Hit-or-miss Transformation Implementation on Graphics Processing Unit

Presented in 1st Data Storage Technology Conference (DST-CON 2008)

Navadon Khunlertgit[1], Nipon Theera-Umpon[2] and Sansanee Auephanwiriyakul[1]*

Abstract

The hit-or-miss transformation (HMT), a binary matching between an image and a structuring element, is a morphological operator. It has become a useful tool for shape detection. However, the HMT processing time is too expensive in real applications. To use this technique for real-time processing requires high computational processor. The graphics processing unit (GPU) is designed to draw images or graphics scenes on the computer screens. GPUs have been rapidly developed to respond to the demand for rendering in realistic game applications. However, modern GPUs are programmable in offering the capability to execute the user's code. Due to their high performance and programmability, general purpose computation on GPUs has become a new research field. In this paper, we present an efficient HMT implementation using a GPU. We compare the execution time of this GPU implementation with that of CPU implementation. Our proposed GPU-based HMT perform faster than the CPU-based HMT. This GPU-based HMT can help in speeding up several real-time image processing applications including the defect detection in hard disk drive industrials.

Introduction

The graphics processing unit (GPU) is a processing unit designed for accelerating graphics pipeline operations, such as real-time rendering in game applications. The demands from the markets of these applications have been massively driving the development of GPUs. Therefore, GPUs have been developed to be the processors with high speed and computational capabilities. They have an explicitly parallel programming model which is similar to the Single Instruction, Multiple Data (SIMD). Their performance continues to increase as the transistor counts increase. The latest GPUs have become programmable, granting users to execute their functions. According to this programmability, we can use a GPU not only for traditional graphics applications but also for general-purpose applications. This new generation of GPUs has brought us to the new field of research known as General Purpose computation on GPU or GPGPU (GPGPU, 2005; Pharr and Fernando, 2005).

[1]*Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Thailand*
[2]*Department of Electrical Engineering, Faculty of Engineering, Chiang Mai University, Thailand*
**corresponding author; e-mail: nipon@ieee.org; phone:663944140; fax:6653221485*

The original task of the GPU is for graphics operations. To program GPU in general purpose is complicated for the programmers because of the differences in available resources, implemented features, and versions of specialized graphics. Consequently, there are some efforts to make general-purpose programming easier by making a supportive environment. One such environment that is commonly used is BrookGPU (Buck et al., 2004;Trancoso and Charalambous, 2005; Charalambous et al., 2005). Brook is an extension for C to support the GPGPU. It is a good approach for porting an application.

Several general-purpose applications have been ported to the GPU in many areas such as the Fuzzy C-Means (FCM) algorithm (Hong and Wang, 2004), as well as image processing applications which include Level Set (LS) algorithm (Hong and , 2004), Generalized Distance Transforms (GDT) and Skeletons (Strzodka and Telea, 2004), Image Filtering (Sugita et al., 2003; Fialka and Cadik, 2006), etc. All of them have already been shown to perform faster than CPU applications. Furthermore, there are many other research works reporting experiences in accelerating the execution on the GPGPU (GPGPU,2005;Owens et al., 2005). Trancoso and Charalambous showed that the GPGPU was suitable for more computational intensity applications. Their work presented that the GPGPU can also be used as a low-cost alternative architecture for high-performance computing (Trancoso and Charalambous, 2005).

Our proposed research focuses on another intensive computation which is widely used in image processing applications namely the Hit-or-miss transformation (HMT). The HMT is a well-known morphological template matching technique introduced by Serra (1982). Its objective is to locate known objects in an image. It has obvious applications in computer vision and image analysis. It is also useful for defect detection in many industrial fields. There are many research works that apply the HMT as a primary tool for shape detection (Zhao and Daut, 1991; Khosravi and Schafer, 1996), However, the HMT is seldom used in real-time processing. This is because its processing time is too expensive. For this reason, using the HMT in real applications requires a high computational processor.

According to the SIMD capability of the GPU we mentioned earlier, it makes the GPU suitable for running image processing tasks, which usually involve similar calculations operating on an entire image. Therefore, we investigate the possibility of the HMT implementation on the GPGPU. The performance of the GPU-based HMT is also compared to that of the CPU-based.

The rest of this paper is organized as follows. Section II presents a background of the GPU, along with its programming environments. Section III shows a brief description of the HMT and the description of the proposed GPU-based HMT implementation. The experiments and the results are shown in Section IV. Finally the conclusions of this paper are presented in Section V.

Graphics Processing Unit

A. Architecture

The original purpose of the GPU is for the rendering process which computes pixels on screen by projecting 3-D coordinate objects. It offers a large degree of parallelism for accelerate this computation.

Its model is similar to the Single Instruction, Multiple Data (SIMD). Therefore, the GPU is naturally suitable for highly data-parallel computations.

The general GPU is consisted of 2 different types of processing units: vertex processors and pixel (or fragment) processors. The vertex processor performs mathematical operations that transform a vertex into a screen position. The pixel or fragment processor, also known as pixel shader, performs the texturing operations.

Figure 1 shows a diagram of a GPU pipeline. The vertex processors transform 3-D triangles into 2-D triangles by projecting their vertices onto the screen. Then, these 2-D triangles are rasterized into fragments for input to the fragment processors. Finally, the fragment processors determine the color of pixels.

B. Programming Environment

Programming GPU in a high-level language is recently developed to offer GPU programmability. At first, the high-level language environments were developed for graphics operation such as Cg from NVIDIA, and OpenGL Shading Language. Although they are helpful for graphics applications, still they are not trivial to be using by general-purpose applications. Therefore, some researchers develop high-level language environments for general-purpose programming. The environment that we use in this research is BrookGPU from Standford University (Buck et al., 2004).

Brook abstracts the GPU as a stream processor. It is an extension of C that contains new stream variables and a new key word that indicates certain functions as kernels. The stream is a data collection which can be operated in parallel. The kernel is a special function that is applied to the elements of the input streams to produce each element of the output streams. Brook compiles kernels into Cg code and generates C++ code to connect to the kernels. Brook runtime allows the users to select the processor to execute their code depending on the value of the Brook additional environment variable namely BRT_RUNTIME.

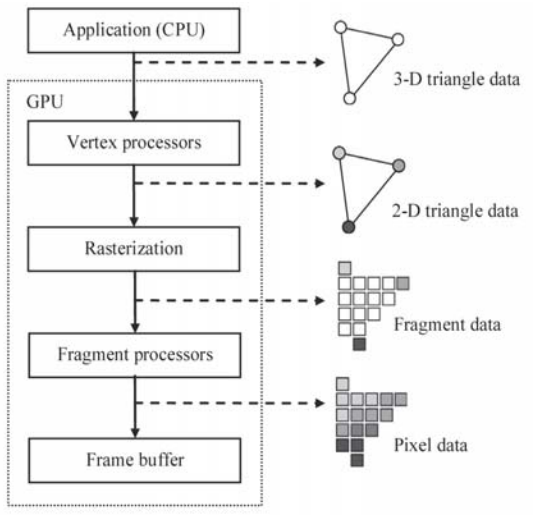


Figure 1. GPU pipeline architecture.

GPU-based HMT Implementation

The hit-or-miss transformation (HMT) is the result of the intersection of two morphological erosion operators defined as

$$A \otimes B = (A \Theta X) \cap [A^C \Theta (W - X)]$$

where A is a binary image, A^C is the complement of A, X is an object from the structuring element B, and $W-X$ is a background containing the structuring element B. $A \Theta B$ is the morphological erosion of the image A by the structuring element B, where $X \Theta Y = \{z : Y + z \subseteq X\}$.

The HMT like any other morphological operators has loop codes for indexing each pixel in both input image and structuring element. Each pixel in the input image does not change during the procedure. Accordingly, there are no dependencies in the data access in the loop. To get the effectiveness in porting to the GPGPU, we must transform loop into vectorizable which can be operated in parallel.

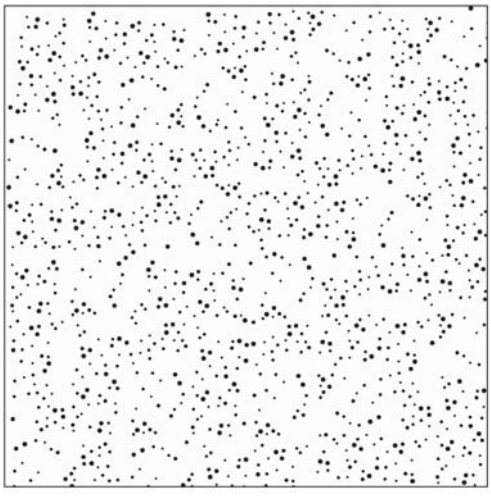
In our implementation, we convert the indexing loop which is used to index pixel in the input image into vectorizable. Then we extract it and replace by a kernel function. Before the kernel call, we have to set the array of the input image into the input stream. Then, after the kernel call, we also have to set the output stream back to array of the output image.

It should be noted that the application of the proposed GPU-based HMT may be limited by the GPU limitations such as the size of the stream, the number of parameters, etc. However, there is a research that analyzed these limitations and proposed the solution (Trancoso and Charalambous, 2005).

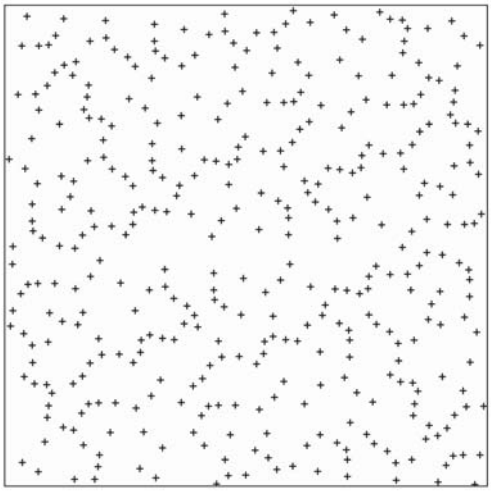
Experimental Results

A. Template Matching Results Using GPU-based HMT

We have implemented the GPU-based HMT to process large size images to speed up the execution time. As shown in Figure 2, for example, the 1024X1024 input image that contains disks with different diameters of 5 to 10 pixels. We apply our proposed GPU-based HMT to locate all disks with diameter of 5 pixels. The original image and the resulting image that shows the location of all 5-pixel diameter disks are shown in Figure 2(a) and 2(b), respectively. The crosses in Figure 2(b) indicate the locations of 5-pixel diameter disks achieved by the HMT. Because the input image is the synthetic one that we generate with the prior knowledge of all disk locations, we can check whether the output image is correct. We found that it is completely correct which implies that the proposed GPU-based HMT is implemented correctly.



(a) Input image.



(b) Result from GPU-based HMT indicating locations of 5-pixel diameter disks.

Figure 2. GPU-based HMT in template matching.

B. Computation Performance

The advantage of the proposed GPU-based HMT is to reduce the execution time. We perform a quantitative performance comparison between the GPU-based HMT and CPU-based HMT by using test images with various sizes. The sizes of the test images are 256X256, 512X512, and 1024X1024. The test platform is a 2.41GHz Dual-core AMD Athlon processor, 1GB RAM, and NVIDIA GeForce 8500GT 256MB at PCI-E. The chart in Figure 3 shows the ratios of the computation times of the CPU-based HMT and the proposed GPU-based HMT. We can see that the execution by the GPU is faster than that by the CPU for all cases. We observe that when the input image size increases, the proposed GPU-based HMT can be performed faster by the larger ratio. It can run faster by 44 times when the input image size is 1024X1024.

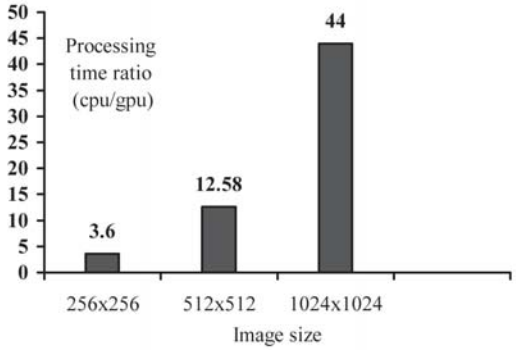


Figure 3. Computation performance comparison of GPU-based HMT and CPU-based HMT for different sizes of images.

Conclusions

In this paper, we present the implementation to perform the hit-or-miss transformation (HMT) on the GPU. The results show that the HMT can be properly implemented on the GPU. The results also show that the proposed GPU-based HMT implementation is faster than the CPU-based implementation. When the size of an input image is bigger, the ratio of the processing times of the CPU-based HMT implementation to that of the GPU-based HMT implementation is even larger. That means the proposed GPU-based HMT provides more speedup over the CPU-based HMT when the input image is bigger. This result makes the GPU-based HMT an alternative approach to implement the HMT in real-time.

The GPU can potentially be adapted to implement many other algorithms that require considerable computation such as shape detection from a real-time video scene. As the development of the GPU still depends on realistic graphics applications, the GPU performance will be improved. That will open an opportunity to more applications that may not be implemented with the current technology. In the future, we plan to continually extend our work to the gray-scale images and then apply to real applications.

Acknowledgment

This work was supported by the Hard Disk Drive Institute (HDDI) and the National Electronics and Computer Technology Center (NECTEC). We would like to thank Lanna Thai Electronic Components (LTEC) Ltd. for the valuable information and thankful cooperation during this research.

References

Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P., 2004. "Brook for GPUs: Stream Computing on Graphics Hardware," *ACM Transactions on Graphics*, pp. 777-786.

Charalambous, M., Trancoso, P., and Stamatakis, A., 2005. "Initial Experiences Porting a Bioinformatics Application to a Graphics Processor," *The 10th Panhellenic Conference in Informatics*, pp.415-425.

Fialka O. and Cadik M., 2006. "FFT and Convolution Performance in Image Filtering on GPU," *Conference on Information Visualization*, pp. 609-614.

"GPGPU, General purpose computation using graphics hardware," 2005; http://www.gpgpu.org/.

Harris, C. and Haines, K. 2005. "Iterative Solutions using Programmable Graphics Processing Units," *The 14th IEEE International Conference on Fuzzy Systems*, pp. 12-18.

Hong, J. Y. and Wang, M. D. 2004. "High Speed Processing of Biomedical Images Using Programmable GPU," *The 2004 International Conference on Image Processing*, pp.2455-2458.

Khosravi, M.and Schafer, R.W. 1996. "Template matching based on a grayscale hit-or-miss transform," *IEEE Transactions on Image Processing*, pp.1060-1066.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Kruge,r J., Lefohn, A. E. and Purcell, T. J. 2005."A Survey of General-Purpose Computation on Graphics Hardware," *Eurographics*, pp.21-51.

Pharr, M.and Fernando, R. 2005.*GPU Gems 2 Programming Techniques for High Performance Graphics and General-Purpose Computation*, Addison Wesley.

Trancoso, P. and Charalambous, M. 2005. "Exploring Graphics Processor Performance for General Purpose Applications," *The 8th Euromicro Symposium on Digital System Design, Architectures, Methods and Tools*, pp. 306-313.

Serra, J. 1982.*Image Analysis and Mathematical Morphology*, New York: Academic Press.

Strzodka, R. and Telea, A.2004. "Generalized Distance Transforms and skeletons in graphics hardware," *EG/IEEE TCVG Symposium on Visualization*, pages 221-230.

Sugita, K., Naemura, T., and Harashima, H. 2003. "Performance Evaluation of Programmable Graphics Hardware for Image Filtering and Stereo Matching," *ACM symposium on Virtual Reality Software and Technology*, pp.176-183.

Zhao, D. and Daut, D.G. 1991. "Shape recognition using morphological transformations," *1991 International Conference on Acoustics, Speech, and Signal Processing*, pp.2565-2568 .