GPU Accelerated Real Time Rotation, Scale and Translation Invariant Image Registration Method

Sudhakar Sah[1], Jan Vanek[2], YoungJun Roh[3], and Ratul Wasnik[1]

[1] CREST, KPIT Cummins Infosystems Ltd. Pune, India
[2] Department of Cybernetics , West Bohemia University, Czech Republic
[3] Productivity Research Institute, LG Electronics Inc., South Korea
{sudhakar.sah,ratul.wasnik}@kpitcummins.com,
vanekyj@kky.zcu.cz, youngjun.roh@lge.com

Abstract. This paper presents highly optimized implementation of image registration method that is invariant to rotation scale and translation. Image registration method using FFT works with comparable accuracy as similar methods proposed in the literature, but practical applications seldom use this technique because of high computational requirement. However, this method is highly parallelizable and offloading it to the commodity graphics cards increases its performance drastically. We are proposing the parallel implementation of FFT based registration method on CUDA and OpenCL. Performance analysis of this implementation suggests that the parallel version can be used for real time image registration even for image size up to 2k x 2k. We have achieved significant speed up of up to 345x on NVIDIA GTX 580 using CUDA and up to 116x on AMD HD 6950 using OpenCL. Comparison of our implementation with other GPU based registration method reveals that our implementation performs better compared to other implementations.

Keywords: GPU, Image Registration, CUDA, OpenCL, Object Recognition.

1 Introduction

Image registration (IR) is defined as detection of the presence of complete or partial image (test image) in a reference image. Image registration is useful in applications like, motion tracking, machine vision, image restoration etc. This paper uses the method of image registration based on spectral components (FFT). FFT based method uses the property of shift theorem according to which, phase difference between two images gives the translation between two images. Phase correlation method was first developed by Kuglin and Hines [1] by utilizing some properties of Fourier transform. Fourier transform is capable of determining shift between two images. So, DeCastro and Morandi [2] proposed a modified method to determine the rotation by using Fourier transform. The method was improved by Reddy and Chatterji [3] for performance and accuracy which was further clarified mathematically by Sierra et al. [4]. Recently, Rittavee Matungka (2009) details the log polar

A. Campilho and M. Kamel (Eds.): ICIAR 2012, Part I, LNCS 7324, pp. 224–233, 2012.

transformation [5] based image registration method used to detect the rotation and scale of image using FFT technique.

The image registration method is computationally intensive algorithm, which hinders its usage in real time applications. Many techniques had been employed to increase the speed [6] [7] but they suffered from low accuracy or found applicable to limited applications. The state of art technique by David Lowe [8] called SIFT (Scale Invariant Feature Transform) gives accurate template matching results with reduced complexity. Still, these methods need significant improvement in the speed for real time analysis of video and images.

Performance improvement of these algorithms is of high importance for general acceptance. FFT based method is inherently parallel and GPGPU (General Purpose Graphical Processing Units) with massively parallel processing cores can be utilized for acceleration of FFT based registration method. The scale and rotation resolution of output directly depends on the input resolution. High resolution requires more computation time. Our parallelization approach gives dual benefits, i.e. high accuracy and real time performance. This paper is organized as follows. Section 2 discussed the registration concept, section 3 explains GPGPU features, section 4 details the parallel algorithm implementation and section 5 discusses the implementation results.

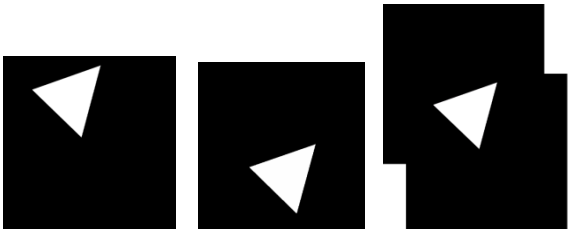


Fig. 1. Reference Image *(left),* Translated Image *(center)*, Overlap between Images *(right)*

2 Image Registration

2.1 Translation

Figure 1 shows two images I_1 and I_2 where I_2 is shifted by Δx columns and Δy rows as represented by equation (1). Let us represent the FFT of any signal by symbol ₣ as shown in equation (2). Applying Fourier transform on both sides of equation (1) gives equation (3). Translation in image can be calculated by inverse of Cross correlation of I_1 and I_2, which is given by equation (4).

$$I_2(x + \Delta x, y + \Delta y) = I_1 \tag{1}$$

$$₣(I(x, y)) = \hat{I}(\alpha, £) \tag{2}$$

$$\hat{I}_2(\alpha, £) = \hat{I}_2(\alpha, £)e^{(i2\Pi\alpha\Delta x + £\Delta y)} \tag{3}$$

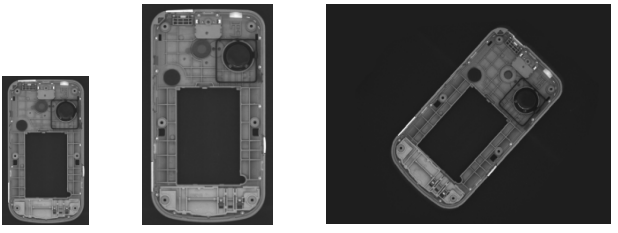


Fig. 2. Reference Image *(left),* Scaled Image *(center),* Scaled and Rotated image *(right)*

$$\frac{\hat{I}_2(\alpha, £)\ \hat{I}_2(\alpha, £)}{|\hat{I}_2(\alpha,£)|\ |\hat{I}_2(\alpha, £)|} = e^{(i2\Pi\ \alpha\ \Delta x + £\Delta y)} \tag{4}$$

In theory, inverse of equation (4) should give impulse function indicating the translation in x and y direction but in presence of noise practical approach is to find the value of Δx and Δy for maximum value of cross correlation.

2.2 Rotation

Suppose I_1 and I_2 are two images and I_2 is rotated by angle φ with respect to I_1. Rotation relationship between I_1 and I_2 is given by equation (5). Applying Fourier transform on both sides,

$$I_2(x, y) = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} I_1(x - \Delta x, y - \Delta y) \tag{5}$$

$$\hat{I}_2(\alpha, £) = e^{(i2\Pi\ \alpha\Delta x + £\Delta y)}\ \hat{I}_1(\alpha\cos(\varphi) + £\sin(\varphi), -\alpha\sin(\varphi) - £\cos(\varphi)) \tag{6}$$

Where,

$$A_2 = A_1(\alpha\cos(\varphi) + £\sin(\varphi), -\alpha\sin(\varphi) - £\cos(\varphi)) \tag{7}$$

It is evident from equation (7) that magnitude of I_1 and I_2 are same except that the spectrum of I_1 is rotated by angle φ with respect to magnitude of I_2. In order to get the rotation angle, amplitude of I_1 and I_2 should be converted to polar co-ordinates. This converts the rotational shift to translation which is shown in equation (8). Phase correlation of polar domain can be used to get the rotation of images.

$$A_2 = A_1(\varphi - \varphi_0) \tag{8}$$

2.3 Scaling

I_1 and I_2 are two images where I_2 scaled version of I_1 and the scaling factor is μ. This can be represented in frequency domain by equation (9). Scaling in frequency domain can be found by converting the Fourier transform values to logarithmic scales and computing phase correlation of the same. It is shown by equation (10).

$$\hat{I}_2(\alpha, £) = \hat{I}_1(\alpha/\mu, £/\mu) \tag{9}$$

$$\hat{I}_2(\log(\alpha),\log(£)) = \hat{I}_1(\log(\alpha) - \log(\mu), \log(£) - \log(\mu)) \tag{10}$$

2.4 Scale, Rotation, and Translation

Using above concepts, it is clear that if image I_2 is scaled, rotated and translated by μ, φ and Δx, Δy with respect to I_1 then it can be represented in Log-polar frequency domain by equation (11).

$$A_2(\log(\alpha), \varphi) = A_1(\log(\varphi) - \log(\mu), (\varphi - \varphi_0)) \tag{11}$$

By computing the maxima of phase correlation of equation (11), phase angle and scaling can be found out. In second step, reverse transformations (scale and rotation) is applied to the test image and phase correlation is applied to the transformed image to find the values of Δx and Δy.

3 GPGPU

The tasks suitable for processing on GPU are characterized by high parallelism, low dependency between individual work elements and rather numerical character with minimal branching. Such tasks are commonly known as data-parallel algorithms. Due to the distinctive characteristics of GPU architecture (high speed, high latency main memory, limited caching capabilities, limited communication with CPU, minimal thread switching and planning overhead), the common CPU programming models (and programming languages based on this programming models) are not suitable for GPU programming. In order to achieve close-to-peak performance, the programmer must consider many low level specifics of the given target architecture and therefore, the programming model as well as the programming language must support explicit expression of programmer's intentions. NVIDIA's CUDA (Compute Unified Device Architecture) has gained a wide acceptance, however the CUDA standard is proprietary and the intellectual property concerns led to development of an open standard OpenCL (Open Computing Language).

The OpenCL standard was developed in cooperation with teams from ATI/AMD, IBM, Intel, NVIDIA and others and many HW vendors and SW producers announced support of OpenCL in their products. Unfortunately, this wide acceptance does not dislodge the burden of hand-tuning the computational kernels for individual distinct HW architectures. The NVIDIA GPU consists of many processing elements (PEs) called multiprocessors. Older NVIDIA GPUs have 8 stream processors in each PE together with 16kB on-chip local memory (called shared memory). Single PE offers 8k or 16k 32-bit registers, depending on GPU series. The Fermi-based PEs are bigger and contain 32 or 48 stream processors. The 64kB on-chip memory can be set to two configurations: 16kB L1 cache and 48kB local memory or vice versa. The core clock (stream processors and local memory) is higher than the rest of the GPU. The peak

memory bandwidth from global memory can be achieved only via coalesced access, where consecutive 16 work-items (half-warp) access consecutive addresses [9]. The ATI/AMD GPUs are based on very long instruction word (VLIW) architecture. This architecture also consists of multiple PEs. However, compared to NVIDIA PEs, the internal architecture is different. Each PE contains 16 stream cores, each equipped with five stream processors (four in Cayman based 69XX GPUs). This is why ATI/AMD GPUs have a higher raw computational performance than comparable NVIDIA GPUs. In real life, however, it is difficult to supply the input data sufficiently fast to keep these high-performance multiprocessors fully utilized. Therefore, the maximal performance cannot be achieved in some tasks or badly optimized implementations.

4 Implementation Description

Figure 3 shows the steps to detect rotation and scale of input image with respect to the reference image. The registration process consists of two stages. First stage computes the rotation and scale. Second stage computes the translation using scale and rotation computed by first stage. This section discusses the parallel implementation of IR algorithm on GPGPU.

The first step of the first stage is computation of gradient image. We have used the Sobel gradient with 3x3 kernel for this purpose. Sobel gradient computation is a pixel level operation and hence it is suitable to the GPGPU. Output of each pixel is the weighted sum of surrounding pixels with Sobel operator. However, for computation of each output pixel, nine input pixel is read which is inefficient as global memory fetch is very slow. To increase the performance, we have proposed a novel kernel with following features:

- Each thread of the kernel process 4x2 pixel block and the threads process consecutive blocks. It ensures better data reusing and save some computing instructions.
- The input data are 8-bit pixels. Each thread fetches 4x4 pixel block from texture memory where four consecutive pixels are stored as single 32-bit integer. Two of the threads read extra two blocks on the boundaries.
- To gather all the pixels needed for processing, two four-pixel blocks are interchanged between threads using local memory. 32-bit data type is used to store/read four pixels at once. Now, each thread keeps 6x4 pixel block in registry.
- To save some computing instructions, vertical and horizontal differences are computed in advance. Than final gradient coefficients are computed.
- The results are stored via coalesced access to the global memory.
- This approach is more than 3-times faster than Sobel sample codes from CUDA SDK or AMD Accelerated Parallel Processing (APP) SDK.

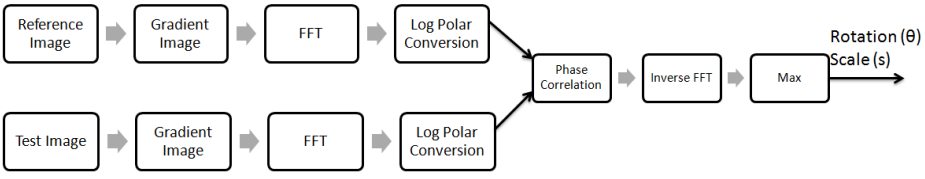


Fig. 3. Rotation and Scale detection

FFT of Sobel output for both the images is computed and the coordinate is shifted to the center. For this purpose, we have used the FFT library (cufft) provided by CUDA and FFT OpenCL library provided by AMD APP. In order to transform rotation and scale as translation, FFT is mapped to from the rectangular to log polar (RLP) coordinate. As there is no integer mapping between log polar and rectangular coordinate, bilinear interpolation method is used. Height of the image is mapped to 180 degrees in polar co ordinate using angular resolution $\Delta\theta$ as in equation (11). As the log conversion is non-linear, corresponding base selection (equation (12)) is required.

$$\Delta\theta = \frac{\mathrm{PI}}{\mathrm{H}}, \text{where H} - \text{Image height} \tag{11}$$

$$\mathrm{b} = \mathrm{pow}\left(10, \frac{\log(\mathrm{W})}{\mathrm{W}}\right) \text{where W} - \text{Image width} \tag{12}$$

$$\begin{aligned} \mathrm{x} &= \mathrm{r} * \cos(\Delta\theta) + \mathrm{x}_0 \\ \mathrm{y} &= \mathrm{r} * \sin(\Delta\theta) + \mathrm{y}_0 \end{aligned} \tag{13}$$

Where, x_0, y_0 – center of image

x, y – transformed co ordinate of image

Using base (b) and angle resolution ($\Delta\theta$) image is converted to the log polar with amplitude ranging from 0 to b^W and angle varying between 0 to 180 degrees as per equation (13). RLP implementation is straightforward on GPU as it is pixel-based operation. To optimize the performance further, texture fetch is used as bilinear interpolation uses four pixels which are selected by taking two integer x and two integer y values on the boundary of real values computed using equation (13).

Phase correlation of log polar values of reference and test image is calculated and the location maximum of inverse FFT gives values of rotation and scaling by applying equation (11) and (12). Phase correlation is computationally intensive operation on CPU but its pixel wise operation makes it suitable for optimization on GPU. Computation of maximum also consumes considerable time. In addition, it is not a pixel operation and involves dependency during computation. We have used a fast two-stage approach to compute the maximum of array together with its coordinates. The first stage is parallel as it searches for the individual maxima in partitioned data. The number of parts should be high enough to ensure full GPU occupancy. But too high number of parts decreases the performance. The second stage computes the final maximum and coordinates. Both the kernels/stages are based on a well-known parallel reduction scheme [10].

Second stage of IR algorithm involves translation detection by applying reverse transformation of image (rotation and scale obtained during stage 1). The scaling and

rotation is done by a simple kernel using texture memory where bilinear interpolation is enabled. Translation is obtained using transformed image instead of test image in stage 1 and repeating all the steps in stage1. Figure 4 shows the breakdown of tasks and overall GPU utilization for FFT based IR method.

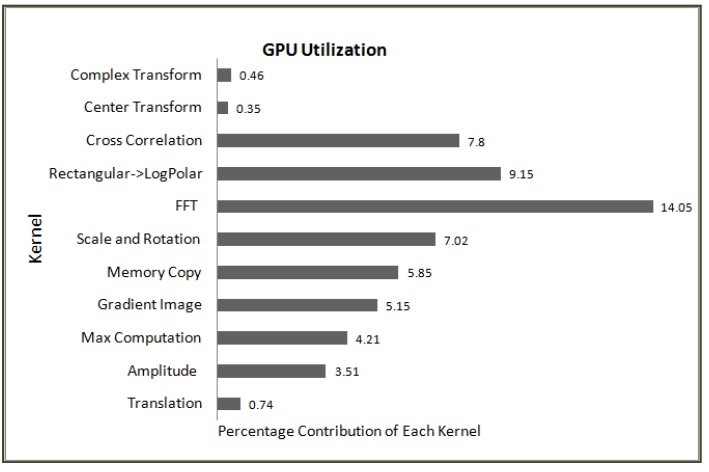


Fig. 4. GPU utilization of FFT based IR method

5 Results

We have implemented the IR method on different GPUs using CUDA and OpenCL. Table 1 shows the results on three GPUs. GTX 580 and GTX 260 are NVIDIA cards where CUDA was used. The last one is AMD Radeon HD 6950 with OpenCL. All the cards were running on Intel Core i7 host CPU with 3.2GHz clock frequency, 8MB cache and 4 GB RAM. We have got very high speed up compared to the CPU version. Our implementation achieves the maximum speed up of 345 times for image size 2k x 2k in GTX 580. The OpenCL on AMD GPU is significantly slower. There are a few reasons for the lower performance. AMD OpenCL implementation does not support fast memory transfers through PCI. These transfers take almost a half of total time for small images. The FFT kernels also take significantly more time. It seems that 2D FFT does not fit on AMD GPU architecture well or the AMD FFT library is not as optimal as the NVIDIA cuFFT library. Most of the other kernels are memory-intensive and high AMD GPU algorithmic performance cannot be utilized. Higher overhead of OpenCL API also degrades performance of simple and fast kernels.

We can see in graph of figure 5 that minimum number of frames per second (fps) processed is approximately 34 for image size 2048x2048 for GTX 580. Point worth mentioning here is that the size of test image does not affect the performance as we have resized the test image to reference image size first and then processed the entire image. Performance is less for HD 6950 and GTX 260 but it is still real time. Hence, the algorithm performs in real time for all the practical image sizes. Comparison in Table 1 assumes single threaded code with default compiler optimization on CPU.

Hence the speed up is very high. So, we have benchmarked our implementation with other implementations proposed in literature. Table 2 summarizes some of the methods that include the state of the art method proposed by David Lowe [8]. To make the comparison fair, we have mentioned the GPU configuration used in earlier papers. Table 2 also suggests that our implementation on GPU performs faster compared to all other methods. Figure 6 shows the output of image registration method and we have found that it is able to detect objects accurately with a resolution of 0.01. Only noticeable disadvantage of our method is that it does not have high accuracy for scale factors more than 2.5 as the features in image above these scale values are not recognized properly by this method.

Table 1. Performance comparison of CPU and GPU for FFT based Image Registration method

Image Size	Time (ms) CPU	Time(ms) / Speed up GTX 580, CUDA	Time(ms) / Speed up GTX 260, CUDA	Time(ms) / Speed up HD 6950, OpenCL
512 x 512	513	2.6 / 197x	4.9 / 105x	17.6 / 29x
1024 x 1024	2240	7.7 / 291x	18.3 / 122x	31.5 / 71x
2048 x 2048	9972	28.9 / 345x	86.3 / 116x	85.9 / 116x

Table 2. Performance comparison with other Image Registration methods

Method	Image Size	Time (ms)	GPU specification	Comments
(1) SIFT [11]	1k x 1k	160		GPU implementation
(2) SURF [11]	1k x 1k	70	GTX 260	GPU implementation
(3) Grayscale [6]	1k x 1k	100	GTX 260	Very low scale and rotation resolution
(4) ORB [12]	640x480	15.3	NA	Implemented for embedded processor
(5) Image Matching [13]	0.5k x 0.5k	6.8	GeForce 8800	GPU Implementation
	1k x 1k	19		
	2k x 2k	71		

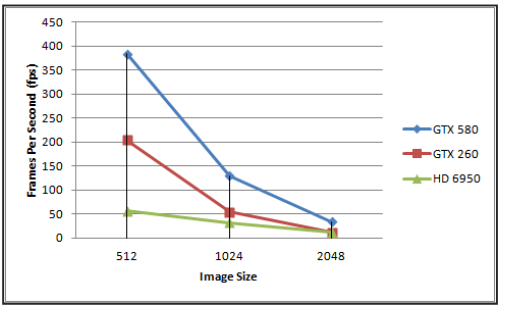
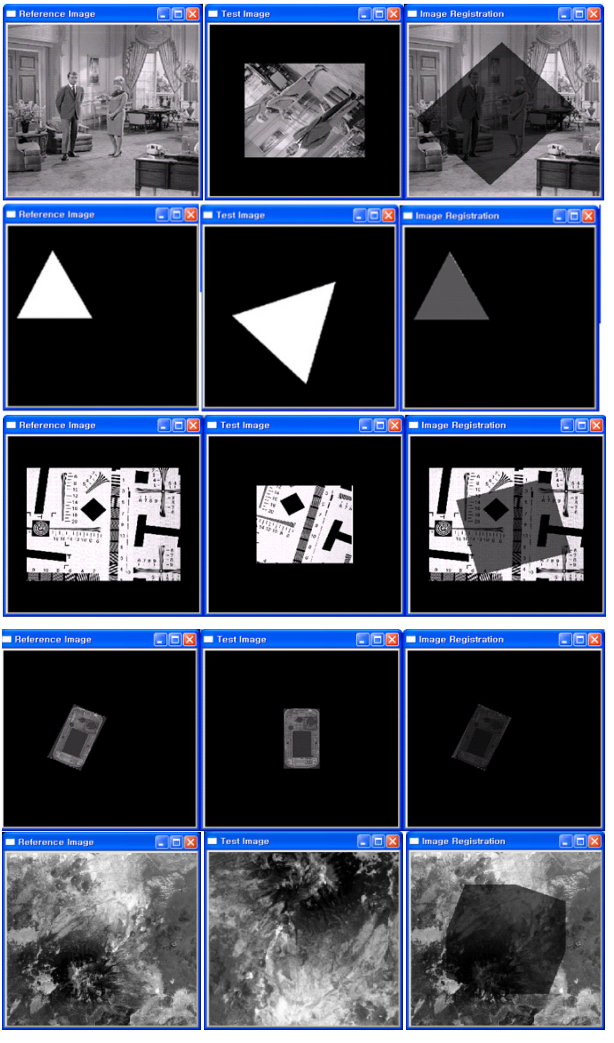


Fig. 5. Performance of IR method for different Image size, GPU and APIs



Living room
Rotation – 135^{O}
(Actual – 135)
Scale – 1
Actual- 1

Triangle
Rotation – 315
Actual – 315
Scale – 1.35
Actual- 1.35

Res chart
Rotation – 335
Actual – 335
Scale – 1
Actual- 1

Mobile
Rotation – 21
Actual – 21
Scale – 1.04
Actual- 1.04

Peaks
Rotation – 200
Actual – 200
Scale – 1.58
Actual- 1.60

Fig. 6. Reference Image *(left),* test image *(center),* Registered Image *(right)*

6 Conclusion

This paper presented the real time implementation of image registration method on GPGPU. We have implemented the method on graphics processors of different make to evaluate our results. We have found that the algorithm has real time performance for image size up to 2k x 2k when most of the registration methods fail to deliver the real time performance. We have not compromised in accuracy for this optimization even for larger image since angle and scale resolution is expected to be reduced to increase the performance. In our implementation, we have not reduced the angle and scale resolution but still able to get the real time performance. The high performance

of this algorithm is due to novel parallelization strategy and intelligent use of different types of GPU memory like texture and shared memory. We have achieved a maximum speed up of 345 times compared to the CPU version for CUDA on GTX 580 and 116 times on Radeon HD 6590 using OpenCL. We have also implemented the algorithm on GPU from other vendors than NVIDIA to make it usable for all graphics processors. The results show that it has comparable accuracy with other implementations for scale factor up to 2.5 and for all rotation angles. In addition, the resolution of angle and scale factor is as low as 0.01. Most of the image registration methods suffer from performance drawbacks. FFT based image registration has sufficient accuracy for machine vision problems and our parallel implementation on graphics processors makes it usable for real time applications.

Acknowledgement. This research was supported by the Grant Agency of the Czech Republic, project No. GAČR P103/12/G084.

References

1. Kuglin, C.D., Hines, D.C.: The Phase Correlation Image Alignment Method. In: Proceedings of the IEEE 1975 International Conference on Cybernetics and Society, New York, pp. 163–165 (1975)
2. DeCastro, E., Morandi, C.: Registration of Translated and Rotated Images Using Finite Fourier Transforms. IEEE Transactions on Pattern Analysis and Machine Intelligence, 700–703 (1995)
3. Reddy, B.S., Chatterji, B.N.: An FFT-based Technique For Translation, Rotation, and Scale-Invariant Image Registration. IEEE Transactions on Image Processing 5, 1266–1271 (1996)
4. Sierra, I.: Geometric foundations for the uniqueness of the FFT- based Image Mosaicking, With The Application to Detecting Hidden Text in Web Images. M.S. Thesis, University of Texas at El Paso, El Paso, TX (131)
5. Rittavee, M.: Studies on log-polar Transformation For Image Registration and Improvements using Adaptive Sampling and Logarithmic Spiral. PhD. Thesis, Graduate School of Ohio State University, 5–40 (2009)
6. Kim, H.Y., de Araújo, S.A.: Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast. In: Mery, D., Rueda, L. (eds.) PSIVT 2007. LNCS, vol. 4872, pp. 100–113. Springer, Heidelberg (2007)
7. Ghafoor, A., Iqbal, R.N., Khan, S.: Robust Image Matching Algorithm. Video/Image Processing and Multimedia Communications, 155–160 (2003)
8. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 91–110 (2004)
9. The CUDA C best practices guide, version 3.2. NVIDIA Corporation (August 2010), `http://developer.download.nvidia.com/compute/cuda/32prod/toolkit/docs/CUDACBestPracticesGuide.pdf`
10. Kirk, D.B., Hwu, W.W.: Programming massively parallel processors: A hands-on approach. Morgan Kaufmann, San Francisco (2010)
11. Martinez, M., Collet, A., Srinivasa, S.S.: MOPED: A scalable and low latency object recognition and pose estimation system. In: Robotics and Automation (ICRA), pp. 2043–2049 (2010)
12. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: An Efficient Alternative to SIFT or SURF. In: International Conference on Computer Vision (2011)
13. Chariot, A., Keriven, R.: GPU Boosted Online Image Matching. In: 19th International Conference on Pattern Recognition, pp. 1–4 (2008)