Acceleration of Profile Creation for Three-Dimensional Vector Video with GPGPU

A. Tsyganov <tsyganov.aa@samgtu.ru>,
Samara State Technical University
244 Molodogvardeyskaya Str., Samara, 443100, Russian Federation

Abstract. In the report the optimization of image similarity metric computation method for three dimensional vector video with general-purpose computations on graphical processor unit (GPGPU) is discussed. The use of stream processors in graphics accelerators and Compute Unified Device Architecture (CUDA) platform allows significant performance gain in comparison to calculations on general-purpose processors, while solving problems of computer vision and image similarity determination. The performance of the GPGPU metric value computation is measured and researched.

Keywords: three-dimensional video, graphical processor unit, computer vision, metrics, key points.

DOI: 10.15514/ISPRAS-2015-27(3)-27

For citation: Tsyganov A. Acceleration of Profile Creation for Three-Dimensional Vector Video with GPGPU. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 379-388. DOI: 10.15514/ISPRAS-2015-27(3)-27.

1. Introduction

Video playback systems for three-dimensional vector format need to determine parameter types of shader programs contained in the video stream. This can be accomplished by creating profiles for each video source type. Profiling is resource-intensive task and the calculations cannot be performed in real time while running the application for which the profile is compiled. The longest stage of the method is the metric calculation. The paper proposes to move its computation to graphics processing unit (GPU) in order to speed up the algorithm.

2. Profiling method

Method for automated profiling based on a comparison of images obtained with the original shader parameters and ones found after applying shaders with modified parameters.

For each shader it is necessary to find the correct types of values transmitted to its parameters. For implementation of stereoscopic effects, parameters containing projection matrixes are important. Thereby the problem is reduced to search such matrixes among parameters of the shader program. Parameters type search in the method is carried out by their search for each separate parameter. The assumption of correctness for the selected type is checked by similarity evaluation of images received from frame visualization of a video stream without modification of parameters and with modification of parameters according to the assumption made.

The selected frame *V* of the initial video stream is modified by transform *T(V, S)* which changes set of shader parameters *S* concerning of which assumption was made about their certain type. The initial frame of *V* and the modified frame *V'* are rasterized by *R(V)* resulting in two images *I* and *I'* respectively. This images are represented by function of brightness in the given point $I = f_I(x, y)$. They are compared by using a metric. The result of applying this metric is the set *D*, consisting of two integral values, which are passed to the decision *A(D)*:

$$D = \{DB, DS\}, \tag{1}$$

$$A(D) = \begin{cases} 1, & D_B \le b_m \cap D_S \ge s_m \\ 0, & D_B > b_m \cup D_S < s_m \end{cases}, \tag{2}$$

where b_m and S_m are the boundary values of the metric components.

Metrics computing algorithm for two images processes raw data in a few steps. Under the original data we will assume two images obtained with initial visualization parameters I_o and with modified visualization parameters I_m. Two color histograms $H(I_o)$ and $H(I_m)$ are calculated from the original image by dispersion method. Initial evaluation of the distance between the images performed by using Bhattacharya distance $D_B(H_o, H_m)$. Second component of metric is specified by comparing sets of control points in the original image. Sets of control points P_o and P_m, received from the image I_m and I_o, respectively, are used to calculate the distance $D_S(P_o, P_m)$. Speeded Up Robust Features (SURF) method is used for point detection, the implementation of which is also available for GPU [1, 2].

3. GPGPU implementation

The architecture of modern graphics cards is designed for vector operations with the data in the form of multi-dimensional arrays. This allows to achieve high memory speed when using SIMD vector processors with independent L1 and L2 caches. In comparison to a general purpose processor, GPU has fewer steps and a smaller amount of the conveyors cache. Exchange of data between video memory and general purpose memory is implemented via the PCI-E x16 bus. The sample data in the cache transfers through a 256-bit bus. As a result, the efficiency of scientific algorithms on the GPU depends on the efficient use of memory and cache [3].

The main purpose of the GPU method implementation is to minimize the number of data exchanges between video memory and general-purpose memory. Communication between the CPU and graphics core negatively affect performance. To reduce the data used by the various stages of the algorithm, it loaded into video memory only once. The result is also available in video memory for the following stages. The essence of the developed method is the efficient use of the cache and loading video streaming GPU cores uniformly. Transfer of resources between the stages of the algorithm is carried out through the video memory, as shown in Fig. 1, which speeds up processing using the GPU.

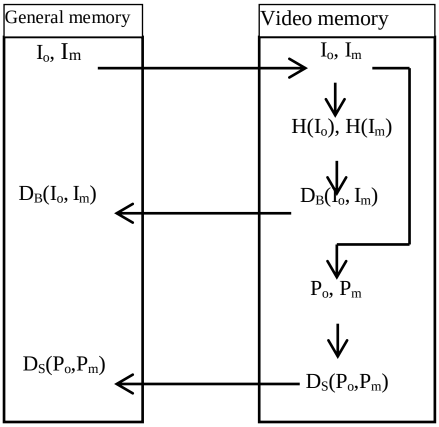

Fig. 1. Data exchange between general-purpose memory and video memory.

Images I_m and I_o are loaded into video memory for processing. On their basis histograms are calculated to find the first components of the metric using Bhattacharya distance. The same source images used by SURF algorithm to calculate set of points, which are used as the basis for the second component of the metric. Only calculated components of the metric unloaded from video memory to general-purpose memory. Their size is extremely small, and video memory reading will not stop the process of computing on the GPU, resulting in high performance parallel computing.

4. Histogram computation

Calculation of the metric component D_B is performed by using the histograms $H(I_O)$ and $H(I_S)$ of corresponding images. The calculation of the histogram on the GPU can be performed using both classical shader programs, and using CUDA technology for general-purpose computation on the GPU. CUDA technology usage described in the works of Podlozhnyuk [4] and Shams [5]. These algorithms provide better performance than those based on the use of conventional means of graphical programming interfaces, as shown by Nugteren et al. [6]. Work of Fluck [7] is an example of the second approach.

Since the main objective is to accelerate the metrics calculation then most appropriate methods for histogram computation are based on CUDA. Such as method of Podlozhnyuk, that implemented in CUDA SDK. Method is cache effective and does not contain steps of data upload into shared memory that allows it to be integrated into the process of metric component calculation.

In this method, the original data is divided into blocks between threads executed on the GPU. Output data stream is stored in individual histogram. In the final pass all histogram are combined by different threads into one. To efficiently use shared memory of streams each individual histogram is created in group of threads called rope. This allows to store histograms of a larger volume, up to 6 kilobytes on G80 hardware architecture.

Bhattacharya distance calculation based on the histogram for two sets of statistics. It is expressed by the following formula:

$$D_B = \sum_{i=0}^{n} \sqrt{H(I_o)_i H(I_m)_i}\ , \tag{3}$$

where *n* - the number of the histogram elements.

Calculation of histogram elements sums can be done by reduction of the initial data array on the GPU. It is proposed to use an optimized method of parallel reduction on CUDA, described by Mahardito et al. [8].

5. Key points detection

The second component *DS* of the metric calculated with SURF algorithm [9]. With its help search is performed for two sets of points *P* and *P’*, available in the original and the modified frames, respectively. The value of component determined by the following expression:

$$D_S = \frac{|P \cup P'|}{|P|}\,. \tag{4}$$

SURF is one of the most common and efficient image points search algorithms. It used in automatic object recognition and tracking, video recording, panoramic image combining and in many other areas of computer vision. The algorithm can process images in HD resolution at more than 30 frames per second.

SURF detects points by approximating the Hessian. Approximation performed by application of block filters to the image. It makes good use of the integral representation of the image *II*, which is determined by the following formula:

$$II(x,y)=\sum_{i=0,\,j=0}^{i\le x,\,j\le y} I(i,j). \tag{5}$$

The calculation of the integral image representation on the GPU is the longest stage of the SURF algorithm and can be implemented by the algorithm of the pyramid points as described in Terriberry et al. [10]

Construction of the integral image is the task of the prefix sum. Pyramid algorithm offers a solution to this problem on the GPU in two stages. At the first stage, pyramid images constructed extending upward, each of which divides into four parts half the width and height than the previous level. Image content is determined by three components of *U(k)*, *H(k)*, *V(k)*:

$$\begin{aligned} U^{(k)}(x,y) &= U^{(k-1)}(2x,2y) \\ &+U^{(k-1)}(2x+1,2y) \\ &+U^{(k-1)}(2x,2y+1) \\ &+U^{(k-1)}(2x+1,2y+1), \end{aligned} \tag{6}$$

$$\begin{aligned} H^{(k)}(x,y) &= U^{(k-1)}(2x,2y) \\ &+U^{(k-1)}(2x+1,2y), \end{aligned} \tag{7}$$

$$V^{(k)}(x,y)=U^{(k-1)}(2x,2y)+U^{(k-1)}(2x,2y+1), \tag{8}$$

where *k* - level of the pyramid, *x* and *y* - coordinates of the image.

It requires two half-sum of *H (k)* and *V (k)* to calculate the sum of the even rows and columns, using formula:

$$X^{(k)}(x,y)=\sum_{i=0}^{x-1} H^{(k)}(i,y), \tag{9}$$

$$Y^{(k)}(x,y)=\sum_{j=0}^{y-1} V^{(k)}(x,j). \tag{10}$$

Using the obtained image pyramid, a reverse pass going from the top downwards. This value is used to calculate four different versions of the formula that depend on the parity argument. For even *x* and *y*

$$W^{(k)}(x,y)=W^{k+1}(\left\lfloor \frac{x}{2} \right\rfloor,\left\lfloor \frac{y}{2} \right\rfloor), \tag{11}$$

for odd *x* and even *y*

$$W^{(k)}(x,y)=W^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right]) \\ +Y^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right]), \tag{12}$$

for even x and odd y

$$W^{(k)}(x,y)=W^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right])+X^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right]), \tag{13}$$

for odd *x* and *y*

$$W^{(k)}(x,y)=W^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right]) \\ +X^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right])+Y^{k+1}(\left[\frac{x}{2}\right],\left[\frac{y}{2}\right]) \\ +U^{(k-1)}(x-1,y-1). \tag{14}$$

The values of the top-level assumed to be zero.

Using the integral image, the key points are determined by searching the extremum of the Hessian determinant. Block filters used for this purpose as described by Bay et al. [9] Their GPU computation requires only 17 texture samples per pixel. Search for a local Hessian maximum can be made by the method of neighboring points 3x3x3.

Each found key point is described by the descriptor, which is a normalized vector calculated using filters similar to the Haar block filter for Hessian. Sets of elements *P* and *P'* are compared using descriptors, which calculates the value of D_S with expression (4).

6. Performance evaluation

An experimental study with various sources of graphic information was carried to determine the performance gain of GPGPU implementation in comparison with the general-purpose processor implementation. Sources of graphical information were selected by statistics of streaming video services.

The first series of experiments aimed at assessing the dependence of the duration profiling on the recording. The results are shown in Fig. 2. As can be seen, the work time increases insignificantly, since longer records contains almost no new shader programs. However, there is a significant reduction in execution time by 8-12 times when using a GPU implementation.

Composition of the shader programs in each application is heterogeneous. The main feature affecting the complexity of the specific shader program analysis is the number of its parameters of interest for the algorithm. To evaluate the impact of this amount on processing time for each shader program, a series of experiments was carried with same sources of image information, as in the previous case.

The values are averaged over all shader programs with a given number of parameters of matrix type for a ten minute record. The results are shown in Fig. 3.

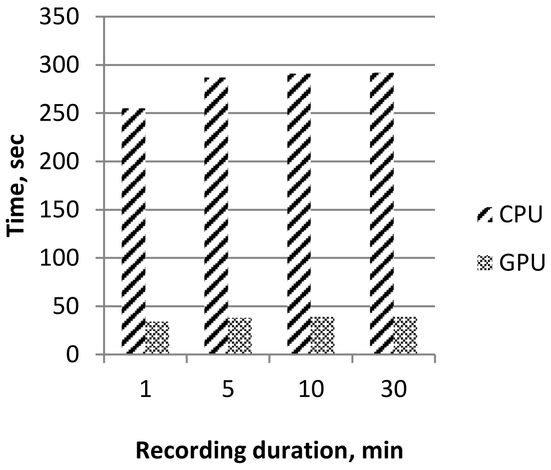

Fig. 2. The diagram of time depending on the duration of the record.

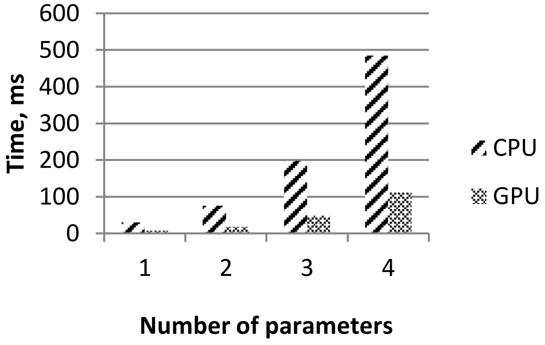

Fig. 3. Diagram of time depending on the number of parameters.

Number of recognizable parameters affects their recognition duration exponentially. Speed of data processing strongly depends on the complexity of video source rendering system. However, GPGPU calculations can reduce it by 8-12 times. This allows comparison of vector video frames and subsequent profiling on the terms that are acceptable to use these methods in practice.

References

[1]. Thorsten Scheuermann, Justin Hensley. Efficient histogram Generation Using Scattering on GPUs. *Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM New York, NY, USA*, 2007, pp. 33-37. doi: 10.1145/1230100.1230105

[2]. N. Cornelis, L. Van Gool. Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1-8. doi: 10.1109/CVPRW.2008.4563087

[3]. N. K. Govindaraju, E. S. Larsen, J. Gray, D. Manocha. A memory model for scientific algorithms on graphics processors. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06), NY, USA: ACM Press,* 2006, no. 89, pp. 6-15. doi: 10.1109/SC.2006.2

[4]. V. Podlozhnyuk. Histogram calculation in CUDA. Technical report. *NVIDIA*, 2007, http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf

[5]. Ramtin Shams, R. A. Kennedy. Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices. *Australia, Gold Coast, ICSPCS*, 2007. pp. 418-422.

[6]. Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, Bart Mesman. High Performance Predictable Histogramming on GPUs: Exploring and Evaluating Algorithm Trade-offs. *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, NY, USA: ACM New York*, 2011. pp. 1-9. doi: 10.1145/1964179.1964181

[7]. O. Fluck, S. Aharon, D. Cremers, M. Rousson. GPU histogram computation. *ACM SIGGRAPH 2006 Research posters, SIGGRAPH '06. ACM*, 2006, p. 53. doi: 10.1145/1179622.1179683

[8]. Adityo Mahardito, Adang Suhendra, Deni Tri Hasta. Optimizing Parallel Reduction in Cuda to Reach GPU Peak Performance. *Proceedings of The Second International Workshop on Open source and Open Content WOSOC 2010, Indonesia, Depok.: Gunadarma University*, 2010, pp. 48-57.

[9]. Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding, New York, USA*, 2008, vol. 110, no. 3, pp. 346-359. doi: 10.1016/j.cviu.2007.09.014

[10]. Timothy B. Terriberry, Lindley M. French, John Helmsen. GPU Accelerating Speeded-Up Robust Features. *Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization and Transmission, Georgia Institute of Technology, Atlanta, GA, USA*, 2008. pp. 355-362.

Ускорение создания профилей для трехмерного векторного видео с помощью GPGPU

А. А Цыганов <tsyganov.aa@samgtu.ru>,
Самарский Государственный Технический Университет,
443100, Россия, г. Самара, ул. Молодогвардейская, дом 244

Аннотация. В работе рассматривается метод оптимизации вычисления метрики схожести изображений с помощью вычислений общего назначения на графическом процессоре (GPGPU). Использование потоковых процессоров графических ускорителей и платформы CUDA позволяет добиться значительного прироста производительности по сравнению с расчетами на процессорах общего назначения при решении задач в области компьютерного зрения, в частности для определения схожести изображений. Приведены результаты исследования производительности GPGPU реализации расчетов значений метрики.

Ключевые слова: трехмерное видео, графический процессор, компьютерное зрение, метрика, ключевые точки.

DOI: 10.15514/ISPRAS-2015-27(3)-27

Для цитирования: Цыганов А.А. Ускорение создания профилей для трехмерного векторного видео с помощью GPGPU. Труды ИСП РАН, том 27, вып. 3, 2015 г., стр. 379-388 (на английском языке). DOI: 10.15514/ISPRAS-2015-27(3)-27.

Список литературы

[1]. Thorsten Scheuermann, Justin Hensley. Efficient histogram Generation Using Scattering on GPUs. *Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM New York, NY, USA*, 2007, pp. 33-37. doi: 10.1145/1230100.1230105

[2]. N. Cornelis, L. Van Gool. Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1-8. doi: 10.1109/CVPRW.2008.4563087

[3]. N. K. Govindaraju, E. S. Larsen, J. Gray, D. Manocha. A memory model for scientific algorithms on graphics processors. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06), NY, USA: ACM Press,* 2006, no. 89, pp. 6-15. doi: 10.1109/SC.2006.2

[4]. V. Podlozhnyuk. Histogram calculation in CUDA. Technical report. *NVIDIA*, 2007, http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf

[5]. Ramtin Shams, R. A. Kennedy. Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices. *Australia, Gold Coast, ICSPCS,* 2007. pp. 418-422.

[6]. Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, Bart Mesman. High Performance Predictable Histogramming on GPUs: Exploring and Evaluating Algorithm Trade-offs. *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, NY, USA: ACM New York*, 2011. pp. 1-9. doi: 10.1145/1964179.1964181

[7]. O. Fluck, S. Aharon, D. Cremers, M. Rousson. GPU histogram computation. *ACM SIGGRAPH 2006 Research posters, SIGGRAPH '06. ACM*, 2006, p. 53. doi: 10.1145/1179622.1179683

[8]. Adityo Mahardito, Adang Suhendra, Deni Tri Hasta. Optimizing Parallel Reduction in Cuda to Reach GPU Peak Performance. *Proceedings of The Second International Workshop on Open source and Open Content WOSOC 2010, Indonesia, Depok.: Gunadarma University*, 2010, pp. 48-57.

[9]. Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding, New York, USA*, 2008, vol. 110, no. 3, pp. 346-359. doi: 10.1016/j.cviu.2007.09.014

[10]. Timothy B. Terriberry, Lindley M. French, John Helmsen. GPU Accelerating Speeded-Up Robust Features. *Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization and Transmission, Georgia Institute of Technology, Atlanta, GA, USA*, 2008. pp. 355-362.