A Framework for Multisensor Image Fusion using Graphics Hardware

Seung-Hun Yoo
Dept. Electronics Engineering
Korea University
Seoul, Korea
capstoney@korea.ac.kr

Sung-Up Jo
Dept. Electronics Engineering
Korea University
Seoul, Korea
easyup@korea.ac.kr

Ki-Young Choi
Dept. Electronics Engineering
Korea University
Seoul, Korea
2xx195@korea.ac.kr

Chang-Sung Jeong
Dept. Electronics Engineering
Korea University
Seoul, Korea
csjeong@korea.ac.kr

***Abstract*—This paper presents a framework for pixel-level multisensor image fusion algorithm using graphics hardware. When it comes to the fusion technology of recent times, not only its visibility through information maximization which is brought about by the progress in sensor technology but also the importance of fusion processing speed has increased. The GPU provides high-performance compared to its price and executes rapid computation by supporting internal parallel processing can be used to improve the image fusion processing speed. Through an implementation of pixel-level image fusion methods using GPU and CPU, this paper shows an increase in the execution time of the GPU compared to the execution time of the CPU as the multi-scale level and resolution increase.**

Keywords: Multisensor fusion, GPU, pixel-level image fusion, parallelism.

I. Introduction

With the rapid development in GPU performance and gradual extension in programmable characteristics in recent times, the possibility of utilizing a GPU for general purposes other than 3D graphics has increased [5]. The processing speed of a recently developed GPU surpasses the performance of a recently developed CPU by a significant margin. Also, another trend in the development of GPUs is that the internal pipeline functions of the GPU can be programmed by the user. This makes it possible for the GPU to be used for general purposes, and although currently restricted, the GPU can eventually be used for purposes similar to those of the CPU.

For improving computer or human interpretation, image fusion has advanced rapidly in the past few years. As a result, many image fusion techniques have been developed in a wide variety of applications such as concealed weapon detection, remote sensing, intelligent robots, digital camera applications, medical diagnosis and surveillance systems.

When it comes to the fusion technology of recent times, not only its visibility through information maximization which is brought about by the progress in sensor technology but also the importance of fusion processing speed has increased. For example, the EO (Electro Optical)/IR (Infrared) aviation images received from an unmanned aircraft have an extremely high resolution. Substantial memory and computing power are required for a high-speed fusion of high resolution images acquired multiple sensors [6]. Image fusion-exclusive processors are developed and used for the fusion processing of real-time images and videos [6] [7]. But when such fusion-exclusive hardware is used, restrictions such as a lack of system extendibility are caused due to memory deficiency, price increase, and modification or addition to algorithm.

According to the stage at which the fusion takes place, image fusion algorithms can be considered to be at one of four levels of abstraction [3] [12]: signal-level; pixel-level; feature-level; and symbolic-level. Currently, it seems that most image fusion applications employ pixel-level methods. At pixel-level fusion, input images are combined by considering individual pixel values or small window centered at the current pixel position or specific regions in order to determine the fusion method. The advantage of pixel-level fusion is implying of original information for source images. Furthermore, the algorithms are easy to implement.

Most of image fusion algorithms require a simultaneous processing into high resolution image or multiple images under a same instruction, namely, they require a SIMD-type approach. This paper presents a GPU-based framework for a high-speed fusion processing of a multisensor image, and shows an improvement in the fusion processing speed in comparison to the CPU. Pixel-level image fusion algorithms such as weighted average, PCA, laplacian pyramid, discrete wavelet transformation are implemented and their execution times are evaluated in our experiment. These algorithms are applied with implementation techniques such as the usage of a structure which utilizes Render-To-Texture (RTT) and Multi-Pass Rendering (MPR) for a high-speed execution in the fragment processor of GPU [4] [13] [17].

This paper is organized as follows. First GPU architecture and the GPU programming model are presented in Section 2, and pixel-level image fusion algorithms are described in Section 3. In Section 4, the structure of the framework for the high-speed image fusion and the implementation process in the GPU are discussed. Section 5 shows a performance comparison through a speed measurement of the fusion methods implemented in the CPU and GPU, Section 6 conclude our work.

II. Graphics Processing Unit

GPU is a dedicated graphics rendering device that accelerates speed of generating 3D scene. With simple and parallel architecture, the latest GPU's computational speed is at least 3-4 times faster than CPU's and the gap between GPU and CPU is increased. Particularly, the benefits of harnessing the massive computing power and speed of the SIMD architecture of GPUs are numerous while at the same time the CPU can be used on other tasks [4]. In this section, we briefly introduce the GPU architecture and the GPU programming model.

A. Graphics Pipeline

The design of graphics processors has traditionally followed a common structure known as the graphics pipeline [4] [5] [13] [14] [15]. This graphics pipeline is designed to allow hardware implementations to maintain high computation rates through parallel execution. The pipeline is divided into primarily three stages:geometry stage, rasterization stage, fragment stage. The input to the pipeline is a list of geometry, expressed as vertices in object coordinates; the output is an image in a framebuffer. In hardware, each stage is implemented as a separate piece of hardware on the GPU. A simplified current graphics pipeline is shown in Fig. 1.

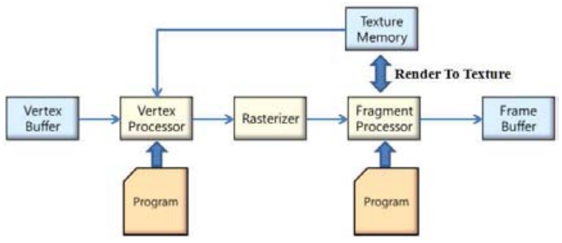

Fig. 1. Graphics Pipeline

Graphics pipeline allows for many levels of parallelism resulting in high computational rates and throughput [15] [16]. Each of the stages is typically implemented as individual hardware components that allow for the overlapping execution of each task (i.e. task-level parallelism). In addition, the pipeline is composed of multiple vertex and fragment(pixel) processors, allowing for many data-parallel operations over the supplied set of input vertices and resulting pixels (i.e. data-level parallelism). For example, common hardware designs use 6-8 vertex processors and 32-48 fragment processors. What is more, within the processing of a single element, we can exploit instruction-level parallelism. While early implementations of the graphic pipeline were based entirely on a fixed-function design, the last several hardware generations have introduced user-level programmability. As shown in Fig. 1, programmers can supply code for both the vertex and fragment processors. Each program is typically limited to a fixed number of available input and output parameters, registers, constant values, and overall number of instructions. The input parameters to programs can be scalar values, two-, three-, or four-component vectors, or arrays of values (scalar or vector) that are stored in the texture memory of the graphics card.

B. GPU Programming Model

GPGPU computing presents challenges even for problems that map well to the GPU, because despite advances in programmability and high-level languages, graphics hardware remains difficult to apply to non-graphics tasks.

We describe the stream programming model which is the basis for programming GPUs today. In the stream programming model, all data is represented as a stream which define as an ordered set of data of the same data type and kernels are small programs operating on streams. Kernels take a stream as input and produce a stream as output. Like above Fig. 1, the graphics pipeline is structured as stages of computation connected by data flow between the stages. This structure is analogous to the stream and kernel abstractions of the stream programming model. Stream programming model on GPU offers SIMD-based parallel processing power.

A typical GPGPU program uses the fragment processor because the programmable stage with the highest arithmetic rates is the fragment processor. Fragment processors have the ability to fetch data from textures, so they are capable of memory gather. However, the output address of a fragment is always determined before the fragment is processed-the processor cannot change the output location of a pixel-so fragment processors are incapable of memory scatter. Programmers must resort to various tricks to achieve a scatter. GPU programming model has the many merits for improvement of system performance, but there are many limitations of programming for general application.

Specially, the GPU programming has the advantage and disadvantage in memory access sides. For example, a frame buffer memory can be display or can be written to texture memory, this work called render to texture. It is implemented direct feedback of GPU output to input without going back to CPU, so more efficient results is made. On the other hand, fragment processor can't use indirect memory addressing in writing operation and same memory as input and output. Shading languages are directly targeted at the programmable stages of the graphics pipeline. The most common shading languages are Cg, the OpenGL Shading Language, and Microsoft's High-Level Shading Language. Cg ("C for graphics") [21] is a high-level shading language developed by NVIDIA, its syntax and semantics are very similar to the C programming language. Most of our fusion processes are implemented on GPU and we have Cg and OpenGL with graphics user interface.

III. Pixel-Level Image Fusion Method

Pixel-level fusion methods can be roughly divided into two groups, grayscale fusion methods and color fusion methods [1] [3] [12]. Grayscale fusion methods are divided into multi-scale-decomposition-based (MDB) fusion methods and non-multi-scale-decomposition.based (NMDB) fusion methods. Color fusion methods are divided into true-color methods and false-color methods. Some representative algorithms such as weighted averaging, laplacian pyramid, DWT, TNO are reviewed in this section.

A. Weighted Averaging

The probably most straightforward way to build a fused image of different sensor images is performing the fusion as a weighted superposition of input images. It takes the weighted average of the source images pixel by pixel.

$$F(i,j) = w_1 I_1(i,j) + w_2 I_2(i,j), \tag{1}$$

where w_1 and w_2 are the weighting coefficients, I_1(i, j) and I_2(i, j) are the values of pixel (i, j) in the source image. The simple approach to determine of weighting coefficients is to build the fused image by the application of a simple nonlinear operator such as max or min. The other optimal weighting coefficients can be determined by a principal component analysis (PCA) of all input intensities [3] [12]. PCA is fundamentally limited by it use of global variance as a saliency measure and will always assign a stronger weight to the source image with the greater variance.

B. Laplacian Pyramid

Image pyramids have been initially described for multiresolution image analysis and as a model for the binocular fusion in human vision [18]. Multi-resolution image pyramids are constructed by successive filtering and down sampling of source image. Due to sampling, the image size is halved in both spatial directions at each level of the decomposition process, thus leading to a multiresolution signal representation. Several pyramid-based fusion schemes have been proposed in recent years. Only the Laplacian pyramid is considered here as it has been shown to perform best among all pyramids [18] [19]. Each level of the Laplacian pyramid is recursively constructed from its lower level by applying the following four basic steps: blurring (low-pass filtering); subsampling (reduce size); interpolation (expand); and differencing (to subtract two images pixel by pixel) [20]. The detailed images, resulting from the difference operation, are fused using a maximum selection rule at each point and then combined with the same image from the lower pyramid level.

C. Discrete Wavelet Transform

A common wavelet transform technique used for fusion is the DWT (Discrete Wavelet Transform) [9] [14]. The one-dimensional (1-D) DWT involves successive filtering and down sampling of the signal. For images the 1-D DWT is used in two dimensions by separately filtering and down sampling in the horizontal and vertical directions. This gives four sub-bands at each scale of the transformation with sensitivity to vertical, horizontal and diagonal frequencies. DWT coefficients from two input images are fused (pixel-by-pixel) by choosing the average of the approximation coefficients at the highest transform scale, and the larger absolute value of the detail coefficients at each transform scale. Then an inverse DWT is performed to obtain the fused image. It has been found to have some advantages over pyramid schemes such as: increased directional information [1]; no blocking artifacts that often occur in pyramid-fused images [1]; better signal to noise ratios than pyramid-based fusion; improved perception over pyramid-based fused images, compared using human analysis.

D. TNO

Color fusion can consist of assigning the two source image to the three planes of an RGB color image. Toet has proposed a straightforward improvement to the basic RGB color fusion scheme which has become known as the TNO method [11]. In this method, the common component, defined as the lowest-valued pixel of each pair of pixels, between two images is calculated. This is subtracted from the two images to produce the unique information in each image. These unique components from each opposing source image are removed from each source image and then the resulting images are assigned to the red and green channels of the fused image.

IV. The Framework for Image Fusion

This section shows the proposed framework for the multi-sensor image fusion using GPU and describes how the pixel-level fusion algorithms were implemented on GPU.

A. The Proposed Framework

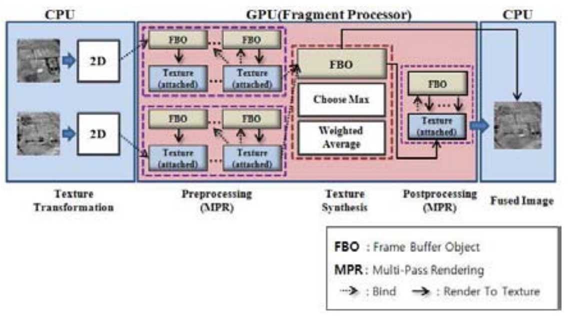

Fig. 2. The Proposed framework for multisensor fusion using graphics hardware

Fig. 2 shows the proposed framework for the multisensor image fusion using GPU. This framework is roughly divided into 4 stages: texture transformation, preprocessing, texture synthesis, and post-processing. In the texture transformation stage, an operation which converts the 8bit gray or 24 bit color input images into a 2-dimensional texture form so that the entire fusion process can be executed by the fragment processors. After the conversion to texture, it is loaded as texture memory according to the functions of the OpenGL. Textures in texture memory are used to input of the fragment program and the fragment program is executed by fragment processors. The output of this process is another texture memory attached to the FBO. This process is called Render-To-Texture which has a feedback structure that enables re-usage by sending the processing results of the fragment processor to the texture memory instead of the framebuffer which is the standard output object. Each fragment program represents one pass of a multi-pass computation on the GPU. So the

Multi-Pass Rendering (MPR) technique is used to execute the multiple-fragment program of the fusion algorithms. All processes executed in the fragment processor for image fusion take on a RTT structure, and the fusion processing can be executed at a high-speed through the utilization of MPR in order to execute more than one fragment program which is needed for the pre-processing and post-processing procedures.

B. Implementation on GPU

The implementation of entire process is summarized as follows. First of all, write each of the functions executed in the algorithms on the fragment program and store the input images in the texture memory. Each of the fragment programs is executed by the fragment processor and the resulting images are stored again in the texture. Fig.3 illustrates the structure of each fragment program in each stage of the framework for the 4 types of pixel-level image fusion algorithms observed in Section 3. The part marked with a round rectangle represents the fragment program, and the arrow represents the input/output of the texture.

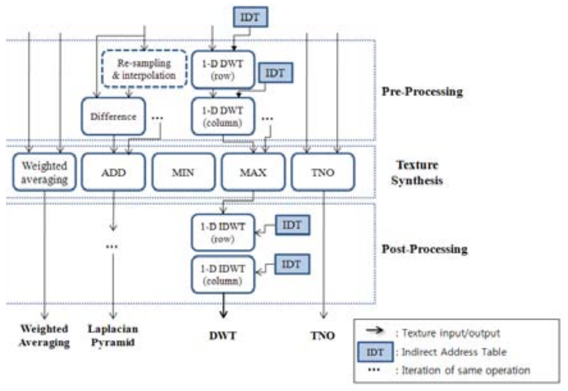

Fig. 3. The structure of each fragment program in stages of the framework

When observing the algorithms individually, weight averaging and TNO method are consist of a single fragment program and the fragment program receives 2 inputs of texture, and a synthesized texture is generated.

```
struct Output {
            float4 color : COLOR;
};

Output Weighted Averaging(half2 coords : TEX0,
            uniform samplerRECT texture1,
            uniform samplerRECT texture2,
            uniform float fuse_param)
{
        Output OUT;
        float w = fuse_param;

        half4 image1 = texRECT(texture1, coords);
        half4 image2 = texRECT(texture2, coords);

        OUT.color = w * image1 + (1 - w) * image2;
        return OUT;
}
```

```
struct Output {
            float4 color : COLOR;
};

Output TNO(half2 coords : TEX0,
            uniform samplerRECT texture1,
            uniform samplerRECT texture2)
{
        Output OUT;
        // Two input texture
        half4 image1 = texRECT(texture1, coords);
        half4 image2 = texRECT(texture2, coords);

        // Calculation of Common Component
        half4 min_image = min(image1, image2);

        // Calculation of Unique Component
        OUT.color.r = image1 - (image2 - min_image);
        OUT.color.g = image2 - (image1 - min_image);
        OUT.color.b = (image2 - min_image) - (image1 - min_image);
        return OUT;
}
```

Fig. 4. An example of the fragment program in Cg. Weighted Averaging(left) and TNO(right)

Fig. 4 illustrates an example implemented by an actual usage of Cg. In the case of the laplacian pyramid algorithm, the reduce/expand calculation was executed through the re-sampling and interpolation provided by the OpenGL. Also, with an increase in the pyramid level, the structures shown in Fig. 3 will take on a more complex form. In the case of the DWT synthesis, the convolution-based DWT was implemented in the GPU. At each level, the 2D DWT is achieved by performing 1D DWT first horizontally and then vertically. For the multiplication and addition calculations required in each rendering process of the DWT/IDWT, an indirect address table according to the method presented by J. Wang should be composed and used as a texture input for the fragment program [5].

V. Experiments

To evaluate the performance of the pixel-level fusion algorithms in the GPU, the fusion processing speed was measured by implementing each of the aforementioned 4 types of algorithms in the CPU and GPU, and changing the image resolution. In the case of the weighted averaging fusion method, weight value 0.5 was simply applied to each of the input images and the implementation speed was measured. In the case of the laplacian pyramid and DWT fusion method, a classification into level 1 and level 3 was made and each of the resulting speed was measured. This test was executed in an environment using Intel Core2 CPU 6600 2.4GHz and the NVIDIA Geforce 8600 GTS GPU. As a standard in the speed measurement, the speed of image fusion was calculated excluding initialization and configuration of OpenGL and Cg.

(unit: sec)

Resolution	Processor	Average	Laplacian Pyramid (Level=3)	Daub. Wavelet	Daub. Wavelet	TNO
				Level=1	Level=3	
256x256	CPU	0.015	0.080	0.046	0.121	0.015
	GPU	0.010	0.067	0.042	0.107	0.012
512x512	CPU	0.016	0.280	0.125	0.364	0.031
	GPU	0.012	0.135	0.089	0.251	0.014
1024x1024	CPU	0.031	1.237	0.515	1.439	0.094
	GPU	0.022	0.762	0.327	0.931	0.027
2048x2048	CPU	0.109	3.031	3.953	5.840	0.375
	GPU	0.033	2.642	1.165	2.528	0.045

Fig. 5. Execution time of image fusion

Fig.5 shows the time measured for each of the fusion algorithms in each resolution in numerical values. When comparing according to different fusion methods, the weighted averaging method is the simplest and fastest for implementation. The weighted averaging method, which almost does not require the optimization of the program to implementation, clearly shows the difference in the performance of the two processors through the difference in the execution speed of the GPU and CPU. A speed difference of over 30% shows the performance difference achieved when the 128 fragment processors in the GPU are executed through a single fragment program as opposed to a single-processor processing of the CPU. The multi-resolution fusion method shows an increase in the execution time of the GPU compared to the execution time of the CPU as the level and resolution increase.

VI. Conclusion

Recently, the commodity GPU is being used in not only 3D graphics rendering but also in general-purpose computation (GPGPU) due to an increase in the goods price/performance ratio and hardware programmability as well as the huge computing power and speed of the GPU. Pixel-level image fusion algorithms execute a high volume of image data under a same command. Also, as the importance of a high-resolution/high-speed image fusion increases with the development in sensor technology, image fusion processing using the GPU will prove to be a positive solution. This paper presented a framework for pixel-level image fusion using the GPU, and actual basic fusion algorithms were implemented in the CPU and GPU for a performance comparison. As a result, an improvement of over 30% in the execution speed compared to the speed when using the CPU has been demonstrated through the data-level parallelism, task-level parallelism, and instruction-level parallelism found in the internal structure of the GPU and programming models; and implementation techniques which utilized Render-To-Texture and Multi-Pass Rendering.. In the future, a framework for an effective fusion processing can be achieved through the usage of the CUDA (compute unified device architecture), a next-generation GPU of NVIDIA, which will enable a more independent arrangement processing and a wide variety of memory usage to be applied to the substantial number of image fusion algorithms that are currently in development.

Acknowledgements

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment), Brain Korea 21 projects in 2008, and Seoul Forum for Industry-University-Research Cooperation.

References

[1] Yang, Yi Han, Chongzhao Kang, Xin Han, Deqiang, "An Overview on Pixel-Level Image Fusion in Remote Sensing", *IEEE conference, Automation and Logistics*, pp.2339-2344, Aug. 2007.

[2] Gemma Piella, "A general framework for multiresolution image fusion: from pixel to regions", *Information Fusion*, pp.259-280,2003.

[3] M. Smith and J. Heather, "Review of image fusion technology in 2005", *www.waterfallsolutions.co.uk*, 2005.

[4] John D. Owens, David Luebke, "A Survey of General-Purpose Computation on Graphics Hardware", in *Eurographics 2005*, State of the Art Reports, pp.21-51, August 2005.

[5] T. T. Wong, C. S. Leung, P. A. Heng and J. Wang, "Discrete Wavelet Transform on Consumer-Level Graphics Hardware", *IEEE Transactions on Multimedia*, Vol. 9, No. 3, pp.668-673, April 2007.

[6] A.Ardeshir Goshtasby, Guest Editors and Stavri Nikolov, "Image Fusion: Advances in the State of the Art", *Information Fusion*, Vol. 8, issue 2, pp.114-118, April 2007.

[7] Fay, D. Ilardi, P. Sheldon, N. Grau, D. Biehl, R. Waxman, A, "Realtime image fusion and target learning & detection on a laptop attached processor", *Information Fusion, International Conference*, 2005.

[8] Nolan Godnight, Rui Wang, and Greg Humphreys, "Computation on Programmable Graphics Hardware", *IEEE Computer Graphics and Applications*, 2005.

[9] G.Pajares and J.M.de la Cruz, "An wavelet-based image fusion tutorial," *Pattern Recognition*, 37, pp.1855-1872 2004.

[10] http://www.nvidia.com/dev_content/nvopenglspecs/GL_EXT_framebuffer_object.txt

[11] A. Toet, J.J. Van Ruyven, J.M. Valeton, "Merging thermal and visual images by a contrast pyramid", *Optical Engineering 28 (7)*, pp.789-792, 1989.

[12] Rick S. Blum and Zheng Liu, *Multi-Sensor Image Fusion and Its Applications*. Taylor & Francis, CRC Press, July 2005.

[13] J. L. Cornwall, "Efficient multiple pass, multiple output algorithms on the GPU", in 2nd *European Conference on Visual Media Production* pp.253-262, 2005.

[14] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, F. Tirado, "Parallel implementation of the 2D Discrete Wavelet Transform on Graphics Processing Units: Filter-Bank versus Lifting, IEEE Transaction on Parallel and Distributed Computing versus Lifting", *IEEE Transactions on Parallel and Distributed Computing*(Magazine), 2008.

[15] Matt Pharr, Randima Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley.

[16] P.S. McCormick, J. Inman, J.P. Ahrens, C. Hansen, G. Roth, "Scout: A hardware-accelerated system for quantitatively driven visualization and analysis", in *Proceedings of IEEE Visualization*, pp. 171.178, 2004.

[17] James Fung, Steve Mann, "Computer Vision Signal Processing on Graphics Processing Units", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (ICASSP), 2004.

[18] Y. Zheng, E. A. Essock, B. C. Hansen, and A. M. Haun, "A new metric based on extended spatial frequency and its application to DWT based fusion algorithms", *Information Fusion* 8, pp.177-192, 2007.

[19] Y. Zheng, E.A. Essock, B.C. Hansen, "An advanced image fusion algorithm based on wavelet transform incorporation with PCA and morphological processing", Proc. *SPIE* 5298 pp.177-187, 2004.

[20] P.J. Burt, E. Adelson, "The Laplacian pyramid as a compact image code, IEEE Trans. Commun". *IEEE Transactions on Communications* pp.532-540, 1983.

[21] Randima Fernando, Mark J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison Wesley.