GPU Based Computation of the Structural Tensor for Real-Time Figure Detection

Marcin Bugaj
AGH University of Science and Technology
Al. Mickiewicza 30
30-059 Kraków, Poland
mm.bugaj@gmail.com

Bogusław Cyganek
AGH University of Science and Technology
Al. Mickiewicza 30
30-059 Kraków, Poland
cyganek@agh.edu.pl

ABSTRACT

In this paper we present a real-time realization of the method of detection of local structures in images of predefined orientation. The method is based on an analysis of the structural tensor computed in monochrome and color images. Thanks to the GPU implementation of the low-level feature detection an order-of-magnitude speed-up was achieved compared to the software implementation. The method can be used for real-time detection of solid objects in HDTV streams as shown by many examples.

Keywords

Structural tensor, corner and edge detection, graphics card, real-time low level feature detection

1. INTRODUCTION

An analysis of low-level features in images constitutes a front-end in many computer vision systems. Thus, there is still an ongoing research on their fast and precise computation methods. One of the very promising methodologies relies on computation of the so called structural tensor (ST) which conveys information on multi-dimensional and multi-scale local neighborhoods of pixels in 2D, 3D and higher dimensional images [1][3].

In this paper we present a method of detection of local features in images of specific orientation. The method relies on prior computation of the ST which is proposed to be done in the graphic card (GPU). For this purpose the HIL library, provided by Cyganek *et al*. [3], as well as the CUDA development environment by nVidia® were used [10][11][12]. The presented method fits well to the concept of smart cameras which are able to precompute and transfer low-level features defined by a user. Thanks to this an order-of-magnitude speed-up was achieved which allows processing of the HDTV streams in real-time, as shown by experiments.

The paper is organized as follows. In section 2 we present an overview of figure detection based on multi-channel and multi-scale structural tensor. In section 3 the architecture of the processing platform is presented. Section 4 contains experimental results. The paper ends with conclusions provided in section 5.

2. FIGURE DETECTION WITH THE STRUCTURAL TENSOR

The structural tensor was first proposed by Bigün et al. [1], and then used by many authors in context of different applications [7]. With the structural tensor each pixel neighborhood can be investigated for their strength and orientations in terms of local intensity or color signal gradients. Having a local neighborhood of pixels Ω, centered at a point $\mathbf{x}_0$, the main idea is to determine a dominating directional vector $\mathbf{w}$ which is as close as possible to all gradients $\mathbf{q}_i$ in this neighborhood. This is depicted in Figure 1.

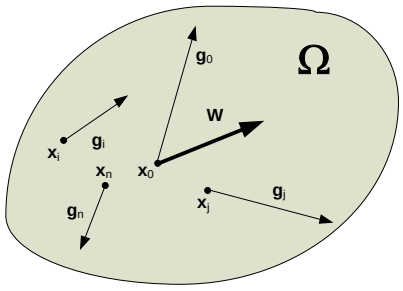

Figure 1. Computation of the dominating orientation vector w in a pixel neighborhood Ω based on signal gradients $\mathbf{q}_i$.

In other words, if such **w** can be determined, then the whole Ω can be represented solely by **w**. Moreover, its parameters, such as phase and magnitude provide us important information on a type of a neighborhood. As will be shown, this can be used to determine corners or local structures with specific orientations. In consequence, ST can be used to detect characteristic figures in images. In our application it was used to detect e.g. road signs in real time, as will be shown in experimental section.

More specifically, to compute **w** of Ω one needs to compute gradient vectors $\mathbf{q}_i$ for all points $\mathbf{x}_i \in \Omega$. For comparison of vectors their inner product is used. Thus, the vector **w** at a point x_0 is an estimator of an average orientation in Ω that fulfills the following optimization problem

$$\begin{aligned} &\arg\max_{\mathbf{w}} \left(\int_{\Omega} \left[\mathbf{q}(\mathbf{x}) \mathbf{w}^{\mathbf{T}}(\mathbf{x}) \right]^2 d\mathbf{x} \right) \\ &= \arg\max_{\mathbf{w}} (\mathbf{w}^{\mathbf{T}} \mathbf{T} \mathbf{w}), \end{aligned} \tag{1}$$

where **q** and **w** are column vectors, whereas **T** denotes the structural tensor, which is defined as follows

$$\mathbf{T}(\mathbf{x}) = \int_{\Omega} \mathbf{q}(\mathbf{x}) \mathbf{q}^{T}(\mathbf{x}) d\mathbf{x} \,. \tag{2}$$

The square of the inner product in (1) fulfils the invariant assumption on rotation of π radians. Otherwise parallel and anti-parallel configurations of vectors would cancel out. On the other hand, the outer product $\mathbf{q}\mathbf{q}^{T}$ in (2) conveys dimension of the tensor **T**.

In a case of multi-channel color images the gradient vector **q** can be defined as proposed by Di Zenzo [5]. In this approach summation of the partial gradient components in image channels is assumed. To find the ST for images with *M* channels we employ this idea to (2), as follows [3]

$$\begin{aligned} \mathbf{T} &= \int_{\Omega} \sum_{k=1}^{M} \left(\mathbf{q}_k(\mathbf{x}) \mathbf{q}_k(\mathbf{x}) \right) d\mathbf{x} = \\ &\sum_{k=1}^{M} \int_{\Omega} \left(\mathbf{q}_k(\mathbf{x}) \mathbf{q}_k(\mathbf{x}) \right) d\mathbf{x} = \sum_{k=1}^{M} \mathbf{T}_k . \end{aligned} \tag{3}$$

Thus the summation in (3) follows all gradient fields, each computed independently for each color channel in an image. Finally, let us observe that there are two dimensions involved in (1) and (3). The first directly follows dimension of the gradients, i.e. it is 2D for single image or 3D for video sequences, and so on. The second dimension comes from a number of image channels *M* in (3). Discrete realization of the structural tensor was analyzed by Hauβecker *et al*. [7]. This is given as follows

$$\hat{T}_{ij}(\rho, \xi) = F_{\rho}(R_i^{(\xi)} R_j^{(\xi)}) \,, \tag{4}$$

where $R_i^{(\xi)}$ a ξ-tap discrete directional operator and F_{ρ} is a smoothing kernel at scale ρ [3]. For $R_i^{(\xi)}$ we used the directional filters provided by Farid *et al*. [6].

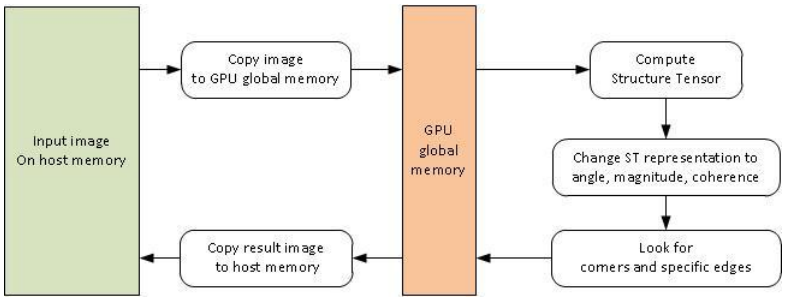

Figure 2. Processing path for corners and edges detection.

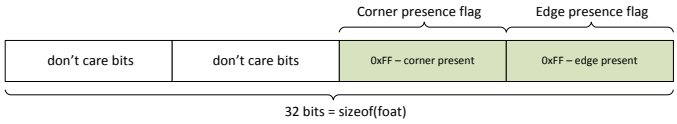

Figure 3. Pixel format of the output image.

In image processing the structural tensor is very useful in a slightly different form, i.e. computing phase and magnitude of the vector **w**. The phase of the vector **w** in (1), which corresponds to an eigenvector of the greatest eigenvalue of **T**, can be found analytically from the following representation [7]

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T = \begin{bmatrix} T_{xx} - T_{yy} & 2T_{xy} \end{bmatrix}^T . \qquad (\mathbf{5})$$

From the above the following is easily obtained

$$\varphi = \operatorname{atan} 2\left(\frac{2T_{xy}}{T_{xx} - T_{yy}} \right). \qquad (\mathbf{6})$$

This formula is directly used to find local structures in images with requested phase. In this case we need to additionally check trace of **T** which should be greater than a predefined threshold τ, that is

$$t = T_{xx} + T_{yy} > \tau . \qquad (7)$$

For corner detection the eigenvalues of **T** should be computed, as follows

$$\lambda_{1,2} = \tfrac{1}{2}\left(\left(T_{xx} + T_{yy} \right) \pm \sqrt{\left(T_{xx} - T_{yy} \right)^2 + 4T_{xy}^{\ 2}} \right). \qquad (\mathbf{8})$$

It can be shown [7] that the analysis of a type of local pixel neighborhoods can be based on the analysis of the local eigenvalues (8). However, the two eigenvalues can be joined together in a form of the following coherence component [1]

$$c = \left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \right)^2 . \qquad (\mathbf{9})$$

The above coefficient takes on 0 for ideal isotropic areas or structures with constant intensity value, and reaches 1 for ideally directional structure.

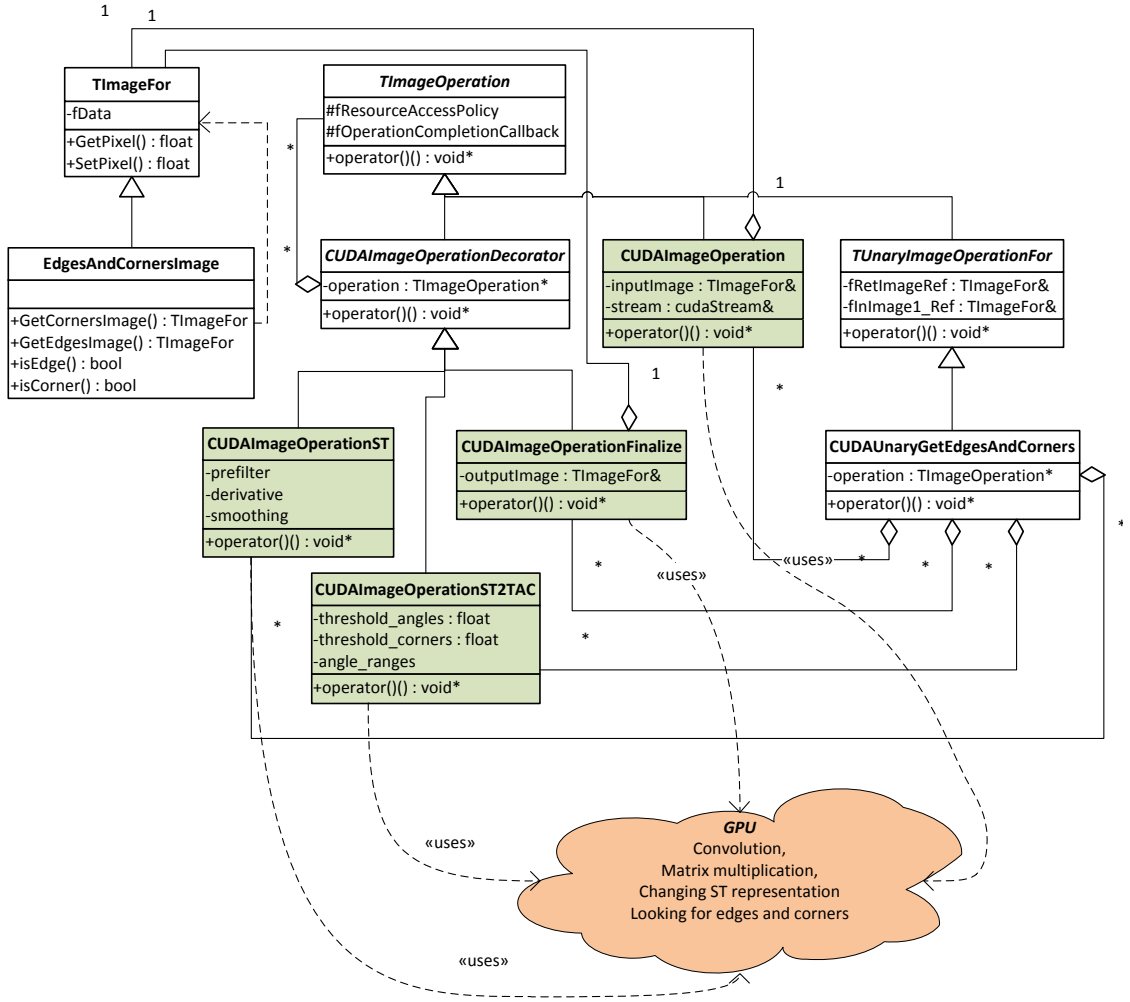

Figure 4. Architecture of the class hierarchy.

3. ARCHITECTURE OF THE IMPLEMENTATION PLATFORM

In implementation of the oriented structures in images (e.g. corners, edges, etc.) based on the aforementioned structural tensor, the CUDA graphics cards were used. The code was compiled with `nvcc` - enhanced-C compiler. HIL library was utilized as a tool for image and image operation representation [3]. Figure 2 shows the data flow for an operation of searching corners and edges. The image is first copied to the GPU global memory. Structure tensor is calculated, which requires pre-filtering of images, calculation of their derivatives, point-by-point images multiplication and their smoothing, as outlined in formula (4). In the present implementation the maximum length of each of the filters is 11.

The matrix representation of the structure tensor obtained at this stage is not too useful, so we go on a vector representation (**5**). Now following parameters are given explicitly: the angle indicating the direction of maximum change in image (**6**), the tensor trace (7) and coherence level *c* (**9**) for each pixel of the image. We are looking for image points for which the direction of maximum change imposed is within the desired range of angles, provided that the coherence is greater than the set threshold value. Minimum value of coherence *c* is particularly important because it significantly affects the accuracy. Corner detection is to find points for which the minimum eigenvalue of the structural tensor exceeds a given threshold value (**8**). The result of these operations is an image in which each pixel contains two flags. The first one indicates the presence of a corner, while the other the presence of an edge having desired pitch. The output pixel format is illustrated in Figure 3. Image in this form is copied from the GPU global memory to host memory.

In this implementation we put special emphasis on efficient GPU memory management. Indeed, it appears that the time of allocation and release of GPU memory is long and comparable with the duration of structure tensor computation. One also cannot neglect the transfer time from the CPU memory to the GPU memory. The first of the dilemmas was resolved by transferring the responsibility for memory allocation for cudaStream class instance. This class was created in our framework for the purpose of effective memory and CUDA streams management. Once the memory was allocated by `cudaStream` it is destroyed by the same stream object. During the detection of corners and edges the memory transfer takes place only twice - when copying the original image to the GPU memory and copying back the result image to the CPU memory. All operations that happen during processing allocate GPU memory using a `cudaStream` class object. Number of GPU-CPU memory transfers has been limited to a necessary minimum. The Figure 4 shows a UML diagram of the classes needed to create the corners and edges detectors. To make any calculations using the GPU one has to create an object of the class `CUDAImageOperation.` Then decorate it with `CUDAImageOperationDecorator` and its derived classes. The last decorator must be `CUDAImageOperationFinalize` class instance. Since the operation of searching corners and edges is often needed, `CUDAUnaryGetEdgesAndCorners` class was defined which provides a convenient façade for above function classes. The following code listing shows exemplary usage of this class:

```
TImageOperation * operation = new
CUDAUnaryGetEdgesAndCorners
(ProcessedImage, OriginalImage,
AngleRanges, *stream);
(*operation)();  // perform detection
```

`CUDAImageOperation` class and the classes derived from `CUDAImageOperationDecorator` use elementary GPU functions - the CUDA kernels. These are the following functors: one-dimensional convolution, matrix point-by-point multiplication, the tensor representation transformation, search engine for edges and corners, etc. The method `operator()` of these classes calls them asynchronously, instructing CUDA driver to carry out them in the near future, but not waiting for them to complete. The time between the completion of execution of the kernel function and the end of their commission is a time where the CPU is in a 'sleep' state. The potential of unused CPU computational power is supposed to be utilized in further development.

4. EXPERIMENTAL RESULTS

Performance of edges and corers detection operation has been investigated on artificial and real images. Tests were performed on two machines: a laptop Core2Duo 1.8GHz, 2GB of RAM with the graphics card GeForce8400M GS and PC computer i7 3.0GHz endowed with the graphics card GeForce Quadro FX 3800M. Achieved performance differs significantly which results from different capabilities of the graphics cards – especially number of processing cores and their clock. Throughput was obtained by measuring the time of 16 times repeated detection operation. Time was measured by CUDA timers.

The plots in Figure 5 compare the throughput of detection operations. Performing calculations using the graphics card results in at least tenfold improvement in efficiency, comparing with bare CPU computations as well as with the OpenMP

parallelized version of HIL [4]. This gain increases slightly with a resolution of the input image, since operating on large images increases the profit resulting from the small CPU-GPU transfer to parallel operations (convolution, matrix multiplication) time ratio. Figure 6 is a diagram of operations performed on the graphics card while testing the edges and corners detector obtained using `computeprof v4.0`. The diagram includes a function performed on the graphics card as well as their duration. GPU work breaks during single detection are rare and short. This is a desired effect, because GPU time is not wasted. This is made possible through the use of asynchronous kernel functions and sparse synchronization of the CPU and GPU. The synchronization function is invoked only in necessary moments - when the CPU needs to use the results of calculations of the GPU.

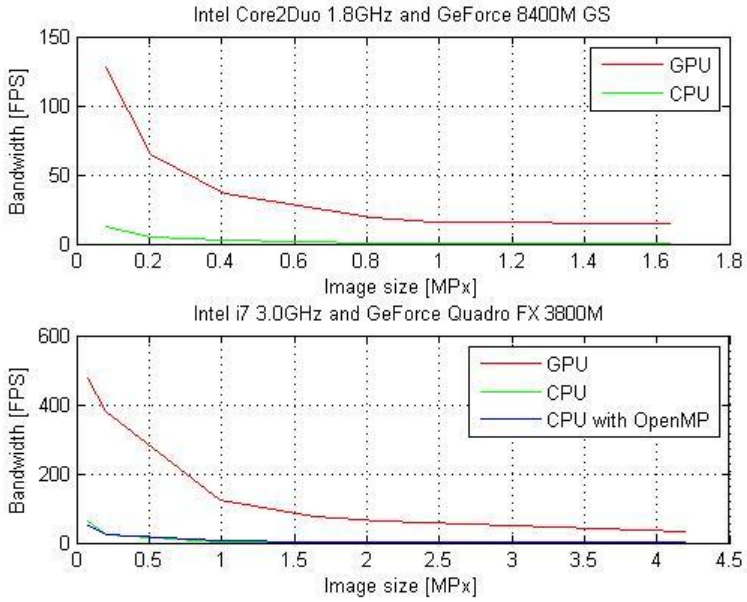

Figure 5 Achieved performance on different machines

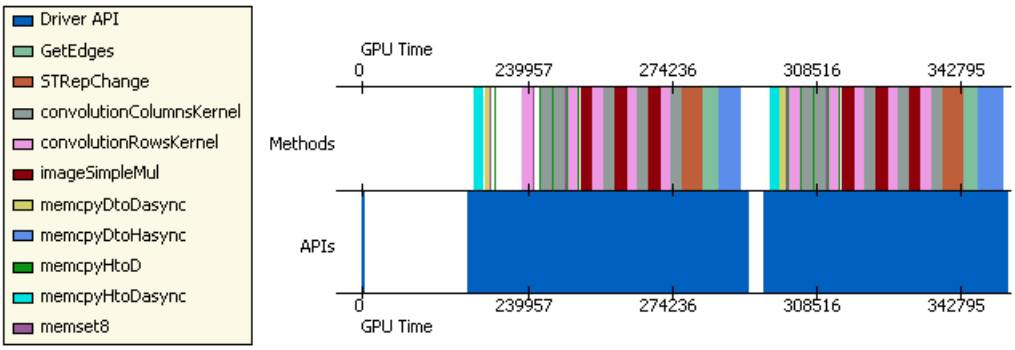

Figure 6 GPU load timing during edges and corner detection

It is worth noting that the performance of these operations should depend on length of the applied filters. In the above implementation always 11-tap filter is used. In the case of smaller length filters redundant coefficients are supplemented by zeros. Figure 7, Figure 8 and Figure 9 present detected corners and edges in computer generated test patterns as well as in a real image containing several road signs. The 3-tap filters (Simoncelli, Binomial) were used during operations. Filter details in [3].

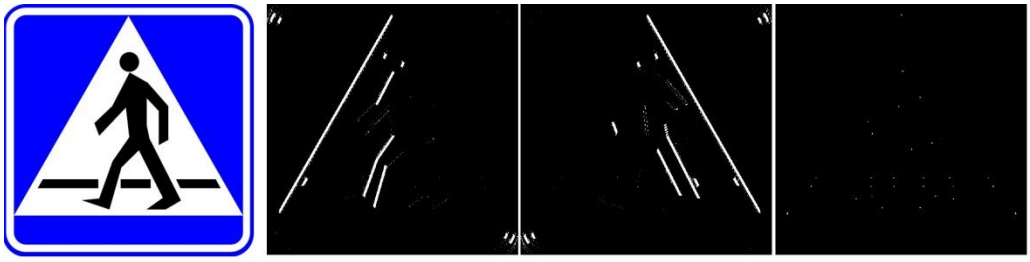

Figure 7. a) Original image b) angles in range (50, 70) c) angle in range (-50, -70) d) corners.

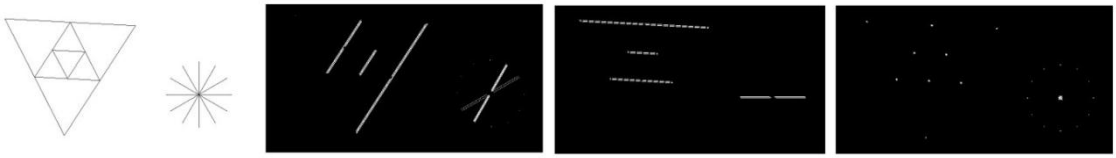

Figure 8. a) Original test image b) angles in range (50, 70) c) angles in range (0, -10) d) corners.

Figure 9. a) Real image b) angles in range (-50, -90) c) corners.

Figure 10. Pattern image and edges detection in full angle ranges

Detection in ranges (0, 20), (20, 40), (40, 80), (80, 10) in first row

Detection in ranges (-20, 0), (-40, -20), (-60, -80), (-100, -80) in second row

5. CONCLUSIONS

In this paper we present a method of detection of low-level features of specific orientation in images. These are corners and linear structures of predefined phase which allow detection of rigid objects, such as road signs, cars, etc. First the structural tensor is computed with help of the HIL library augmented with the graphic card with the CUDA environment. The presented method allows computations which in the worst case are at least an order-of-magnitude faster that an equivalent serial and multi-core software realization. This, in turn, allowed real-time operation on HDTV streams which is shown in the provided experimental results. Finally, the presented implementation was made available from the Internet and can be downloaded at [9].

6. ACKNOWLEDGMENTS

Financial support of the Polish funds for scientific research is greatly acknowledged.

7. REFERENCE

[1] Bigün, J., Granlund, G.H., Wiklund, J., Multidimensional Orientation Estimation with Applications to Texture Analysis and Optical Flow. IEEE PAMI **13**(8), (1991) 775-790

[2] Carson, C., Belonge, S., whichGreenspan, H., Malik, J., Blobworld: Image Segmentation Using Expectation-Maximization. IEEE PAMI **24**(8), (2002) 1026-1038

[3] Cyganek, B., Siebert J.P.: An Introduction to 3D Computer Vision Techniques and Algorithms, Wiley (2009)

[4] Cyganek, B.: Adding Parallelism to the Hybrid Image Processing Library in Multi-Threading and Multi-Core Systems. 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA 2011), Perth, Australia, 2011

[5] Di Zenzo S., A note on the gradient of a multi-image. Computer Vision, Graphics and Image Processing, **33**: (1986) 116-125

[6] Farid, H., Simoncelli, E.P., Differentiation of discrete multidimensional signals. IEEE Trans. Image Proc. **13**(4) (2004) 496-508

[7] Hauβecker, H., Jähne, B., A Tensor Approach for Local Structure Analysis in Multi-Dimensional Images. Technical Report, University of Heidelberg (1998)

[8] http://www.wiley.com/legacy/wileychi/cyganek3dcomputer/supp/HIL_Manual_01.pdf

[9] http://student.agh.edu.pl/~mbugaj/Detector/Detector.rar

[10] NVIDIA, CUDA C Programming Guide (2011)

[11] Sanders, J., Kandrot, E., CUDA by example: an introduction to general-purpose GPU Programming, Addison-Wesley (2011)

[12] NVIDIA, CUDA Toolkit Reference Manual, (2011)