schunck, Determining optical flow, *Artif. Intell.* **17** (1981) 185–203.
10. B. D. Lucas and T. Kanade, An iterative image registration technique with an application to stereo vision, *Int. Joint Conf. Artificial Intelligence (IJCAI)* (1981), pp. 674–679.
11. M. Tistarelli, Multiple constraints for optical flow, *European Conf. Computer Vision (ECCV)* (1994), pp. 61–70.
12. S. Uras, F. Girosi, A. Verri and V. Torre, A computational approach to motion perception, *Biol. Cybern.* **60** (1988) 79–87.
13. T. Corpetti, É. Mémin and P. Pérez, Dense estimation of fluid flows, *IEEE Trans. Pattern Anal. Mach. Intell.* **24** (2002) 365–380.
14. L. Zhou, C. Kambhamettu and D. B. Goldof, Fluid structure and motion analysis from multi-spectrum 2D cloud image sequence, *Conf. Computer Vision and Pattern Recognition (CVPR)* (2000), pp. 2744–2751.
15. Y. Nakajima, H. Inomata, H. Nogawa, Y. Sato, S. Tamura, K. Okazaki and S. Torii, Physics-based flow estimation of fluids, *Pattern Recognit.* **36** (2003) 1203–1212.
16. E. Arnaud, É. Mémin, R. Sosa and G. Artana, A fluid motion estimator for Schlieren image velocimetry, *European Conf. Computer Vision (ECCV)* (2006), pp. 198–210.
17. J. Yuan, C. Schnörr and É. Mémin, Discrete orthogonal decomposition and variational fluid flow estimation, *J. Math. Imaging Vis.* **28** (2007) 67–80.
18. T. Kohlberger, É. Mémin and C. Schnörr, Variational dense motion estimation using the Helmholtz decomposition, *Int. Conf. Scale Space Methods and Variational Methods in Computer Vision (SSVM)* (2003), pp. 432–448.
19. H. Sakaino, Fluid motion estimation method based on physical properties of waves, *Int. Conf. Computer Vision and Pattern Recognition (CVPR)* (2008), pp. 1–8.
20. W. Chen and H. Han, 264/AVC motion estimation implementation on compute unified device architecture (CUDA), *Int. Conf. Multimedia and Expo (ICME)* (2008), pp. 697–700.
21. H. Zhuang and H. Quan, Fluid motion estimation based on energy constraint, *Asia Simulation Conf. (AsiaSim)* (2012), pp. 308–318.
22. University of La Rochelle, DynTex dynamic texture library (2010), http://projects.cwi.nl/dyntex/index.html.
23. D. Sun, R. Stefan and J. B. Michael, Secrets of optical flow estimation and their principles, *Int. Conf. Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 2432–2439.
24. R. Nilanjan, Computation of fluid and particle motion from a time-sequenced image pair: A global outlier identification approach, *IEEE Trans. Image Process.* **20** (2011) 2925–2936.
25. NVIDIA, CUDA C programming guide (2015), https://docs.nvidia.com/cuda/cuda-c-programming-guide.