Przemysław MAZUREK
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE, KATEDRA PRZETWARZANIA SYGNAŁÓW I INŻYNIERII MULTIMEDIALNEJ
26. Kwietnia 10, 71-126, Szczecin

Circle parameters estimation using Hough transform implemented on GPGPU

Dr eng. Przemysław MAZUREK

Assistant professor in the Department of Signal Processing and Multimedia Engineering at the Faculty of Electrical Engineering, West-Pomeranian University of Technology, Szczecin. Author of more then 100 papers related to the digital signal processing, estimation of object kinematics, biosignals acquisition and processing.

e-mail: przemyslaw.mazurek@zut.edu.pl

Abstract

In the paper implementation of the Hough transform using NVidia CUDA platform is considered. The GPGPU implementation is based on processing in parallel a set of Hough transforms with synchronized memory accesses for better utilization of the texture cache. The best code variant is based on quadrature sine and cosine functions, an unrolled loop and a single write to the global memory. The processing time is about 1000 shorter in comparison to the Matlab code, which is necessary for processing the video data.

Keywords: estimation, pattern recognition, Hough transform, GPGPU.

Estymacja parametrów okręgu z wykorzystaniem transformaty Hougha dla GPGPU

Streszczenie

W artykule rozpatruje się implementację transformaty Hougha [1] dla okręgów (1). Celem implementacji jest skrócenia czasu przetwarzania wielu obrazów o dużej rozdzielczości na potrzeby estymacji położenia i promienia półsferycznego próbnika oświetlenia stosowanego [3] podczas pomiarów światła na potrzeby realistycznej grafiki i animacji komputerowej (rys. 1). Kolorowy obraz przetwarzany jest za pomocą algorytmu [2] (rys. 3), a w celu redukcji czasu przetwarzania skoncentrowano się na wykorzystaniu platformy NVidia CUDA 3.2 [5, 6] do równoległej realizacji transformaty Hougha [7-12]. Wykorzystano oryginalną konfigurację bloków wątków oraz siatki w celu efektywnego wykorzystania pamięci podręcznej tekstur przy równoległym próbkowaniu obrazu. W implementacji 32 wątki bloku wykonują transformatę Hougha pobierając wartości obrazu z pierścienia w sposób synchroniczny w celu optymalizacji wykorzystania pamięci tekstur. Porównano 14 metod wyznaczania (tab. 1) próbkowanego piksela. Porównano metodę zapisu wyniku najlepszego dla bloku z wykorzystaniem jednego i wszystkich wątków. Najbardziej efektywnym rozwiązaniem jest wykorzystanie funkcji kwadraturowej wraz z rozwijaniem pętli i pojedynczym zapisem. Dla procesora G80 (Geforce 8800 GTS) uzyskano 1000-krotne przyspieszenie obliczeń w stosunku do kodu w Matlabie wykonywanego na procesorze Pentium 4 (2.4GHz). Dla 32 sąsiednich promieni i 100 próbek każdego okręgu czas przetwarzania jest rzędu 1 sekundy.

Słowa kluczowe: Estymacja, Rozpoznawanie obrazów, Transformata Hougha, GPGPU.

1. Introduction

Estimation of low-level geometrical objects like points, lines, and circles is very important for machine vision applications. Proper detection and estimation of object parameters is useful for mid-level and high-level scene understanding. Fast and reliable low-level algorithms reduce complexity and computation costs of higher level processing.

Estimation of the position and orientation of lines located in an image is possible with use of the Hough transform. This transform accumulates pixel values over all possible line orientations [1].

Such accumulation approach is optimal and computationally demanding. Similar accumulation technique is applied to the other shapes so any curve (closed or not) can be also processed. One of the most important curves is the circle and estimation of the position and radius of multiple circles is necessary in numerous applications.

The estimation of light probe parameters is important for light measurement and considered in [2]. The light probe is located in a selected point of the 3D space and the reflected light is obtained using the light probe [3]. Hemispherical or spherical mirrors are used for such purposes.

Rys. 1. Przykładowy obraz z próbnikiem oświetlenia wyposażonym w kołnierz
Fig. 1. Example image of a flange-based light probe

The reflected image is extracted from the image frame manually due to lack of available information about placement of the light probe. It is especially difficult if the background and the reflected images are similar and solid (like clear sky), or if there are a lot of small and similar objects (like leaves in a forest scene).

The light probe extracted image is transformed to the spherical coordinates and used in computer graphics software for light simulation. This technique gives ability of the correct lighting of virtual objects placed in a real scene.

This technique is simple for a static object. Camera or virtual object (and corresponding light probe device) movements need extractions of the reflected image from every frame, which is a time consuming task for a human. Automatic technique is necessary and the flange based light probe is introduced in [2]. The chroma keying and edge detection techniques are used together for the image enhancement, especially two circles of the flange (inner and outer). The Hough transform is applied to estimation of the circle position and radius using the enhanced image shown in Fig. 2.

The computation cost of the Hough transform is very high due to the nonlinear relation between following pixels of the circle during successive accumulation. Matlab based computations are very expensive and the single frame is processed for about 16 minutes (512x512 pixels, 32 radiuses, 100 points of the circle are accumulated) using Pentium 4 at 2.4GHz. High resolution frames (2k or 4k) are used in a digital cinema so computation costs are unrealistic.

One of the optimization methods is parallel processing using GPGPU (General Purpose Graphics Processing Unit). The NVidia CUDA processing platform is very interesting because the code is written using the C language.

Rys. 2. Obraz po przetworzeniu metodą z artykułu [2]
Fig. 2. Enhanced image after application of the algorithm proposed in the paper [2]

2. Hough transform for circles

The Hough transform accumulates values for every circle [1, 4]. The input image is 2D space and the Hough transform output is 3D space in a general form. Two dimensions of the result are related to the position (*x,y*) and correspond to the input image. The third dimension is related to the variable radius.

$$H(x,y,r) = \sum_i I\left(x + r \cdot \sin(\alpha_i), y + r \cdot \cos(\alpha_i)\right) \qquad (1)$$

where:

I – image,
x,y – circle centre,
r – radius,
i – particular angle,
α – angle.

Such formulation of the Hough transform (1) allows accumulation by testing a selected set of points located on the circle. Image pixels are used directly if the position rounding is applied and for small distances between neighborhood pixels the same value is used (accumulated) multiple times. Bilinear interpolation for images allows accumulation of the more correct values. Bilinear interpolation is supported by the CUDA supported GPGPUs but only for the float (32-bits per pixel) image data representation [5, 6]. This is a serious disadvantage because unsigned byte (unsigned char: 8-bits per pixels) is more important for reduction of the main bottleneck of the GPGPUs – the memory transfers between GPGPU and memory [5, 6]. In this implementation the byte based representation of the pixel is used. The image enhancement algorithm [2] uses a color (RGB) image and returns single byte per pixel that is depicted as a grayscale image (Fig. 2).

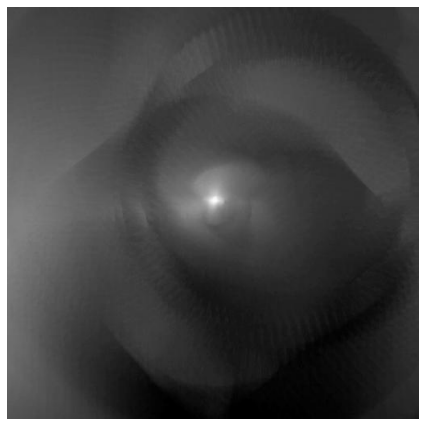
The fixed number of pixels used in computations support the comparison possibility of the circles with different radiuses.

The Hough transform maximum value (2) for a set of radiuses allows estimation of the position and is depicted in Fig. 3.

$$H_{\max}(x,y) = \max_r H(x,y,r) \qquad (2)$$

Additional image normalization is shown in Fig.3. The brightest area in the image corresponds to the best detected circle position.

There are a lot of implementation possibilities of the Hough transform [7-12] using in parallel processing. Theoretical approaches are very important but implementation details, and especially a processing platform, influence the computation time significantly. The most important for particular application (light probe tracking) is the processing time reduction to the more realistic value for the fluent workflow. Assuming 24 frames per second and 60 seconds of the footage, the Matlab implementation needs 16 days of processing, which is not acceptable.



Rys. 3. Reprezentacja transformaty Hougha dla maksymalnej wartości
Fig. 3. Hough transform results for the maximum value

3. CUDA implementation

The direct implementation of formula (1) is used in the considered implementation. The cost of computations of sine and cosine functions is quite low because they are processed inside GPGPU and there are not necessary memory accesses to the LUT (Look-up Table). Such approach is suggested in numerous papers and books [5, 6]. The main cost is the memory transfers between GPGPU and global memory (outside GPGPU chip). Assuming N accumulation points per a circle, there are $X \cdot Y \cdot N \cdot R$ read and $X \cdot Y \cdot R$ (or $X \cdot Y$) write accesses.

Conventional CUDA image processing in parallel assumes assignment of the single thread to the e.g. processed pixel or matrix element but such technique is not always efficient. Two main elements of data organization and processing are the thread block and grid. The block joins of multiple threads for processing the selected memory area and threads cooperation. A shared memory is used for cooperation between threads and storage of the result. After cooperation, the result is transferred from the shared to the global memory. This approach is efficient but depends on numerous factors, especially on the algorithm and data representation. The grid is used for organization of multiple blocks for complete data processing. The block size defines also number of concurrent blocks processed by the available hardware. Blocks are processed in order defined by the CUDA scheduler that is not controlled by the programmer.

Blocks and grid are virtual concept in CUDA and are not related to the data organization. It is possible to use them more freely and reduce processing time or implementation cost.

In this paper a non-standard block and grid assignment is used. Single thread blocks (32x1 size) process 32 radiuses. Every thread is assigned to the different radius. The grid is extended in one dimension 32 times (width). Calculation of the proper 2D position in the image is obtained by the arithmetic shift right (>>5 operation) of the horizontal position of the thread.

Thread blocks process input data organized into a ring. Input data are located in the texture memory intentionally. The shared memory is fast but needs more complicated accesses in comparison to the texture memory. The texture memory is cached, which is used in this implementation. Ring accesses start synchronously for all 32 threads in block and the same angles are processed in parallel. This approach improves a hit ratio in the texture cache memory and the number of accesses to the slow global memory is reduced.

Every circle is sampled by the fixed number of points. Dense sampling allows better estimation of the Hough transform and sparse sampling reduces the computation cost. Dense sampling is important for the result quality. The computation cost is low due

to texture cache even if the same pixel is used in dense sampling approach. CUDA supports bilinear interpolation but not for desired 8-bit data per pixel representation, unfortunately.

The accumulated results are compared in parallel by the threads and the largest one is returned. Corresponding radius for the maximal fitness of the circle in the image at some position can be returned also. Such formulation reduces the number of output data to the global memory.

The position estimated by the Hough transform is used directly or as a starting point for a more precise second estimation step. Next, the Hough transform or another algorithm can be applied.

4. Performance

A set of performance tests for Nvidia G80 GPU (Geforce 8800 GTS, 128 stream processors, 650 MHz core clock, 1625 MHz shader clock, 1944 MHz memory data rate, 256-bit memory interface, PCI Express x16) were prepared. The input image has 512x512 size. The algorithm has fixed computation cost and does not depend on the image content.

The processing time does not depend significantly on the radius and the processing cost is almost a linear function of the number of circle samples (Fig. 4) for the best solution (No. 7).

Many optimization techniques are tested for calculation of formula (1) inside the loop (Tab.1). The 'i' is the loop control variable, 'x0' and 'y0' are positions of the circle centre, alpha is the constant used for successive sampling of the circle, 'R' is the variable radius dependent on the assigned thread.

Tab. 1. Computation cost for radiuses 100-132 and 512x512 image
Tab. 1. Koszt obliczeniowy dla promieni 100-132 i obrazu 512x512 pikseli

No.	Part of kernel code responsible for calculation of the circle points coordinates	Processing time [s] (loop)	Processing time [s] (unrolled loop)

1	x = x0 + R*__sinf(angle);
	y = y0 + R*__cosf(angle);
	angle += alpha;
		2.4	0.9

2	x = x0 + R*__sinf(alpha*i); y = y0 + R*__cosf(alpha*i);	2.6	0.9

3	x = x0 + R*sin(angle);
	y = y0 + R*cos(angle);
	angle += alpha;
		9.1	9.1

4	x = x0 + R*sin(alpha*i); y = y0 + R*cos(alpha*i);	9.2	9.2

5	float t=alpha*i;
	x = x0 + R*sin(t);
	y = y0 + R*cos(t);
		9.2	9.2

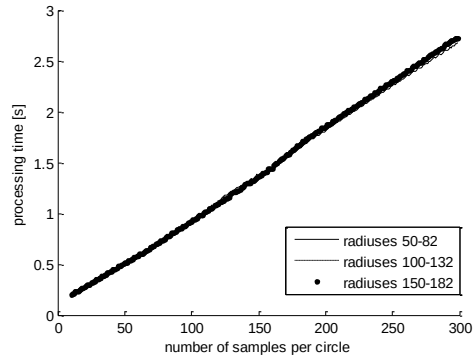
6	float s1,c1;
	__sincosf(alpha*i,&s1,&c1);
	x = x0 + R*s1;
	y = y0 + R*c1;
		2.6	0.9

7	float s1,c1;
	__sincosf(angle,&s1,&c1);
	x = x0 + R*s1;
	y = y0 + R*c1;
	angle += alpha;
		2.4	**0.9**

There are three possible implementations of sine and cosine functions. The conventional 'sin' and 'cos' functions process data with the high precision quality [5, 6] but the computation time is about 9 [s]. This is quite fast in comparison to the Matlab implementation but still far from the real-time processing and the obtained processing time values should be rescaled for the larger image frames and number of tested radiuses.

Two next options are '__sinf', '__cosf' functions or quadrature function '__sincosf'. Unrolling of the loop is very important for further processing time reduction. The processing time is about 0.908 [s] for the last (best) variant. The conventional 'sin' and 'cos' functions are very slow and even loop unroll benefits are not visible.

All threads write the shared memory but only one thread writes result to the global memory. If all of them write the best result, the computation time is increased to 0.913 [s].



Rys. 4. Czas przetwarzania zaimplementowanego algorytmu
Fig. 4. Computation time of the implemented algorithm

5. Conclusions

The GPGPU based implementation of the Hough transform is much faster (about 1000 times) in comparison to the regular Matlab code. Proper analyses of the code multiple variants, the memory organization are important for time efficient results. The proposed techniques are important not only for the Hough transform but for other algorithms with non-linear accesses to the memory.

The obtained implementation is a very useful tool for processing a multiple image and initialization of light probe tracking algorithms.

This work is supported by the UE EFRR ZPORR project Z/2.32/I/1.3.1/267/05 "Szczecin University of Technology - Research and Education Center of Modern Multimedia Technologies" (Poland).

6. References

[1] Hough P.V.C., Arbor A: Method and Means for Recognizing Complex Patterns, US Patent no. 3,069,654, 1962.
[2] Mazurek P.: Estimation of position of the light probe device for photorealistic computer animation purposes, Elektronika – konstrukcje, technologie, zastosowania, R. LII nr1, 2011.
[3] Debevec P.: Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography, SIGGRAPH 98, pp. 189-198, 1998.
[4] Demirkaya O, Asyali M.H., Sahoo P.K.: Image Processing with MATLAB, Applications in Medicine and Biology. CRC Press, 2009.
[5] NVIDIA CUDA C Programming Guide v.3.2, NVidia, 2010.
[6] NVIDIA CUDA, CUDA C Best Practices Guide v.3.2, NVidia, 2010.
[7] Diard F.: Using the Geometry Shader for Compact and Variable-Length GPU Feedback, in GPU Gems 3, NVidia, 2007.
[8] Guil N., Zapata E.L.: A Parallel Pipelined Hough Transform, in Euro-Par, vol II, pp. 131-138, 1996.
[9] Ruiz A., Ujaldon M., Guil N.: Using Graphics Hardware for Enhancing Edge and Circle Detection, in J. Marti et al. (Eds.): IbPRIA 2007, Part II, LNCS 4478, pp. 234–241, 2007.
[10] Underhilla A., Atiquzzamanb M., Ophela J.: Performance of the Hough transform on a distributed memory multiprocessor, Elesevier, Microprocessors and Microsystems 22, pp. 355–362, 1999.
[11] Ujaldon M., Ruiz A., Guil N.: On the computation of the Circle Hough Transform by a GPU rasterizer, Elsevier, Pattern Recognition Letters 29, pp. 309–318, 2008.
[12] Yuen H.K., Princen J., Illingworth J., Kittler J.: A comparative study of Hough transform methods for circle finding, Image and Vision Computing – Special issue: 5th Alvey vision meeting, Vol 8 No 1, 1990.

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*