

Towards real-time SVD based motion detection on GPU

Ádám Botos¹, Dmitry Chetverikov², and Péter Kovács¹

¹ Eötvös L. University, Pázmány Péter stny. 1/C, Budapest, Hungary,
fokosalja@gmail.com, kovika@inf.elte.hu

² Institute for Computer Science and Control, Kende út 13-17, Budapest, Hungary,
csetverikov@sztaki.hu,

Abstract. The Singular Value Decomposition (SVD) can be efficiently used to detect motion in videos captured by a static camera. However, the SVD is computationally demanding when a large matrix - a spatio-temporal data window typically composed of ten to thirty frames - is repeatedly processed. Recently, a running (incremental) version [1] of the SVD has been proposed and applied to motion detection. Although much faster than the direct implementation, the CPU-based running SVD is still too slow for real-time processing of VGA or larger size videos. In this paper, we present a GPU implementation of the running SVD that is suitable for robust, close to real-time motion detection in challenging full-size, standard frame rate videos. The computational performances of different hardware configurations are compared showing a significant gain in processing speed due to the proposed solution.

Keywords: Video processing, motion detection, SVD, GPU.

1 Introduction

In this research, we address the problem of real-time motion, or target, detection in full-size, standard frame rate videos acquired by static cameras in indoor or outdoor environments. Numerous approaches to motion detection have been proposed based on different principles. The reader is referred to study [2] for a survey and comparison of methods for background modelling and target detection. Note that the problem addressed in this paper differs from that of change detection [3] when multiple images of a static scene are acquired at different times and compared to find changes. The well-known approach to activity recognition [4] is based on adaptive background subtraction and motion segmentation by modelling the intensity in each pixel as a mixture of Gaussian distributions. However, such pixel-wise methods cannot capture the natural dependencies between neighbouring pixels within regions. In particular, they are sensitive to background variations typical for both indoor and outdoor data (illumination, visibility, dynamic background). The methods that operate with regions rather than individual pixels are usually more robust as they are able to incorporate and exploit local structural information. In particular, the Principal Component

Analysis (PCA) and the Singular Value Decomposition have been applied in the context of moving target detection. Due to its generality and robustness, the SVD proved to be more successful than the PCA as far as motion detection is concerned. It has been demonstrated that the SVD based algorithms [1, 5, 6] can cope with background variations even in the cases of complex backgrounds such as water or vegetation. A critical problem with the SVD is that the amount of computation increases drastically with the video resolution and the number of frames in the spatio-temporal window used to estimate the background. Some studies [5, 6] tried to decrease the computational load of the SVD by reducing resolution, splitting frame images into rectangular subimages, or using approximate decompositions. However, splitting the frame images leads to artifacts at the subimage borders while the approximate decompositions result in error accumulation and the need to reinitiate the detection process from time to time. Recently, a computationally efficient running version of the SVD [1] has been proposed and successfully used for motion detection. As the video proceeds, the incremental approximation-free algorithm [1] does not calculate the SVD from scratch when the data window steps in time. Instead, the current decomposition is updated with the entering frame, then downdated with the leaving one. This solution is much faster than the direct implementation of the SVD. On a CPU, it allows to process a few frames per second at a reduced video size and for up to 15 frames in the sliding spatio-temporal window. However, the CPU based running SVD is still too slow for real-time processing of VGA or larger size videos. In this study, we propose a GPU implementation and test the parallelised running SVD whose speed approaches the level required for real-time applications with full-size video streams. The structure of the paper is as follows. First, we briefly discuss the mathematical background of the SVD and its incremental variant [1] in the context of motion detection. Then the GPU implementation of the running algorithm is described and comparative test results are shown and discussed. Finally, the paper concludes with a summary of the current state of the research and an outlook.

2 Running SVD for motion detection

The Singular Value Decomposition is defined as follows. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a real-valued matrix. Then there always exists a decomposition $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, where the orthogonal matrix $\mathbf{U} \in \mathbb{R}^{m \times m}$ consists of m orthonormal eigenvectors of $\mathbf{A}\mathbf{A}^T$; the orthogonal matrix $\mathbf{V} \in \mathbb{R}^{n \times n}$ consists of n orthonormal eigenvectors of $\mathbf{A}^T\mathbf{A}$; $\mathbf{S} \in \mathbb{R}^{m \times n}$ is composed of a diagonal matrix \mathbf{D} and a null matrix. The matrix $\mathbf{D} = \text{diag}(\delta_1, \dots, \delta_m)$ contains the singular values of \mathbf{A} which are the square roots of the eigenvalues of $\mathbf{A}\mathbf{A}^T$. It is usually assumed that $\delta_1 \geq \delta_2 \geq \dots \geq \delta_m \geq 0$. A number of methods exist that calculate the SVD. Following the study [1], we use the well-known Golub–Reinsch SVD algorithm [7]. The algorithm decomposes an $m \times n$ size matrix, where m is the number of frames used to estimate the background (temporal depth of the data window), n the total number of pixels in a frame image. Typically, $5 \leq m \leq 30$, while in our

tests the video size was 768×576 (PAL), that is, $m \times n$ and $n > 400000$. For such a large matrix, applying the SVD to each data window in a long video would be prohibitive even if implemented on a GPU. Fortunately, after the single initialisation for the first m frames, the running procedure [1] can be used at each temporal step, which applies a fast update algorithm, then a fast downdate algorithm. The former updates the three matrices of the SVD by adding the entering frame to the current m frames, the latter removes the exiting frame from the updated SVD. Both algorithms modify the sizes of the SVD matrices. The Update increases the dimension of each matrix by one, the Downdate restores the original size. Between the two algorithms, the procedure executes a shift in one of the matrices. The time complexity of the running SVD is $\mathcal{O}((m+n)m^2) \approx \mathcal{O}(nm^2)$. The linear dependence on the frame size is important as it allows to work with full-size video. Note that the time complexity of the Golub–Reinsch SVD [7] is also $\mathcal{O}(nm^2)$. However, the same asymptotic behaviour does not mean that in practice the execution times of the two algorithms are the same. As demonstrated in [1], the incremental solution is much faster for typical values of the two parameters. Our tests discussed later in this paper also confirm that the difference is essential. Contrary to the approaches [5, 6] that apply the conventional SVD, the incremental solution does not need splitting frames into blocks to achieve acceptable processing time. Both steps of the running SVD are approximation-free, so no error accumulation occurs and no reinitialisation is needed. Before applying the SVD, the image data must be normalised by subtracting the mean and mapping the values onto a standard range. For each spatio-temporal window, the background is estimated using the largest singular value and the associated vectors. (If necessary, more singular values can be used [6].) Then the residual is calculated as the absolute difference between the input data and the background. Large residual indicates the areas in motion. After thresholding, the binarised residual image is cleaned using morphological operations. When fast noisy motion like snow or rain is present, it may also be useful to apply a spatio-temporal median filter to several consecutive thresholded frames. In order to visualise the result of motion detection, the borders of the blobs in the thresholded and filtered residual are overlaid on the central frame of the data window. Sample results of the running SVD are shown in Figure 1. The algorithm features robustness to noise, visibility conditions and varying background. More results are available at the web site³ of motion detection by the running SVD.

3 Hardware implementation of SVD based motion detection

We have implemented, tested and compared three procedures for SVD based motion detection:

1. the direct Golub–Reinsch algorithm that processes each data window separately;

³ athos.vision.sztaki.hu/~mitya/research/runsvd/results.html



Fig. 1. Examples of motion detection and segmentation by the running SVD.

Table 1. Three hardware configurations used in the tests.

Hardware	Configurations		
	Conf. 1	Conf. 2	Conf. 3
CPU	Intel Pentium Dual Core 2.16 GHz	Intel Core i3 2310M 2.1 GHz	Intel Core i5 4200U 1.6 GHz
GPU	NVIDIA GeForce 9500M 510 MB	NVIDIA GeForce GT 520M 1 GB	NVIDIA GeForce GT 730M 2GB
Memory	DDR II - 4 GB	DDR III - 4 GB	DDR III - 8 GB
Comp. Cap.	1.0	2.0	3.0
Produced	2009	2012	2014

2. the sequential running SVD;
3. the parallel running SVD.

Each of the three procedures was executed on three different hardware configurations whose main features are summarised in Table 1. The abbreviation Comp. Cap. stands for Compute Capability. Its values indicate that more recent configurations are more powerful.

The image processing functions of the procedures were implemented using the Open Source Computer Vision (OpenCV) software package [8] that offers GPU support. The package contains motion detection functions, as well. However, they are not suitable for real-time video processing. The package also includes certain matrix operations, but they are not really applicable to very large matrices, and their scope is limited. The parallel data processing algorithms were implemented with the Computer Unified Device Architecture (CUDA) developed by the NVIDIA Corp. The data are stored in the memory of the graphics card, so image data transfer from the CPU memory to the GPU memory should be optimised for faster operation. The transfer is complicated by the different formats of matrix storage in OpenCV and CUDA, which necessitates additional data movement. In this project, two external program libraries were used with

CUDA: The CUDA Basic Linear Algebra Subroutines (cuBLAS) [9] and the CUDA Linear Algebra (CULA) [10]. The former provides CUDA implementation of basic matrix operations, while the latter includes more sophisticated algorithms such as the SVD we need for our purposes.

4 Test results

For testing, we used the 768×576 resolution video data from the Image Sequence Server [11]. First, the direct implementation of the Golub–Reinsch SVD (procedure 1) was tested on the three hardware configurations defined in Table 1. This solution is definitely not suitable for real-time processing since even on the most powerful configuration (No. 3) the total execution time for a short 50-frame sequence was over 200 seconds. For more realistic, longer sequences the execution time was too long even for testing. For this reason, only the two versions of the running SVD, the CPU and the GPU, were tested and compared in detail. Figure 2 demonstrates the results of the sequential CPU implementation. The plot shows the execution times, in seconds, for the growing number of processed frames on the three hardware configurations. In this case, the number of processed frames ranged from 5 to 50. The best result of about 1.3 fps was achieved on the strongest configuration for the longest sequence. For longer sequences, the processing speed is the same.

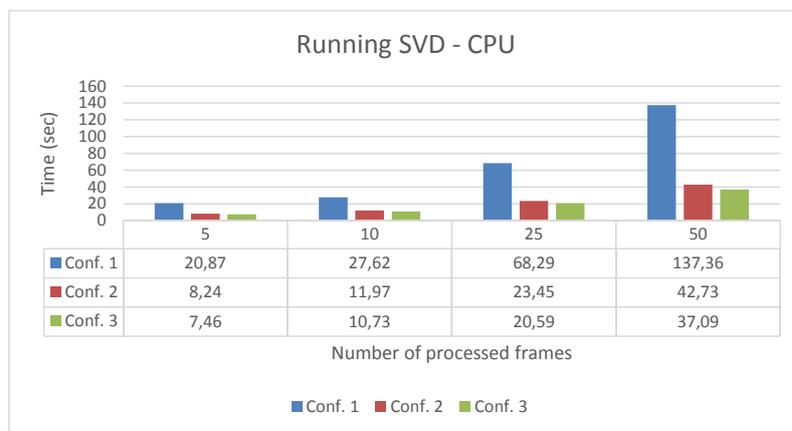


Fig. 2. Plot of execution times on CPU for the three hardware configurations.

Figure 3 shows the results of the parallel GPU implementation. As this solution is faster, in this case, the number of processed frames ranged from 100 to 600. The best result of about 8.0 fps was achieved on the strongest configuration for the sequence of 250 frames. The processing speed seems to be slightly

content-dependent, but the dependence on content and video length is not significant. Due to the GPU implementation, the processing speed was increased by a factor of six.

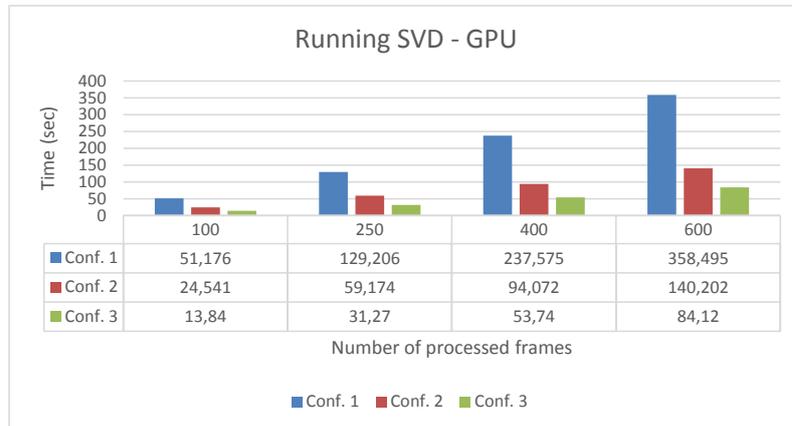


Fig. 3. Plot of execution times on GPU for the three hardware configurations.

5 Conclusion

The 8 fps processing rate provided by our GPU solution for full-size videos can already be sufficient for the applications where the motion is relatively slow and the standard full frame rate of 25 – 30 fps is not required. Our further research and development nevertheless aims at achieving the full frame rate in near future. Naturally, in this development we hope to make use of the persistent improvement of the processing power of modern GPUs. As one can see in Figure 3, configuration 3 is more than four times faster than configuration 1, which well illustrates the progress made within the four years. Still, there is room for improving our software solution, as well. The success of a GPU implementation is mainly influenced by two critical issues. The first one is the selection of the appropriate program package. Since our task requires a large amount of computation, we have selected the CUDA programming environment supported by NVIDIA video cards. An essential advantage of this environment is that it supports numerous other packages including matrix operations and linear algebra. When CUDA is used for parallel implementation on a GPU, the data copied to the GPU memory cannot be accessed directly by the CPU. Data transfer is needed which increases the execution time. The CUDA Thrust library [12] can be used to access the GPU memory in a more convenient way. Using this library would need a major programming effort as the Thrust has its own data structure for supporting parallel implementations. The second critical issue is that

of the code simplicity. In our development, we have always tried to implement each program in the simplest possible way. When possible, we have used already created variables and data structures in order to avoid unnecessary allocations and data movement.

Acknowledgement. This research has been supported by the Highly industrialised region on the west part of Hungary with limited R&D capacity: Research and development programs related to strengthening the strategic future-oriented industries manufacturing technologies and products of regional competences carried out in comprehensive collaboration program of the National Research, Development and Innovation Fund (NKFI), Hungary, Grant. No. VKSZ_12-1-2013-0038.

References

1. D. Chetverikov, A. Axt, Approximation-free running SVD and its application to motion detection, *Pattern Recognition Letters*, vol. 31, no. 9, pp. 891–897, 2010.
2. D. Hull, J. Nascimento, Comparison of target detection algorithms using adaptive background models, in *Proceedings of the 2nd Joint IEEE Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, 2005, pp. 117–128.
3. R. Radke, S. Andra, O. Al-Kofahi, B. Roysam, Image change detection algorithms: a systematic survey, *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp. 294–307, 2005.
4. C. Stauffer, W. Grimson, Learning patterns of activity using real-time tracking, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747–757, 2000.
5. A. Monnet, A. Mittal, N. Paragios, V. Ramesh, Background modeling and subtraction of dynamic scenes, in *Proceedings of the 9th IEEE International Conference on Computer Vision*, 2003, pp. 1305–1312.
6. F. Kahl, R. Hartley, V. Hilsenstein, Novelty detection in image sequence with dynamic background, in *Proceedings of the 2nd Workshop on Statistical Methods in Video Processing (SMVP)*, European Conference on Computer Vision, 2004, pp. 117–128.
7. G. H. Golub, C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, Maryland, USA, 4th edition, 2013.
8. Intel Russia, OpenCV 2.4.11.0 Documentation, Available: <http://docs.opencv.org/>, 2015, [Online].
9. NVIDIA Corporation, USA, cuBLAS Library v7.0 Documentation, Available: <http://docs.nvidia.com/cuda/cublas>, 2015, [Online].
10. EM Photonics Inc., USA, CULA Reference Manual R17, Available: <http://culatools.com>, 2014, [Online].
11. University of Karlsruhe, Institute of Algorithms and Cognitive Systems, Image Sequence Server, Available: http://i21www.ira.uka.de/image_sequences, 1998, [Online].
12. J. Hoberock, N. Bell, Thrust. A parallel template library 1.8.0., Available: <http://thrust.github.io>, 2015, [Online].