Image Blending Techniques Based on GPU Acceleration

Jung Soo Kim
Department of Electronics and Computer Engineering
Hanyang University
Seoul, Korea
+82 2-2220-4701
jungsookim4080@gmail.com

Min-Kyu Lee
Department of Electronics and Computer Engineering
Hanyang University
Seoul, Korea
+82 2-2220-4701
hanlovelan@hanyang.ac.kr

Ki-Seok Chung*
Department of Electronics and Computer Engineering
Hanyang University
Seoul, Korea
+82 2-2220-4701
kchung@hanyang.ac.kr

ABSTRACT

Today, image blending has been used for a high-resolution image in medical, aerospace, and even defense areas. To blend images, several filters and various processing steps such as Gaussian pyramid, Laplacian pyramid, and multi-band computation will be needed. However, these computations consist of a large amount of arithmetic operations. As the processing capability of graphic processing units (GPUs) grows very rapidly, GPUs have commonly been used to supplement central processing units (CPUs) for high-performance computing. By employing hardware accelerators such as GPU, a significant speedup can be achieved. In this paper, we present an implementation of fast image blending methods using compute unified device architecture (CUDA). The proposed implementation utilizes a shared memory in GPU better than conventional implementations leading to a better speed-up. The proposed implementation of this paper shows an improvement of 3.9 times in the overall execution time compared to a conventional implementation.

CCS Concepts

• Computing methodologies→Computer graphics→Image Manipulation→Image processing.

Keywords

Image blending; GPGPU; CUDA; padding; shared memory; image pyramid; multi-resolution spline; multi-band blending.

1. INTRODUCTION

Image blending is a technique to combine several images in order to form a single output image. In the past, image blending had been adopted to combine only the images. Since multi-band blending [1-3], which is also known as multi-resolution spline, was introduced, image blending is used not only to blend images but also to correct and calibrate images from different sensors to make a natural conjunction between images at the joints. Because of these reasons, multi-band blending has often been adopted in a variety of computer vision areas such as medical and aerospace applications.

ICIGP 2018, February 24–26, 2018, Hong Kong, Hong Kong
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6367-9/18/02…$15.00
https://doi.org/10.1145/3191442.3191471

It is strongly required for processors to have a high processing capability to satisfy the growing need for processing a large amount of data. For CPUs, the processing power is improved mainly either by increasing the clock frequency or by increasing the number of cores. However, neither one will achieve a groundbreaking performance improvement. Thus, it is necessary a new processor architecture to achieve a much better performance.

One of the widely adopted methods to overcome the CPU's limitation is to employ General-Purpose Graphics Processing Unit (GPGPU) [4]. GPU had been mainly developed to speed up only the graphic processing. However, due to its relatively regular hardware architecture, the number of cores in a GPU has increased tremendously. When the GPU executes massively data-parallel applications, thousands of threads can be executed in parallel resulting in an excellent speed-up in many data-parallel applications over the CPU. Therefore, more and more application areas find it very advantageous to employ GPUs to achieve performance improvement. Compute unified device architecture (CUDA) is a parallel programming framework and an application programming interface (API) model provided by NVIDIA. It allows software developers to use a CUDA-enabled GPU for general purpose processing [5].

The higher the required resolution gets, or the more images the image blender has to take care of, the more critical the computational capability becomes. In this paper, to guarantee sufficient processing power, the CUDA framework is employed for image blending.

To maximize the performance improvement by utilizing the GPU, we have attempted two implementations: computation without a shared memory and that with a shared memory option [6]. It is to show that it is very advantageous to utilize the shared memory option to get better performance. Without the shared memory option, each thread must copy the data from a global memory called Graphics DDR SDRAM (GDDR) to cores directly. On the other hand, with the shared memory option, data is copied from the global memory to the shared memory so that it should be possible that threads in the same block can share the copied data. In this paper, we show how the shared memory option should be utilized to reduce the memory loading time. This is the main contribution of this paper.

The rest of this paper is organized as follows. We will introduce previous studies on the image blending in Section 2 and the CUDA computing framework in Section 3. We propose our proposed implementation of the image blending in Section 4. Next, we will show experimental results and evaluate the proposed scheme in terms of execution time in Section 5. Finally, we conclude our work in Section 6.

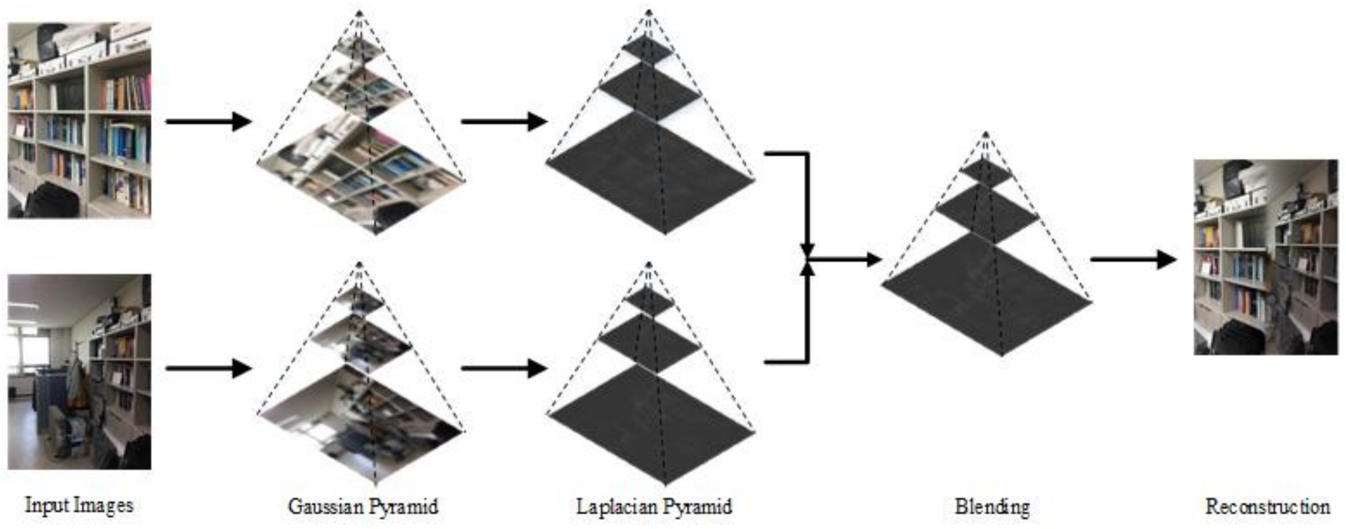

Figure 1. The steps of multi-band blending.

2. IMAGE BLENDING

2.1 Na ïve Image Blending

Image blending is a process of merging multiple images to form one natural image seamlessly. A blending scheme called "average blending" was used because of blending speed and simplicity. The average blending makes output pixels using the average pixel value that is computed between images that follow seams. Unfortunately, it would make the blended image blurry or leave boundary lines. Therefore, a new blending method called "multiband blending" has been employed commonly, and the details will be addressed in the following subsection.

2.2 Multiband Image Blending

The multiband blending is capable of blending low frequency components of the images while preserving the details of high frequencies. Thereby, it smoothens an image in the low-frequency band and sharpens an image in the high-frequency band.

It consists of several steps as shown in Figure 1 [1, 2]. To make a blended image, steps such as building image pyramids, blending computation and reconstruction should be carried out. Because many steps are computationally intensive, it is crucial to optimize the implementation in terms of performance, accuracy, and power consumption in each step.

The first step in image blending is building image pyramids of all images. This step takes a considerably large amount of execution time. It is mainly composed of two parts, a low-pass pyramid and a band-pass pyramid. A low-pass pyramid is to make an image using a smoothing filter and then to carry out subsampling by a factor of 2. Every cycle of this process results in a smoothing-filtered image of the original image. The size of the output image is a quarter of that of the original image. A band-pass pyramid is to make an image using the difference between images of adjacent levels in the pyramid and then to carry out the interpolation to raise the resolution to the next level. In this paper, we adopted the Gaussian pyramid as the low-pass pyramid that has a weight 5-by-5 Gaussian filter and the Laplacian pyramid as the band-pass pyramid.

The second step in image blending is to compute an output pixel and then to map the computed pixels to an output image. In this step, the differences between Laplacian pyramids of each image are computed. Output pixels are computed using the difference of Laplacian pyramids multiplied by a mask. Then, we used accumulation pixels of adjacent levels in the pyramid while interpolating between a high-level image and a low-level one in the pyramid.

3. GPGPU AND CUDA

A GPU is a specialized device for graphic processing such as creating output images to display. It typically specializes in processing only for computer graphics. However, a GPU device has evolved to a more flexible architecture so that it can carry out not only graphics processing but also computations in applications that are conventionally handled by the CPU. Commonly, such GPU is called as GPGPU. GPGPU consists of many multithreaded SIMD processors called scalar processor (SP). SPs have many lanes per processor. GPU generates a huge number of threads that can be executed in parallel so that a significant speed-up should be achieved as long as a large number of cores are efficiently utilized [4].

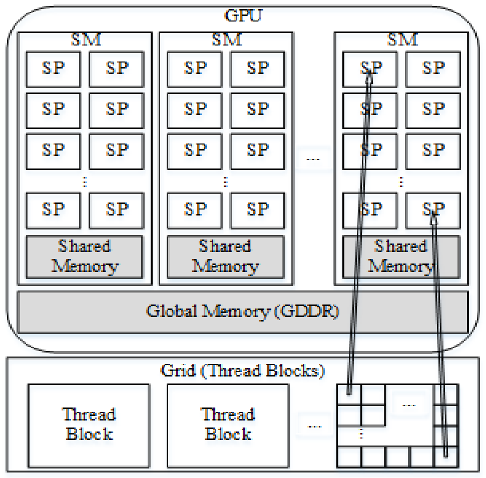

Figure 2. GPGPU architecture and CUDA computing strategy.

CUDA is a parallel computing platform and an application programming interface (API) model to support software development for GPGPU. It comes with software development environment that allows developers to use C as a high-level programming language. In addition, executing a large number of parallel threads implies that a large amount of data would be needed. Correspondingly, it is very important how the memory system is organized in GPU. There are several memory hierarchies such as global, shared, and constant memory as shown in Figure 2. The global memory is the memory known as GDDR. The shared memory is slow, but it can be shared among all threads in the same block that is typically called as streaming multiprocessor (SM). Using the shared memory would be advantageous in terms of performance, because it reduces memory accesses to the global memory. In this paper, we focus on using the shared memory to improve performance in image blending [4, 6].

4. PADDED IMAGE BLENDING BASED ON GPU

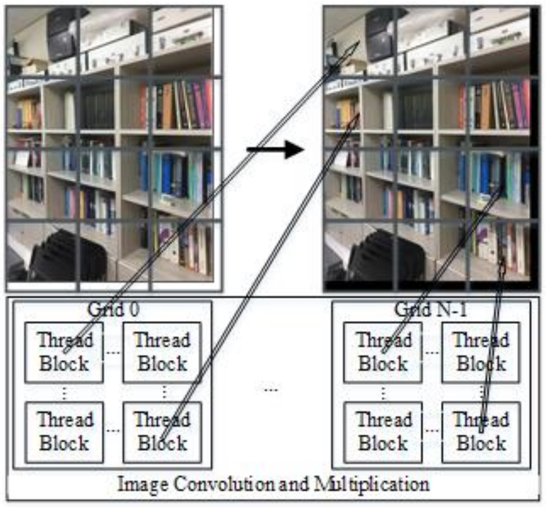

Figure 3. Zero-padding for image blending with shared memory.

Building Gaussian and Laplacian pyramids, computing differences of the Laplacian pyramid, and reconstruction are matrix convolutions. The basic approach is to compute those operations in parallel on GPU using CUDA. Clearly, CUDA implementation shows a large improvement, compared to implementation only on CPU.

Using the shared memory would lead to further performance improvement. However, there is a limitation to use the shared memory when images have variable resolutions. To utilize the shared memory option effectively, the thread block size and the memory size should be fixed regardless of the image size. For images with variable resolutions, therefore, additional method to resolve this concern would be needed.

In order to resolve this issue, we propose a novel image blending method called "padded image blending" where sides of the input image are selectively padded with zeros to make the size of input images to be uniform. The uniform size is determined to be a number that corresponds to a multiple of the maximum thread block size. Zero-padding performs no computationally meaningful work because any computations with the edge that is padded do not influence on the output image. However, by inserting the padding, it is possible to make the thread block and the shared memory size to be equal regardless of the size of input images. Thus, image blending can be parallelized efficiently using the shared memory on GPU with the same number of threads. The proposed method is shown in Figure 3. In this proposed method, input images are divided into the equally sized blocks and then they are mapped to the shared memory.

5. EXPERIMENT AND DISCUSSION

5.1 Experimental Setup

The target platform consists of i7-3770K CPU and GeForce GTX 1060 GPU. CPU and GPU are externally linked through the PCI Express (PCIe) bus. Intel i7-3770K is a quad-core CPU and NVIDIA GeForce GTX 1060 consists of 1152 cores that are composed of 9 SMs by 128 SPs with a 3GB GDDR memory. The host program was compiled using Microsoft Visual Studio 2015 and CUDA 8.0 library. The detailed platform specification is summarized in Table 1.

To evaluate the performance of the proposed method, two input images with 2448*3264 high-resolution shown in Figure 4 and Figure 5 were used.

Table 1. The specification of target platform

Processor Type	CPU	GPU
Processor Architecture	Intel i7-3770K	NVIDIA GeForce GTX 1060 3GB
Processor Clock	3.5 GHz	1.71 GHz
# of Core	4	9 (SM) 128 (SP)
Performance (GFLOPS)	18	3,935
Memory Size	10 GB	3 GB
Floating Point	Single Precision (32bit)	
Operating System	Windows 10 Education 64bit	
Library	CUDA 8.0	

5.2 Experimental Results

Table 2 shows comparison results of the execution time between the CPU-only implementation and one with the GPU acceleration without the shard memory utilization, and Figure 4 shows the output image. Table 3 shows comparison results of the execution time between the CPU-only implementation and the GPU acceleration with the shared memory using CUDA after zero-padding. Figure 5 shows the output image.

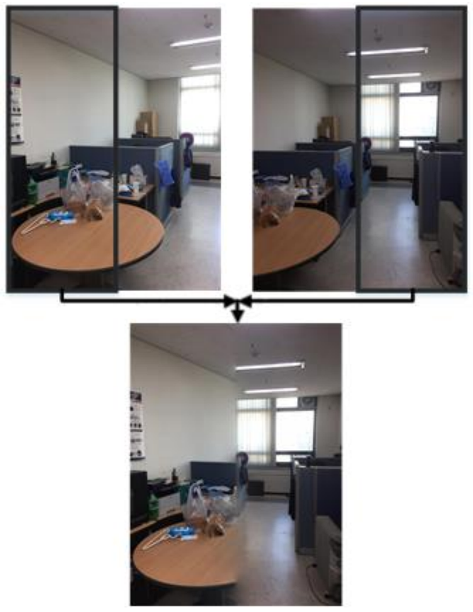

Figure 4. Blended image of CPU-only implementation and with the GPU acceleration w/o shared memory.

Table 2. The execution time of image blending using CPU and GPU (2448*3264, 16 images).

	CPU	GPU	speedup
Gaussian Pyramids	2.19	0.83	2.62
Laplacian Pyramids	4.82	1.66	2.90
Blending	0.52	0.51	1.01
Reconstruction	2.49	0.78	3.17
Total	10.04	3.80	2.64

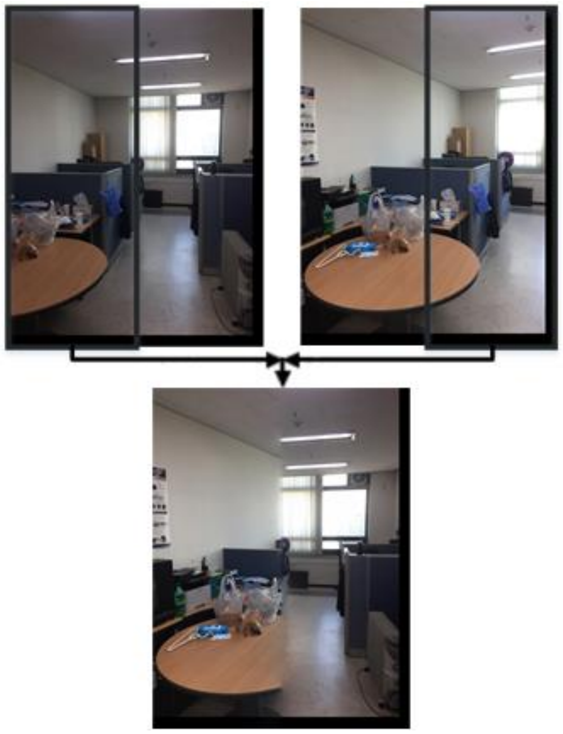

Figure 5. Blended image of CPU-only implementation and with the GPU acceleration with shared memory after zero-padding.

Table 3. The execution time of image blending using CPU and GPU with the shared memory (2448*3264, 16 images).

	CPU	GPU w/shared	speedup
Gaussian Pyramids	2.19	0.54	4.05
Laplacian Pyramids	4.82	1.01	4.76
Blending	0.52	0.52	1.00
Reconstruction	2.49	0.48	5.16
Total	10.04	2.56	3.92

As shown in Table 2, the execution time of image blending of the CPU-only implementation is 10.04 sec while that of the GPU execution is 3.80 sec, which means the execution time is improved by more than twice.

As shown in Table 3, the execution time of image blending is 2.56 sec using GPU with the utilization of the shared memory. The execution time is about four times faster than the CPU-only implementation. Also, it is faster than the GPU implementation without the shared memory. With the shared memory, threads that belong to the same thread block have accesses to the shared memory instead of the global memory. Thus, reduction of memory accesses to the global memory has led to the performance improvement.

In both experiments, the implementation using GPU is clearly advantageous to image blending. Also, the implementation using GPU with the shared memory option achieves a remarkable performance improvement.

6. CONCLUSION

Demands for high-resolution image processing techniques in computer vision grow very rapidly as high-resolution devices such as camera sensors become widely available. As the resolution grows, the amount of computation in image processing rapidly increases as well. In this paper, we have presented a novel image blending method with GPU acceleration. We proposed a new technique with zero-padding in order to utilize the shared memory on GPU efficiently. The experimental results show that the execution time of the proposed implementation is faster by 3.9 times than the CPU-only implementation. Thus, we conclude that the implementation using GPU is very effective for image blending.

7. ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (R7119-16-1009, Development of Intelligent Semiconductor Core Technologies for IoT Devices based on Harvest Energy).

8. REFERENCES

[1] Peter J. Burt and Edward H. Adelson. 1983. A multiresolution spline with application to image mosaics. ACM Trans. Graph. 2, 4 (October 1983), 217-236. DOI=http://dx.doi.org/10.1145/245.247

[2] Matthew Brown and David G. Lowe. 2007. Automatic Panoramic Image Stitching using Invariant Features. Int. J. Comput. Vision 74, 1 (August 2007), 59-73. DOI=http://dx.doi.org/10.1007/s11263-006-0002-3

[3] M. Brown and D. G. Lowe. 2003. Recognising Panoramas. In Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2 (ICCV '03), Vol. 2. IEEE Computer Society, Washington, DC, USA, 1218-.

[4] David B. Kirk, Wen-mei W. Hwu. 2016. Programming massively parallel processors: a hands-on approach (3rd. ed.). Morgan Kaufmann.

[5] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (PPoPP '08). ACM, New York, NY, USA, 73-82. DOI=http://dx.doi.org/10.1145/1345206.1345220

[6] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable Parallel Programming with CUDA. Queue 6, 2 (March 2008), 40-53. DOI: https://doi.org/10.1145/1365490.1365500