Phase Map Generation for Phase Shift Moire using CUDA

Sudhakar Sah  
Center for Research in Engr Sciences and Tech-CREST  
KPIT Cummins Infosystems Ltd.  
Pune, India  
e-mail: Sudhakar.sah@kpitcummins.com

YoungJun Roh, KyoungSeop Chang and DaeHwa Jeong  
FPG, Productivity Research Institute  
LG Electronics Inc., Korea  
e-mail: {youngjun.roh, kyoungseop.chang, daehwa.jeong}@lge.com

Abstract—Phase shift Moiré is a very popular and one of the most successful techniques for shape measurement of 3-D objects such as PCB (printed circuit board), TFT (thin film transistor), LCD (liquid crystal display) etc. Various implementations of phase shift moiré are available for improving accuracy and/or speed. Although, these methods contribute a lot in reducing the computation with some compromise in accuracy, there is a lot of scope of improving the performance of these algorithms with increased accuracy, especially when specialized hardware like GPU is available. GPU contains many core or processing elements that can process the same work concurrently resulting in dramatic increase in performance. In this paper, we propose the parallel implementation of the phase shift moiré method on CUDA. A novel method called image stacking method is proposed that can also be used for CUDA implementation of similar algorithms to improve performance. Using this technique, we are able to execute the application 180 times faster compared to the CPU implementation.

Keywords-CUDA; GPU; Image Processing; parallelization; Phase shift Moire; Speed up

I. INTRODUCTION

With the increasing need of processing power, various processor architectures have been proposed by chip manufacturers. GPU (Graphics Processing Unit) is one such advancement in processor technology. Earlier generation GPUs were used primarily for gaming applications and image rasterization for high quality graphics giving realistic visual effects. The earlier limitations of GPU usage for general purpose computation were mainly due to the need of specific graphics APIs. NVIDIA’s Compute Unified Device Architecture (CUDA) provides simple C like APIs that can be used for programming GPUs easily for general purpose computation (apart from the graphic processing applications). Current GPUs are available with 512 cores (NVIDIA Tesla architecture) that provide massive processing power to exploit data parallelism in applications.

Phase shift Moiré (PSM) method is used to measure the 3D shape or surface profile of an object [9]. There are different methods proposed for 3D object measurement and real time performance of these algorithms is essential for industrial applications. In order to get high accuracy from these experiments, large number of phase shifted images is processed simultaneously. The computational logic is simple but the computation requirement comes from the huge amount of data that is required to be processed. There are various modifications of these algorithms for increasing the speed of execution but those methods compromise either accuracy or are too complex to implement. In this paper, we are presenting one of the simplest phase shift moiré method implementation on GPU. We target to achieve the real time performance for this algorithm (50 ms) that is used for finding the shape of an object. We are introducing a concept called 3D stacking of data to enable processing of large data set with same functionality efficiently. The results are compared with pipelining implementation and also with the CPU implementation. We have reported maximum speed up of 180 times (as) compared to the CPU implementation and 3 times as compared to other CUDA based implementation of similar algorithm.

The paper is organized as follows. Section 2 describes the related work done in this area and some of the prominent work on PSM [9] which is referred in this paper. Section 3 gives a brief introduction of the PSM method. Section 4 describes the CUDA framework in detail along with the hardware details. Section 5 analyzes the results and concludes with some best practices to be used on CUDA.

II. RELATED WORK

CUDA is a new technology offered by NVIDIA that includes the changes in software and hardware framework with aim to reduce the programming complexity and improve the parallel performance. Hawick et.al in [1] presented connected component analysis implementation for CUDA. There are different kernel functions derived for improving the performance of the algorithm. Rifat and Yuzhong [2] have given a realistic feel of capabilities of GPU. Yuancheng and Duraiswami [3] have presented their work on the Canny edge detection on CUDA. They have reported up to 4 times increase in the performance compared to OpenCV [7]. Nadathur Satish, Mark Harris and Michael Garland [8] have reported a new design for porting algorithm on GPU. There is no paper available on CUDA implementation of PSM. This paper presents the PSM algorithm implementation for CUDA in detail and it gives comparative analysis of the same algorithm on various platforms (because of lack of results available in literature). It also gives analysis of the
performance improvement and degradation due to changes in various parameters during CUDA programming.

III. PHASE SHIFT MOIRE

Phase Shift Moiré (PSM) technique is used to measure the surface profile of 3D objects. It is applicable in surface flatness measurement for applications like LCD, TFT etc. Fig. 1 and 2 shows the experimental set up required for the PSM method. A light source is projected on to the subject (reference and object) to create fringe patterns on the surface of subject plane. Two types of fringe patterns can be obtained, first using the reference plane and second using the object plane (test object). Depth of the object surface can be found by analyzing the object fringe pattern (formed) because of the presence of discontinuities in the modulated fringe pattern of object under inspection compared to the reference fringe pattern. Phase shift can be applied to the fringe by the movement of grating. This phase shift can vary between 0 to 2\(\pi\). Following equation gives the intensity \(I_r\) and \(I_m\) (in equation (1) and (2)) of the reference and modulated image respectively for location (x, y) [9].

\[
I_r = a_1 + b_1 \cos(\alpha + \varphi) \tag{1}
\]
\[
I_m = a_1 + b_1 \cos(\alpha + \varphi) \tag{2}
\]

Where \(\alpha\) is the phase shift at which intensity is computed and \(\varphi\) is the phase at the location (x, y). Figure 3 shows four images captured at phase shift of 0, 90, 180, and 270 degrees respectively.

To get the information about surface profile, phase values at each point in the image needs to be calculated. We have used the maxima of intensities method to calculate the phase value at each pixel of the image.

There are 20 images captured at the progressive phase difference of 18 degrees. Figure 4 shows the intensity variation at one of the pixels for these 20 images. The sinusoidal signal in this figure is taken just for reference. In actual scenario, it is not an exact sinusoidal and can vary, but the signal will still be periodic. Following are the steps to calculate the phase values at each pixel using maxima of intensity method.

A. Phase computation for each image

\[
p = \text{maxima of intensity for 20 images}
\]
\[
P_{\text{est}} - \text{Peak estimation}
\]
\[
h_1\text{ and } h_{+1} \text{ left and right neighborhood of } p \text{ respectively.}
\]
\[
N - \text{Number of images}
\]
\[
\text{Phase } (k) - \text{phase shift for } k^{\text{th}} \text{ image}
\]
\[
P_{\text{est}} = p + \frac{h_{-1} - h_{+1}}{h_{-1} + h_{+1}} \tag{3}
\]
\[
\text{Phase } (k) = (360 + (k - P_{\text{est}}) \frac{360}{N}) \% 360 \tag{4}
\]
**B. Phase Map**

\[ P_m(x,y) = \text{Phase at pixel } (x,y) \]

\[ P_m(x,y) = \sum_{k=1}^{N} \max(P(k)) \]

In this paper, a maximum of 20 images are taken with a phase shift of 18 degrees. PSM method is used to find the shape of 3D objects and it is very useful in computer vision applications. This system receives 20 images after every 50 ms and hence, phase map generation step is expected to finish in 50ms. Since this method is used to automatically detect the defects in objects in production line, real time operation is a must. Otherwise, it will affect the overall production efficiency. The image preprocessing and phase map generation algorithm is completely data parallel. GPGPU is best suited for data parallel algorithms; hence we have used CUDA as development platform to get real time performance.

**IV. NVIDIA CUDA**

GPU (Graphics processing unit) were earlier used to get high quality graphics in gaming applications. In 2007, NVIDIA introduced a new architecture called CUDA (Compute Unified Device Architecture) to enable GPU usage for general purpose computation. CUDA provided top level APIs to make program development for GPGPUs easier in order to utilize massive parallelization powers of the GPU. Fig 5 shows the hardware architecture of GPGPU [5]. The architecture consists of array of multiprocessor called streaming multiprocessor (SM), each having its own shared memory and stream processors (SP). NVIDIA GTX 480 has 15 SM and each SM contains 32 SPs. The code is dispatched from CPU to the thread execution manager which schedules these threads to cores. Hence, user is freed from the burden of writing code and scheduling it for load balancing. Each core can run multiple threads at the same time and hence can produce exceptional speed up required for high performance computation. Every SM can access large chunk of memory called global memory. Global memory size is huge but its performance is low compared to shared memory. Hence the memory management affects the GPU implementation throughput heavily. An application written using CUDA can be seen as a host program which runs on CPU in addition to a 'kernel' which is run by multiple threads at the same time on different cores of GPGPU. Threads are executed in groups, which are called as 'blocks'. Grid is the collection of blocks to be executed on the device. The kernel code is executed by multiple threads at the same time.

CUDA architecture provides different types of memories like global, shared, texture and constant. Performance of the CUDA code largely depends on how well the architecture of the GPU has been exploited. Some of these techniques will be discussed in the implementation section.

**V. IMPLEMENTATION**

Figure 6 shows the block diagram for the process of phase map computation. System has N input images and after processing, it produces a single image called the phase image. In the first stage, all the images are filtered through the averaging mask (low pass filtering). Output of first stage is subjected through some algebraic operations (computationally intensive) and a value for each pixel is selected from N images based on the algebraic operation. Output of the algebraic function is the phase value for each pixel (Calculated using equation 2).

![GPGPU Architecture](image)

**Figure 5. GPGPU Architecture**

It is evident that this process is computationally intensive and it needs faster processing for real time performance. Since the algorithm has inherent data level parallelism (DLP), it becomes a natural candidate of parallelization on CUDA. In current setup, processing 20 images (1024x1024 pixels) is expected to complete in 50 ms so that the system is ready to process next set of images. Average filter operation on CUDA is trivial and doesn’t require any special parallel design.

**A. Shared memory implementation of average filter**

To compute the average filter of single image, it is divided into grids and blocks and average for each pixel is
calculated on separate thread as shown in figure 7. In order to increase the speed further, concept of tiles is used where data required for one tile (smaller than the block) is copied to the shared memory. The reason of copying the values to shared memory is that for a 5x5 kernel, total 25 pixels are used and out of total 25, 20 pixels are reused for next pixel. Shared memory is shared between threads inside a block and hence the data reuse optimized the performance considerably. This method reduces access of global memory considerably. Downside of this method is that some of threads do not participate in computation and they are only used once for copying data to the shared memory.

![Figure 7. Pipelined computation of average filter of multiple images](image)

### B. Pipelining method

For our experiment, first we used shared memory implementation for averaging filter and subjected all the images for filtering operation sequentially (concurrent pixel execution and sequential image execution). This means each image is processed in different kernel but pixels of each images were executed concurrently. The technique resulted to a very high speed up but the sequential operation of each image became bottleneck in achieving the real time performance target. This technique is termed as pipelining method because the images are kept in pipeline for execution but parallel execution of each pixel exploits the GPGPU architecture.

### C. 3D kernel method

To process all the images concurrently, another option provided by CUDA is to use 3D (three dimensional) kernel. In 3D kernel, third dimension of data type is provided and accordingly filter output is computed. So, number of images will form third dimension for 3D kernel. This is one of the possible solutions, but there is a catch here. We are allocating threads from same pool to perform computation of other images in 3D. Hence, speed up is possible only by increasing the total number of threads which unfortunately is not possible because of hardware limitations. Hence, this method didn’t contribute to further speed up. Although, it saves lot of overhead of separate kernel execution in the pipelining method, this effort is still insufficient to achieve the target performance. Hence, it requires a method which can keep number of threads to be same as the pipelining method but can process all the images concurrently. GPU has got powerful processing cores specially designed to perform logical and arithmetic computation. In order to exploit massive processing power provided by GPU, each thread should be sufficiently loaded with heavy computation. During calculation of average filter for single image, number of arithmetic and logical calculation done by each core is less than 100-150. So, each thread is theoretically able to perform much more operation than it is handling in this case. 3D stacking technique is used take care of these problems.

### D. 3D Stacking method

In 3D stacking method, the block size, tile size and grid size is same as individual kernel in pipelining method, but data for all the images is passed to one kernel. Images are stored in a single array and copied to the device once and passed to the kernel. Images are sequentially stored in pointer so that after the end of first image data, next image data starts. Although, kernel parameters are passed such that the data has to be processed for only one image (grid and block size same as size of one image), passing all the images to the kernel enables processing every pixel of all the images by a thread. Each pixel of the image can be processed independently by separate thread. But, due to low computational requirement by each core, each thread can be given more tasks to optimize the power of each core. Figure 8 shows the 3D stacking method where each thread is calculating the average value for one pixel for all the images. Advantage of this method over pipelining method is that only one kernel is used for processing. Also, by increasing the computation done by each thread, overall time for computation by each pixel for all the images is approximately same in case of pipelining method for single image.

### E. Algebraic function implementation

Algebraic function processing uses the averaged filtered output from the first step as input and calculates the phase image. Majority of functions to be performed in this block are functions related to the data analysis like max, min, sort, average. This step is computationally intensive for each pixel. The implementation of algebraic function is done in similar way using 3D stacking except one minor difference. For computation of each pixel of the output image, data for nearby pixels are not required. Hence, there is no need to tile the data for using shared memory. This allows us to increase the number of threads executing...
acordingly and to make sure that none of the threads are sitting idle.

Figure 8. Average filter computation using 3D Stacking

VI. RESULTS

The implementation of Phase shift algorithm is done on NVIDIA GTX 480. The host CPU used for this experiment is the Intel dual core processor with 2 GB of RAM and clock speed 1.8 GHz. NVIDIA GTX 480 core has clock frequency of 1.4 GHz.

TABLE I. PERFORMANCE COMPARISON

<table>
<thead>
<tr>
<th>Image size</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask size</td>
<td></td>
</tr>
<tr>
<td>GPU (ms)</td>
<td>CPU (ms)</td>
</tr>
<tr>
<td>Speed up</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>3x3</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
</tr>
<tr>
<td>1024</td>
<td>3x3</td>
</tr>
<tr>
<td></td>
<td>5x5</td>
</tr>
</tbody>
</table>

Figure 9. CPU Vs GPU performance

We have not found papers available in literature about implementation and performance improvement of the PSM algorithm on GPU. There are papers available for filter implementation on CUDA but it is not possible to benchmark our results with those because the challenge in our case is to process multiple images at a time and getting improvement for overall processing. Hence, we are comparing our results with C implementation on CPU and also with the optimized pipelined implementation. Table I shows that the minimum range of speed up achieved compared to the CPU implementation varies between 77 and 183 times for different image size and filter mask size. The speed up increases significantly with the increase in image size. Results are summarized with the help of bar graph in figure 9. It is visible from the graph that GPU implementation completely outperforms the CPU implementation.

CUDA implementation requires the data to be copied to device and after processing copying the result back to the CPU. At times, the memory copy speed is more than the actual processing time. So, we have also analyzed the processing time vs the copy operations time. As we are copying very huge amount of data to the CPU, the copy to and from GPU almost consumes 85% of the total execution time (as shown in table II) which is significant. Still, the technique is able to reduce the computational time immensely and hence the memory copy overhead can be ignored. This impact is depicted by a pie chart in figure 10. The pie chart suggests that the data copy to device takes 51%, processing takes 15% and copying the result to the CPU takes 34% of the total execution time.

Our final comparative analysis is with the pipelined GPU implementation. We have implemented the optimized average filter executed it for 20 images sequentially. The same process is repeated for algebraic function. It is evident from table III that pipelining method scores high when the number of images is small, as there are some constant overheads for 3D stacking method. 3D stacking method dominates in terms of performance as number of images starts increasing. The reason is proportional increase in the sequential operation with increase in the number of images. We can see from the graph in figure 11 that the performance for pipelining method is linear with high slope, where as 3D
Stacking is linear with very small slope (approximately 20 degrees).

### TABLE III.

<table>
<thead>
<tr>
<th>Image size</th>
<th>Number of Images</th>
<th>Execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1024</td>
<td>ST</td>
<td>PL</td>
</tr>
<tr>
<td>512</td>
<td>3.46</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 12 shows the input image and resultant output phase. The image is for one of LG’s mobile phone surface flatness testing. The resultant phase map contains lot of information about the object shape. We are not discussing the object shape measurement method here as it is out of the scope of this paper work.

V. CONCLUSION

The implementation of phase shift moiré algorithm on GPU is discussed in detail. This algorithm suffers from the performance problems because of large data processing in small time frame. GPU based image processing implementation provides great speed up but current algorithm has a peculiar requirement of doing similar operations on each image and then on each pixel. Implementation of pipelining method is also discussed along with its limitation for PSM algorithm. Finally, we discussed our approach of 3D stacking of image for faster and concurrent processing of multiple images when same operation is to be performed on multiple images. Maximum speed up of 180 x compared to the CPU implementation and 3x compared to pipelined implementation is reported. Hence GPU along with the CUDA framework has a great prospect for applications with data level parallelism and the results achieved are not possible to replicate for other hardware of similar cost and complexity.

ACKNOWLEDGMENT

Our sincere thanks to the member of Crest team in KPIT, India and LG-PRI team in Korea for their support to improve the quality of this paper. This work was sponsored by LG-PRI, Korea.

REFERENCES


