university of groningen

University Medical Center Groningen

University of Groningen

GPU-ASIFT

Codreanu, Valeriu; Dong, Feng; Liu, Baoquan; Roerdink, Jos B.T.M.; Williams, David; Yang, Po; Yasar, Burhan

Published in:
EPRINTS-BOOK-TITLE

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2013

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Codreanu, V., Dong, F., Liu, B., Roerdink, J. B. T. M., Williams, D., Yang, P., & Yasar, B. (2013). GPU-ASIFT: A Fast Fully Affine-Invariant Feature Extraction Algorithm. In *EPRINTS-BOOK-TITLE* University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science.

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 04-06-2022

GPU-ASIFT: A Fast Fully Affine-Invariant Feature Extraction Algorithm

Valeriu Codreanu*, Feng Dong†, Baoquan Liu†, Jos B.T.M. Roerdink*, David Williams*, Po Yang† and Burhan Yasar‡
*Scientific Visualization and Computer Graphics
Rijksuniversiteit Groningen, Groningen, The Netherlands
Email: v.b.codreanu@rug.nl
†Department of Computer Science and Technology
University of Bedfordshire, Bedfordshire, UK
Email: Feng.Dong@beds.ac.uk
‡Rotasoft Inc. Email: see http://www.rotasoft.com.tr/

***Abstract*—This paper presents a method that takes advantage of powerful graphics hardware to obtain fully affine-invariant image feature detection and matching. The chosen approach is the accurate, but also very computationally expensive, ASIFT algorithm. We have created a CUDA version of this algorithm that is up to 70 times faster than the original implementation, while keeping the algorithm's accuracy close to that of ASIFT. It's matching performance is therefore much better than that of other non-fully affine-invariant algorithms. Also, this approach was adapted to fit the multi-GPU paradigm in order to assess the acceleration potential from modern GPU clusters.**

***Keywords*—High performance computing, GPU, Computer Vision**

I. Introduction

Local feature extraction from images is one of the main topics in pattern matching and computer vision in general. For any object present in an image, the locally extracted points provide the feature description of that object. The purpose of this description is to recognize that object when present in a different image. The image matching problem has as inputs two sets of images: the *query* and *search* images. The *search* images are the images that contain the objects of interest. The features of these images are usually computed first and then saved in a database of features. The next step is the computation of features from the *query* images. The *query* features are then compared to the *search* set of image features by using a similarity measure to find the closest matches. The composition of each set of images is dependent on the target application.

Probably one of the most popular feature detection algorithms is the Scale-Invariant Feature Transform (SIFT), proposed by Lowe [1]. It is scale, rotation and translation invariant, but offers relatively modest robustness regarding viewpoint change. Some of the applications developed using SIFT include robot localization [2]–[4], panorama stitching [5] or traffic sign recognition [6].

However, as most algorithms of its kind, SIFT is not fully affine-invariant, and thus does not handle optimally the viewpoint change variations between the *query* and *search* images. A fully affine-invariant extension to SIFT, called affine-SIFT (ASIFT), has been proposed [7]. In that extensive evaluation it has been shown that ASIFT is superior to SIFT on various datasets, both in terms of the number of detected matches and matches' stability. This improved accuracy method comes however with a cost. The computational complexity is relatively high when compared to the other approaches. This is because the method applies SIFT multiple times, SIFT alone already having a high computational cost [8]. This high computational complexity of ASIFT forms the motivation behind our approach.

To our knowledge, the method proposed in this paper is the first implementation of the ASIFT algorithm for efficient execution on GPUs. We chose the NVIDIA CUDA environment for this implementation, mainly because it is more tightly coupled to the NVIDIA hardware than OpenCL, and hence gives better control over the new hardware features introduced in the latest NVIDIA GPUs. The speed-up obtained goes to almost 70 times when applying GPU-ASIFT to high-resolution 5MPixel images, while having a matching accuracy close to that of the original ASIFT. Also, the proposed approach can make use of multiple GPU devices to speed up the computation even more. Our experiments done using 2 GPUs have shown performance increase with speed-up factors ranging from 1.8 to 1.9 times when compared to using a single GPU.

The remainder of this paper is as follows. Section II briefly presents the state-of-the-art feature extraction algorithms and some of the GPU implementations. Section III presents the proposed method for accelerating ASIFT using the GPU. Section IV goes further into the implementation details, with Section V showing the experimental results. Section VI presents our conclusions and possible directions for future work.

II. Related work

Modern feature matching algorithms are typically composed of two stages: the feature detector and the feature descriptor.

978-1-4799-0838-7/13/$31.00 ©2013 IEEE

This scheme works by first detecting the points of interest in the image (the "keypoints") and then by assigning a "descriptor vector" to the region associated with each keypoint. The comparison between the "descriptor vectors" associated to the *query* and *search* images is usually done using a form of the nearest-neighbor algorithm [9] and is defined as the image matching process.

Two of the most popular algorithms designed to solve this problem are the SIFT and the Speeded Up Robust Features (SURF) [10] algorithms. Comparisons between the SIFT and SURF descriptors recommend the former for applications where high-accuracy is required [8]. It has been proven that SIFT outperforms all other image feature extraction methods in terms of scale invariance [11].

A. Scale-Invariant Feature Transform

Scale-invariant feature transform (or SIFT) is an algorithm in computer vision used to detect and describe local features in images [1]. The SIFT method designed by Lowe is composed of a DoG (Difference-of-Gaussians) region detector and a 128-dimensional descriptor based on the orientation distribution in the region. The DoG detector has been proven to provide invariance to translation, rotation and scale change, while the descriptor ensures limited invariance to viewpoint and illumination change. It has been shown by Lowe that a lower-dimensional descriptor reduces the accuracy of the algorithm.

B. GPU-based SIFT implementations

There are a few groups that have presented various SIFT implementations suitable for execution on programmable graphics hardware [12]–[17]. Heymann *et al.*reported a speed-up of about 8 times against a baseline CPU implementation for 640×480 images [17], while Sinha *et al.*obtained a speed-up of 10-12 times over an optimized CPU implementation [18]. However, these two older implementations harnessed the GPU power prior to the introduction of CUDA, making use of Cg and GLSL shader programs, which are more cumbersome to write and debug. From the above implementations, we could access the source code and experiment with two of the ones already implemented in CUDA [12], [16].

C. Affine-SIFT

ASIFT (standing for Affine-SIFT) is a fully affine-invariant feature matching algorithm described in [7]. As most modern feature matching algorithms, ASIFT consists of two distinct stages: keypoint detection and keypoint matching. The first part of the algorithm consists of simulating a predefined number of image tilts and rotations (variations of the camera viewpoint's longitude and latitude angles) and then applying the SIFT algorithm on each of them. Because SIFT is proven to be invariant to scale, rotation and translation, by simulating these viewpoint variations the ASIFT extension covers effectively all six parameters of the affine transform. This method has been mathematically proven to be fully affine-invariant in [7]. The descriptors resulting from the application of ASIFT can be matched using any matching algorithm that would work for SIFT, the descriptors' structure being equivalent.

III. Proposed method

We propose a GPU method to efficiently execute a fully affine-invariant local image feature detection algorithm. The method, based on the ASIFT algorithm, proves to be robust and much faster than the original implementation for a diverse range of image sizes. Moreover, it uses the same parameters as the original implementation, as well as the same sampling methods.

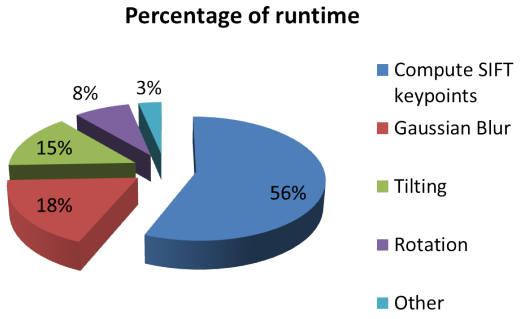

Figure 1. ASIFT keypoint detection profiling data.

As stated previously, the strong motivation towards the development of this implementation was the very high computational cost of ASIFT. In order to find out what the most demanding parts of the application were, we have profiled the original CPU implementation of the ASIFT keypoint detection for 640×480 images. The profiling results are presented in Figure 1. As can be seen in the chart, the computational load is not concentrated in a single function as is the case with many algorithms, but is instead divided across multiple functions. Therefore, in order for the algorithm as a whole to be massively accelerated all of the following four functions need to have a GPU implementation:

- Variation of the latitude angle (rotation)
- Anti-aliasing filtering along the vertical axis using a Gaussian convolution kernel
- Variation of the longitude angle (tilt)
- SIFT keypoint computation for all resulting images

Due to Amdahl's law [19], if we would only parallelize the function that computes the SIFT keypoints, the maximum possible speed-up would be around a factor of 2. On the other hand, if all four functions are accelerated, the maximum application speed-up for 640×480 feature extraction would be around a factor of 33. Moreover, for images of higher resolution, the fixed cost denoted as *Other* in the profiling data is below 1%, permitting even higher speed-ups.

Furthermore, in order to obtain a big whole-application speed-up, one important goal of this implementation was to

minimize as much as possible the communication between the CPU and the GPU, this being the typical bottleneck in GPGPU applications.

For this implementation of the ASIFT algorithm on the GPU, adaptation of the original CPU algorithm was required for a better fit. The function that has the highest computational cost from the ASIFT keypoint detection step is the one that detects SIFT keypoints out of a distorted image. It has a weight of more than half of the runtime, and thus is a clear candidate for parallelization. For the purpose of efficiently computing the SIFT keypoints we chose *siftGPU* [12] as our building block.

We have implemented CUDA kernels for the rest of the functionality, giving us the advantage of only needing a single memory copy from host to device for the input image. All the subsequent image transformations are executed on the GPU, on data already present in device memory. After finishing the execution of ASIFT, the GPU memory is freed and the detected keypoints are read back in host memory for further processing. The detailed description is given in Section IV.

IV. GPU Implementation

We have implemented the ASIFT for efficient execution on the GPU with the following objectives in mind: (1) Perform keypoint detection and keypoint matching in a fraction of the time taken by the CPU implementation, (2) Create an implementation that is scalable in terms of the number of GPUs and (3) Obtain results with similar accuracy to those from the original ASIFT implementation.

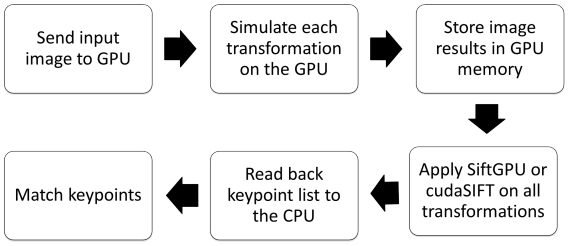

Figure 2. GPU-ASIFT execution diagram.

The diagram of this implementation is presented in Figure 2. Following is the description for each of the computational stages.

A. Transferring the image to the GPU

This is the only image copying throughout the execution of the algorithm. All other generated images are allocated and computed directly in device memory, in order to minimize the CPU↔GPU data transfers. This is the typical problem in GPGPU applications, and we consider that the execution flow of our algorithm greatly reduces the need for such data copies.

B. Simulation of all image transformations

This step computes the three functions required to simulate all possible affine distortions caused by the change of the camera optical axis orientation from a frontal position. The distortions depend upon two parameters: the longitude angle ϕ and the latitude angle θ shown in Figure 3. The images undergo ϕ -rotations followed by tilts with parameter t given by Equation (1).

$$t = |\frac{1}{\cos\theta}|. \tag{1}$$

The image rotation operation is formally defined in Equation (2), where $u(i,j)$ is the input image and $\bar{u}(i,j)$ represents the bilinear interpolated image.

$$u'(i,j) = \bar{u}(i\cos\phi - j\sin\phi, i\sin\phi + j\cos\phi). \tag{2}$$

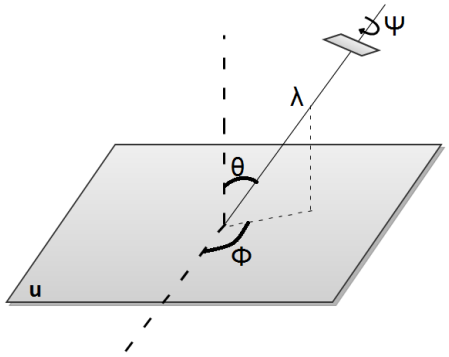

Figure 3. Geometric interpretation of the six affine parameters.

A CUDA kernel was created for efficiently computing the image rotation operation. The kernel performs a rotation by angle ϕ on a 2D input image $u(i,j)$, as described by Equation (2), using bilinear interpolation to compute the final rotated image. Since the rotated image must be included in a rectangular window, a background of grey level 128 is set on the undefined parts of the rotated image.

The tilts are modeled by applying an anti-aliasing Gaussian filtering kernel followed by a directional subsampling operation. The filtering operation is done only in the vertical direction, as defined by ASIFT, and is given in Equation (3).

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2}) \quad , \sigma = c\sqrt{t^2-1} \tag{3}$$

The modeling of the directional subsampling operation is defined in Equation (4).

$$u(x,y) \rightarrow u(x,ty) \tag{4}$$

We have implemented a CUDA kernel inspired from the texture-memory based convolution example from the CUDA SDK [20] to provide this functionality. We have used the value $c = 0.8$ as standard deviation for the convolution kernel, the same as in the original ASIFT.

The tilting operation represents the application of an affine transformation to an image. NVIDIA offers along with the CUDA toolkit different sets of libraries containing accelerated primitives. The NPP (NVIDIA Performance Primitives) library [21] contains a set of image/signal processing functions executed efficiently by NVIDIA hardware. We have identified the function *nppiWarpAffine* as solving the directional subsampling equation (4) required by ASIFT and further use it in this implementation.

C. SIFT keypoint computation for all resulting images

The next step after having all tilted/rotated versions of the input image is to detect the SIFT features associated to each of the transformations. For this purpose we use a modified version of *siftGPU*, tailored to our specific needs. An important modification to the *siftGPU* original code is adding the possibility of loading image data stored directly in device memory. In the original implementation, *siftGPU* supported only image data transferred from the host to device memory. For our application this would mean that each GPU-computed image transformation should first be transferred to host memory and then transferred back from host to device, or that the image transformations should be computed on the CPU. Both these options would deteriorate the performance of a GPU approach, and through this modification the computational cost of passing an image to *siftGPU* is not bigger than that of a pointer assignment.

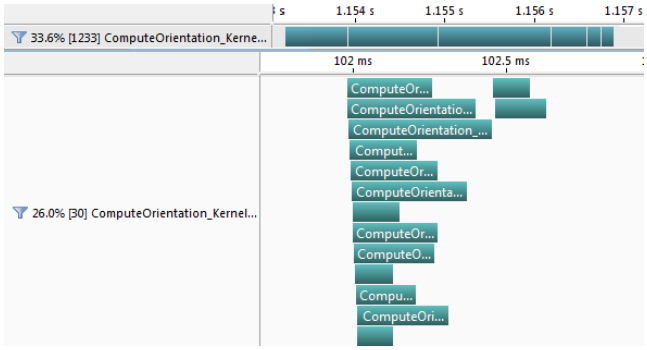

Figure 4. CUDA streams optimization to *siftGPU*. Serial kernel execution is compared to concurrent kernel execution.

Another important modification done to the *siftGPU* implementation was to increase its level of concurrency. This is done through the usage of CUDA streams in some of its important functions. For example, the function that computes the keypoint orientations calls a CUDA kernel multiple times, but the kernels individually don't fully occupy the GPU. All the calls are independent though, so newer CUDA devices of compute capability 2.0+ can launch multiple kernels concurrently to increase the GPU occupancy if the code is written accordingly (asynchronous with regard to memory copies). The impact of this modification is outlined in the GPU profiling data obtained from NVIDIA's CUDA Visual Profiler from Figure 4. For a sequence of 15 independent kernel calls, the serial version executes more than four times slower than our stream-enhanced concurrent one. This improvement alone speeds up *siftGPU* by about 20% when running on NVIDIA GTX680 hardware.

After having the keypoints computed for each affine distortion of the input image, the list of keypoints is read back and stored in host memory.

D. Matching keypoints between two images

In order to match the keypoints computed for two images, we are using the same approach as the original ASIFT. The first and most computationally-expensive part of the keypoint matching stage is computing the similarity between all keypoint descriptors for each pair of image transformations(rotation/tilt). The basic idea for the matching stage is to find the two minimum distances between each detected feature from each transformation of the *query* image and all other features from all transformations of the *search* image. The distance function used is either the L_1 or L_2 norm. If the distance ratio is greater than 0.8 the match is discarded, otherwise the match is considered stable and is saved in the matched keypoints list [1]. Thus, if we set the tilt parameter t such that the total number of generated distorted images is N, N^2 calls to the SIFT descriptor matching routine will be made for each *search* image.

After all matches from all transformations are detected, some further refining is needed. The identical matches from different transformations, as well as the one-to-multiple and multiple-to-one matches are discarded. The next step is discarding the false matches using the powerful Moisan-Stival epipolar geometry-based ORSA (Optimized Random Sampling Algorithm) [22]. The number of matches resulted after applying ORSA is essentially the number of correct matches, and is being used in the results section to assess the accuracy of our implementation. All these steps follow closely the ASIFT implementation of keypoint matching. Because ASIFT typically detects a large number of keypoints, especially for large images, the biggest computational cost is associated to the N^2 calls made to the SIFT descriptor matching routine. In order to alleviate this cost, we have integrated the keypoint matching routine from *siftGPU* in our implementation. All the other parts of the matching stage remain identical as in the original ASIFT method.

E. Multi-GPU implementation

Apart from providing a SIFT implementation for the GPU, a proof-of-concept multi-GPU approach to SIFT is presented

in [12]. This multi-GPU solution makes use of either a multi-process multi-GPU or a multi-threaded multi-GPU implementation. It essentially creates multiple processes/threads and each of them controls one distinct NVIDIA GPU. Then, the SIFT keypoint extraction is started independently, with each ⟨*CPU thread*,*GPU device*⟩ pair computing features for a different image concurrently.

This type of multi-GPU parallelism is a good fit for the ASIFT algorithm, given that the ASIFT computation is largely independent, several image transformations being generated and SIFT being applied independently on all of them. In our multi-GPU implementation initially only one thread is started and the number of available GPUs is detected. If multiple GPUs are present on the system, the master thread divides the computational workload among the participating GPUs by creating a CPU thread for each detected GPU device. After all child threads finish the computation, control is returned to the master thread that creates the final keypoint list.

As an example, if we choose to simulate 3 tilts in our application, the sampling defined within ASIFT generates 10 independent images and SIFT is applied on all of them, while if we choose to simulate 7 tilts, 61 independent images are generated. Thus, ideally, each image transformation followed by the application of SIFT can be executed as an independent thread, and the limit of achievable concurrency is the number of independent images processed. By reaching this limit of parallelism, we expect ASIFT keypoint detection to have a computational cost as high as a single instance of SIFT on the largest image transformation from within ASIFT.

We have evaluated this implementation on the NVIDIA GTX690 dual-GPU card and the results were as expected. Our execution times when using the GTX690 almost halved as compared to the GTX680, as will be presented in Section V.

V. Experimental results

In this section we present the results obtained using the previously described implementation in terms of both execution speed and matching accuracy. We first focus on evaluating the execution speed-up when using images with resolutions ranging from 640×480 to 5MPixel. Secondly, we evaluate the system using the classic evaluation metric (number of true matches) and a classic dataset. The experiments include tests with the standard Mikolajczyk database [23] and also with another set of images that exhibit strong viewpoint change.

Our test hardware is composed of a quad-core Intel Core i7-2600K CPU with 16GB DDR3 RAM and an NVIDIA GTX680 GPU with 2GB GDDR5 device memory. The operating system is 64-bit Windows 7, and the programming environment is composed of Visual Studio 2010 and NVIDIA CUDA 5.0. To assess the performance of the multi-GPU approach, we have used an NVIDIA GTX690 dual-GPU card with 4GB of GDDR5 memory.

Figure 5. Images used for the quantitative analysis [24].

A. Evaluation of execution speed

In this section we focus on evaluating the speed-up obtained against the original ASIFT implementation. The most important factor for the keypoint detection and descriptor generation stage is the size of the input image (number of pixels). Thus, we vary the size of the input images, and compare the implementations regarding the keypoint detection time.

The image pair used in this evaluation is shown in Figure 5. The images are originally sampled at 3MPixel (1536×2048) resolution and are taken from different viewpoints. They are then downscaled and upscaled to cover the following resolution values: 480×640, 600×800, 1024×1280, 1200×1600 and 1936×2584 chosen for experimentation. Because the viewpoint change is not so abrupt for this image pair, and also to limit the number of detected keypoints that tends to be very large for high resolution images, we have set the maximum value for the tilt parameter from Equation (1) to 2 for this set of evaluations. The experimental results are presented in Table I. The "Time" column represents the total time spent detecting the keypoints out of both input images. "*single-core*" is the single-core ASIFT implementation and "*quad-core*" is the OpenMP-augmented implementation utilizing all 4 CPU cores. "*single-GPU*" is the single-GPU ASIFT implementation running on the NVIDIA GTX680 and "*dual-GPU*" is the multi-GPU implementation running on the dual-GPU NVIDIA GTX690.

The speed-up factor of GPU-ASIFT relative to the single-core CPU implementation ranges between 22 and 68, and between 6 and 17 relative to the OpenMP 4-core implementation. This is including all the memory copies between host and device. The OpenMP approach is highly scalable, offering near-linear or even supra-linear speed-up up to the number of SIFT instances called. An advantage of the GPU implementation is that it relieves the CPU of excessive load, so in this scenario the CPU can be also used for other tasks.

The same high degree of scalability that the OpenMP implementation exhibits is observed in our multi-GPU implementation. Due to the highly-independent processing employed

within ASIFT, the workload can be evenly divided across multiple GPUs as presented in the previous section. The obtained speed-up by using two GPU devices instead of one ranges from factors of 1.75 to 1.9, depending on the image sizes. When using this approach, the execution time for finding keypoints in a VGA(640×480) image is about 110ms. We expect that by using more GPU devices(e.g. two NVIDIA GTX690s) we can detect ASIFT features in real-time. This would be a tremendous speed-up, considering that initially more than 4 seconds were needed to analyze a single VGA frame.

One noticeable problem of the original ASIFT implementation is the very high cost of performing keypoint matching, both in terms of memory consumption and of execution time. As an example, for the 1024x1280 image test, the CPU version of ASIFT detected 780 matches out of two sets of about 25,000 keypoints each in 230 seconds. In contrast, GPU-ASIFT generates about 20,000 keypoints for each image, that are matched in 1.1 seconds. Even for images having detected keypoint numbers in the range of 100,000 each, as is the case for the high resolution 1936x2584 image, the matching time is under 10 seconds when using this method.

The ASIFT-CPU implementation typically generates between 11% and 40% more keypoints than our GPU implementation for these input images. The relative difference in terms of detected keypoints is reduced as the size of the input image increases. This happens because of the parameter difference in the design of the underlying SIFT routines. *siftGPU* thus tends to find less keypoints than the SIFT routines used in the original ASIFT. However, the number of keypoints detected by GPU-ASIFT is consistently over 10 times larger than that detected by *siftGPU* on the whole range of image sizes. This greatly improves the matching accuracy, the number of correct matches being also on average about 10 times larger.

TABLE I
TIMING RESULT IN MILLISECONDS FOR ALL IMAGE SIZES, USING ALL ASIFT IMPLEMENTATIONS.

Resolution	single-core	quad-core	single-GPU	dual-GPU
480x640	8861	2262	390	222
600x800	13634	3464	483	272
1024x1280	35101	8414	765	425
1200x1600	54039	13136	921	492
1536x2048	90465	21840	1342	710
1936x2584	141711	34086	2075	1092

The total image area for which SIFT is applied within ASIFT is given by the input image area multiplied by $A(t)$ from Equation (5).

$$A(t) = 1 + (|\Gamma_t| - 1)\frac{180^\circ}{72^\circ}$$
$$|\Gamma_t| = |\{1, \sqrt{2}, ..., \sqrt{2}^{t-1}\}| \quad (5)$$

$|\Gamma_t|$ represents the number of simulated tilts. With the maximum tilt value set to 2, the value of $|\Gamma_t|$ is 3 and hence the area over which SIFT is applied is 6 times the area of the input image. For obtaining SIFT keypoints on 6 times the input image, the cost is on average only about 4 times higher than that of a single SIFT call for the same input image, including the cost for performing the image distortions. This is possible because all image data was calculated on the GPU, thus eliminating the need for expensive CPU↔GPU copying. Another contributor to this improvement is the *siftGPU* concurrent kernel execution optimization described in the previous section.

B. Evaluation of accuracy

For the accuracy results presented in this section we have used images that exhibit abrupt viewpoint change, to assess the quality of this implementation. We have chosen the *graffiti* images from the Mikolajkzyk [23] standard dataset, presented in Figure 6. Also, we have chosen the *magazine* image set from Figure 7 out of the original ASIFT dataset [7].

Figure 6. Images from the Mikolajczyk standard dataset [23]. Image 1 and Image 6 are shown.

The first set of images is composed of one frontal view and 5 different views of the same object, with increasing change in viewpoint, all sampled at a 800×640 resolution. Figure 6 shows the frontal image and the one with the largest viewpoint change. The second set aims at evaluating the detector with respect to variation of scale and viewpoint. It is composed of 9 images sampled at a 600×450 resolution, one frontal view zoomed by a factor of 4, and 8 images with 10° variation in viewpoint, ranging from 10° to 80° and without zoom.

Figure 7. Images from the ASIFT dataset [7].Taken from middle distance (zoom by a factor of 4) at frontal view and at 80 degree angle.

As has been also shown previously [7], for a very high

viewpoint difference SIFT cannot detect matches between the images. From the first set, SIFT cannot detect matches for 2 out of the 5 tests, the same holding for the second set. Typically, SIFT's robustness drops sharply when the viewpoint variation increases to over 50°, as also stated by Lowe in [1]. The accuracy comparison between GPU-ASIFT, CPU-ASIFT and *siftGPU* on the images that exhibit the largest viewpoint difference out of these datasets is found in Tables II and III. We use the number of correct matches to assess the accuracy of the three approaches. The first column from the tables lists the compared methods, together with two parameters. The t parameter is the maximum tilt parameter, and the u parameter selects whether to perform upsampling by a factor of two in each direction prior to execution or not. The second column lists the total number of correct matches detected by each method.

TABLE II
ACCURACY RESULTS FOR THE MIKOLAJCZYK DATASET ON THE IMAGES FROM FIGURE 6.

Method	Number of correct matches [#]
ASIFT-CPU($t = 2, u = 0$)	160
GPU-ASIFT($t = 2, u = 0$)	159
GPU-ASIFT($t = 2, u = 1$)	445
ASIFT-CPU($t = 4\sqrt{2}, u = 0$)	753
GPU-ASIFT($t = 4\sqrt{2}, u = 0$)	497
GPU-ASIFT($t = 4\sqrt{2}, u = 1$)	1508
siftGPU	1

TABLE III
ACCURACY RESULTS FOR THE ASIFT DATASET ON THE IMAGES FROM FIGURE 7.

Method	Number of correct matches [#]
ASIFT-CPU($t = 2, u = 0$)	69
GPU-ASIFT($t = 2, u = 0$)	34
GPU-ASIFT($t = 2, u = 1$)	90
ASIFT-CPU($t = 4\sqrt{2}, u = 0$)	141
GPU-ASIFT($t = 4\sqrt{2}, u = 0$)	151
GPU-ASIFT($t = 4\sqrt{2}, u = 1$)	467
siftGPU	0

The tables are composed of two sets of measurements, one with a maximum tilt value t set to 2, and the other with the maximum value set to $4\sqrt{2}$. The second set generates more matches than the first because more image distortions are performed, at the expense of a higher computational cost. Also, $u = 1$ in the table means that the input image was upsampled by a factor of two prior to performing ASIFT.

By comparing lines $1 - 3$ and $4 - 6$ in Tables II and III, we can see that our GPU implementation of ASIFT has similar accuracy with regard to the CPU one. The accuracy can be further increased either by upsampling the input image or by simulating a larger number of image transformations. Simulating a larger number of transformations increases the execution time by the same factor for both CPU and GPU implementations of ASIFT.

However, for images not exceeding 2MPixel, the input image size can be safely doubled without running out of GPU memory, and all computation applied to the larger image. This is the case for the results from lines 3 and 6 in Tables II and III. In this scenario the number of correct matches in the GPU-ASIFT case is about 3 times bigger, with the execution time only doubling, but still being a fraction of the CPU-ASIFT execution time. When using the same upsampling technique for increasing the keypoints detected by the CPU version of ASIFT the execution time is 4 times higher. For this purpose one should compare the execution time data from Table I.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have demonstrated a method to increase the throughput of the ASIFT algorithm by using programmable graphics hardware. We have shown performance speed-ups by factors of up to 70 when compared to the single-core CPU implementation, while keeping the same level of accuracy. We have evaluated the algorithm on a standard dataset for quality checks and also on a range of image sizes to assess speed performance. The result is that the current approach can process a 5MPixel image in about one second while adhering to the high level of accuracy of ASIFT, making it the fastest implementation currently available. Also, a multi-GPU approach that further accelerates ASIFT was implemented. The results were as expected, the algorithm's performance almost doubling when using a dual-GPU card. This leads to the idea that the initially intractable problem of real-time ASIFT detection can be solved by using 4 GPUs.

As future work we are thinking of implementing a GPU-ASURF (affine-SURF) implementation using this framework. We are expecting that implementation to work in real-time using a single GPU, based on the experiments presented in [8] that show the relative difference in computing time between SIFT and SURF. Also, we aim at using these highly accurate descriptors as input data to machine learning classifiers and address challenging object recognition datasets like the PascalVOC and Caltech-256 [25] [26].

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's Seventh Framework Programme managed by REA-Research Executive Agency http://ec.europa.eu/research/rea (FP7/2007-2013) under grant agreement no. 286545.

REFERENCES

[1] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," in *IEEE International Conference on Robotics and Automation, Proceedings 2001 ICRA*, vol. 2. IEEE, 2001, pp. 2051–2058.
[3] H. Tamimi, H. Andreasson, A. Treptow, T. Duckett, and A. Zell, "Localization of mobile robots with omnidirectional vision using particle filter and iterative sift," *Robotics and Autonomous Systems*, vol. 54, no. 9, pp. 758–765, 2006.
[4] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The International Journal of Robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.
[5] M. Brown and D. Lowe, "Recognising panoramas," in *Proceedings of the Ninth IEEE International Conference on Computer Vision*, vol. 2, no. 1218-1225, 2003, p. 5.
[6] M. Kus, M. Gokmen, and S. Etaner-Uyar, "Traffic sign recognition using scale invariant feature transform and color classification," in *23rd International Symposium on Computer and Information Sciences, ISCIS'08*. IEEE, 2008, pp. 1–6.
[7] J. Morel and G. Yu, "ASIFT: A new framework for fully affine invariant image comparison," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 438–469, 2009.
[8] L. Juan and O. Gwun, "A comparison of SIFT, PCA-SIFT and SURF," *International Journal of Image Processing (IJIP)*, vol. 3, no. 4, pp. 143–152, 2009.
[9] J. Beis and D. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Proceedings*. IEEE, 1997, pp. 1000–1006.
[10] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006.
[11] J. Morel and G. Yu, "Is SIFT scale invariant?" *Inverse Problems Imaging*, vol. 5, pp. 115–136, 2011.
[12] C. Wu, "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)," http://cs.unc.edu/~ccwu/siftgpu, 2007.
[13] A. Chariot and R. Keriven, "GPU-boosted online image matching," in *9th International Conference on Pattern Recognition, ICPR 2008*. IEEE, 2008, pp. 1–4.
[14] S. Sinha, J. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, vol. 22, no. 1, pp. 207–217, 2011.
[15] S. Warn, W. Emeneker, J. Cothren, and A. Apon, "Accelerating SIFT on parallel architectures," in *IEEE International Conference on Cluster Computing and Workshops, CLUSTER'09*. IEEE, 2009, pp. 1–4.
[16] M. Bjorkman, "A CUDA implementation of SIFT," *http://www.csc.kth.se/~celle/*.
[17] S. Heymann, K. Muller, A. Smolic, B. Frohlich, and T. Wiegand, "SIFT implementation and optimization for general-purpose GPU," in *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2007, p. 144.
[18] S. Sinha, J. Frahm, M. Pollefeys, and Y. Genc, "GPU-based video feature tracking and matching," in *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, vol. 278, 2006, p. 4321.
[19] G. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the Spring Joint Computer Conference*. ACM, 1967, pp. 483–485.
[20] V. Podlozhnyuk, "Image convolution with CUDA," *NVIDIA Corporation white paper, June*, vol. 2097, no. 3, 2007.
[21] NVIDIA, "NVIDIA NPP library," *https://developer.nvidia.com/npp*.
[22] L. Moisan and B. Stival, "A probabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix," *International Journal of Computer Vision*, vol. 57, no. 3, pp. 201–218, 2004.
[23] K. Mikolajczyk, "Affine covariant features," *Collaborative work between: the Visual Geometry Group, Katholieke Universiteit Leuven, Inria Rhone-Alpes and the Center for Machine Perception*, 2007.
[24] S. El-Hakim, "Florence (italy) data set," *http://www.isprs.org/data/florence/default.aspx*.
[25] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.
[26] D. Hoiem, S. K. Divvala, and J. H. Hays, "Pascal VOC 2008 challenge," 2009.