2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.2.01

Accelerated Wide Baseline Matching using OpenCL

CAO Jian[1+], LIANG Jie[2], XIE Xiao-fang[1] and HU Xun-qiang[2]

[1]Department of Ordnance Science and Technology, Naval Aeronautical and Astronautical University

Yantai, China

[2]Graduate Students' Brigade, Naval Aeronautical and Astronautical University

Yantai, China

Abstract—Wide baseline matching is the state of the art for object recognition and image registration problems in computer vision. Robust feature descriptors can give vast improvements in the quality and speed of subsequent steps, but intensive computation is still required. With the release of general purpose parallel computing interfaces, opportunities for increases in performance arise. In this paper we present an implementation of Speeded-Up Robust Feature (SURF) extractor, based on the OpenCL system of GPU programming developed by NVIDIA. For an 800x640 pixel image, the GPU-based method executes nearly 5 times faster than a comparable CPU-based method, with no significant loss of accuracy.

Keywords- Wide Baseline, Match, SURF, GPU, OpenCL

1. Introduction

Wide baseline matching (WBM), based on abstract local features, is the state of the art paradigm in computer vision for object recognition and image registration. While highly effective across a wide array of problems, these methods tend to be very computationally expensive [1]. Recently a multitude of work has emerged using the graphics processing unit (GPU) for scientific computing. The most notable in this context are NVIDIA's CUDA [2] and the Khronos group's OpenCL [3]. OpenCL is a new framework for programming across a wide variety of computer hardware architectures (CPU, GPU, Cell BE, etc) .This characteristic lets programmers writes a single program that uses all resources in the heterogeneous platform, which brings the possibility to make final portable real-time 3D model reconstruction system. To help understand the NVIDIA OpenCL programming guide was used as a reference, as well as a number of excellent tutorials and samples casts at MacResearch.com.

Feature point detection and description is a necessary tool for much computer vision, especially for WBM. Since features can be viewed from different angles, distances, and illumination, it is important that a feature descriptor be relatively invariant to changes in orientation, scale, brightness, and contrast, while remaining descriptive enough to guarantee match precision. We chose the Speeded-Up Robust Features (SURF) descriptor [5]. Comparing those previous algorithms, such as the Scale-Invariant Feature Transform (SIFT)[6], SURF locates features adopt many approximations to increase the speed of subsequent matching operations with almost same matching precision, while themselves being less expensive to compute, as illustrated in Table I[7]. However, SURF cannot yet achieve interactive frame rates on a traditional CPU.

[+] Corresponding author.
E-mail address: ddcjd@163.com

TABLE I. THE COMPARISON RESULT OF SIFT, PCA-SIFT AND SURF

method	Time	Scale	Rotation	Blur	Affine
SIFT	common	best	best	common	good
PCA-SIFT	good	good	good	best	best
SURF	best	common	common	good	good

The rest of this paper is organized as follows: In section 2 presents a survey of OpenCL concepts, along with a short introduction to OpenCL working procedure. In section 3 I will present the SURF algorithm briefly. Section 4 contains the SURF implementation by using OpenCL. In section5, the performance comparison of SURF implementations on OpenCL fit GPU (NVIDIA Geforce 8800GT OpenCL 1.0) and an existing CPU (Intel Core2 Quad Q6600 2.4GHz OpenSURF Library 1.0) will be exhibited. In section 6, conclusions are drawn and directions for future work are given.

2. OpenCL concepts and the working procedure

The main software framework that is under consideration in this work is the Open Computing Language. The first most important concept is that of a compute kernel. A compute kernel is a function that gets executed one GPU in parallel. In OpenCL these are written in a subset of the 1999 C language specification. Computational scientists would need to rewrite the performance intensive routines in their codes as OpenCL kernels that would be executed on the compute hardware. The OpenCL API provides the programmer various functions from locating the OpenCL enabled hardware on a system to compiling, submitting, queuing and synchronizing the compute kernels on the hardware. Finally, it is the OpenCL runtime that actually executes the kernels and manages the needed data transfers in an efficient manner [8].

An OpenCL program requires a number of steps to run. First a context must be created; the context describes a means to communicate with an OpenCL devices such as a single or multiple CPUs, GPUs, or FPGAs; however the focus of this discussion will only be on GPUs [9]. The context also retains and manages memory allocated on the device, and allows for reading and writing to this memory. Allocating memory for use by OpenCL is done through the use of a special malloc function clCreateBuffer. Other functions exist such as clCreateImage2D and clCreate-Image3D, which are useful for more specific applications. Reading memory is done through clEnqueueReadBuffer and clEnqueueWriteBuffer, both of these commands support synchronous and asynchronous operations. Reading and Writing do require the addition of a queue to perform their tasks. To submit commands to the device within a context we use a queue. The queue is specified for all kernels, reading and writing to device memory, and supporting functions such as barriers for synchronization. The OpenCL function, clCreateCommandQueue, creates the queue. Once an operation is enqueued it runs immediatly and asynchro-nously, however to enable synchronous communication OpenCL provides cl_events data types and the functions clWaitForEvents and clEnqueueWaitForEvents. Here cl_ events work as tokens allowing subsets of a group of operations to work together while preserving dependencies. To create a barrier over an entire queue though would be quite cumbersome using cl_events so OpenCL provides clEnqueueBarrier. And with the OpenCL function clCreate-ProgramWithSource the kernel will be compiled during runtime from strings containing the program or read in as pre-compiled binaries, the kernel arguments can be set by function clSetKernelArg. Once a program is compiled it can then be linked against its arguments to create a kernel. This kernel can be enqueued for immediate processing by the GPU. Once we receive the calculated result from the device memory back to CPU memory by clEnqueueReadBuffer function, clReleaseKernel ,clReleaseProgram clRelease-CommandQueue, clReleaseContext and clReleaseMem-Object functions will follow to release the device memory and GPU resources [10].

3. The Surf Algorithm

This section reviews the original SURF algorithm. We defer some of the details to the next section, which discusses our OpenCL based GPU implementation, but we highlight the main points here.

SURF locates features using an approximation to the determinant of the Hessian, chosen for its stability and repeatability, as well as its speed. An ideal filter would construct the Hessian by convolving the second-order derivatives of a Gaussian of a given scale σ with the input image. This is approximated by replacing the second order Gaussian filters with a box filter, as illustrated in Figure 1. Box filters are chosen because they can be evaluated extremely efficiently using the so-called integral mage II, defined in terms of an input image I as

$$II(x,y)=\sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \tag{1}$$

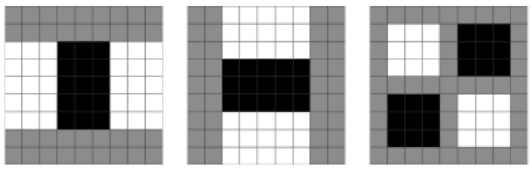

Fig. 1: Box filters approximating 2nd-order partial derivatives of a Gaussian.

Given II, the sum over any arbitrarily-sized, axis-aligned 2D region can be computed in just four lookups as below.

$$\sum = A + D - (C + B) \tag{2}$$

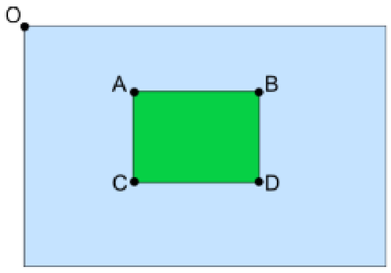

Fig. 2: Area computation using integral images

To achieve scale invariance, the filters are evaluated at a number of different scales, s, and the 3×3×3 local maxima in scale and position space form the set of detected features. Here $s=\sigma$, the scale of the Gaussians used to derive the box filters. A minimum threshold H_0 on the response values limits the total number of features. The location x_0 of each feature is then refined to sub-pixel accuracy via

$$\hat{X} = X_0 - \left(\frac{\partial^2 H}{\partial X^2}\right)^{-1}\frac{\partial H}{\partial X} \tag{3}$$

where $X=(x,y,s)^T$ are scale-space coordinates and $H=|\det(H)|$ is the magnitude of the Hessian determi-nant. The derivatives of H are computed around x_0 via finite differences.

Rotation invariance is achieved by detecting the dominant orientation of the image around each feature using the high-pass coefficients of a Haar filter in both the x and y directions inside a circle of radius 6s. The size of the Haar filter kernel is scaled to be 4s×4s, and the sampling locations are also scaled by s, which is easily accomplished using the integral image.

The resulting 2D vectors are weighted by a Gaussian with $\sigma = 2.5s$ and then sorted by orientation. The vectors are summed in a sliding window of size $\frac{\pi}{3}$, and the orientation is taken from the output of the window with the largest magnitude. Once position, scale, and orientation are determined, a feature descriptor is computed, which is used to match features across images. It is built from a set of Haar responses computed in a 4×4 grid of sub-regions of a square of size 20s around each feature point, oriented along the dominant orientation. Twenty-five 2D Haar responses (dx, dy) are computed using filters of size 2s × 2s on a 5 × 5 grid inside each sub-region and weighted by a Gaussian with $\sigma = 3.3s$ centered at the interest point [5].

The total algorithm runs in approximately 354 ms on a 3 GHz Pentium IV for an 800×640 image [5], or at just under 3 Hz.

4. Implementation

This section covers the major parts of our OpenCL SURF implementation. Our implementation has the follow-ing major steps:

4.1. Integral Image Computation

The primary workhorse of the SURF algorithm is the integral image, which is used to compute box filter and Haar filter responses at arbitrary scales in constant time per pixel. Since it must be computed over the entire image, it is one of the more expensive steps [11].

The computation of the integral image itself is a classic parallel prefix sum problem. It can be implemented as a prefix sum on each row, followed by a prefix sum on each column of the output. Consulting the sample 'oclScan' in OpenCL SDK, we designed the kernel called Integral-Img_kernel. This program operates strictly within a single workgroup and each work item in this workgroup performs a coalesced copy to a local array.

4.2. Feature Detection

Having constructed the integral image, we turn to the evaluation of the box filters used to locate interest points. The first step is to confirm the size of box filters. In Bay's paper, the values for each octave-interval pair dictate the size of the filter which will be convolved at that layer in the scale-space. The filter size is given by the formula,

$$\text{Filter Size} = 3(2^{\text{octave}} \times \text{interval} + 1) \tag{4}$$

Like Bay et al. [3], we use filters of size 9, 15, and 21 at the first three scales. Bay et al. derive their size 9 filters as the best box-filter approximation of the second-order derivatives of a Gaussian with scale $\sigma = 1.2$ and compute the scale associated with the rest of the filters based on the ratio of their size to that of the base filter. After reaching a filter size of 27, Bay et al. begin incrementing by 12, for several steps, then 24, etc., simultaneously doubling the sampling interval at which filter responses are computed each time the filter step size doubles. Like SIFT algorithm, SURF calculate the Hessian matrix, H, as function of both space x = (x; y) and scale σ

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{5}$$

Here $L_{xx}(x,\sigma)$ refers to the convolution of the second order Gaussian derivative $\frac{\partial^2 g(\sigma)}{\partial x^2}$ with the image at point $X = (x, y)$ and similarly for L_{xx} and L_{xy}. These derivatives are known as Laplacian of Gaussians.

Working from this we can calculate the determinant of the Hessian for each pixel in the image and use the value to find interest points. SURF performing conjunctions labeled D_{xx} D_{xy} D_{yy} between varying size box filters and integral image mentioned above to approximate Laplacian of Gaussians, and Hessian determinant using the approximated Gaussians:

$$\det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{6}$$

The determinant here is referred to as the blob response at location $X = (x, y, \sigma)$. The search for local maxima of this function over both space and scale yields the interest points for an image.

Following detection ideas we set three OpenCL kernels named fasthessian_kernel nonmaxsuppression_Kernel and keypointinterpolation_kernel. fasthessian_kernel takes care about the conjunctions of varying size box filters and the IntegralImg_kernel output in device Global memory and establish the scale-space for further work. nonmaxsuppression_Kernel search for local maxima of this function over both space and scale yields, check to see if we have a max (in its 26 neighbors) as described in Bay's paper(show in Figure 3). Finally use the algorithm in (3) keypointinterpolation_kernel output a vector of accurately localized sub-pixel interest points.

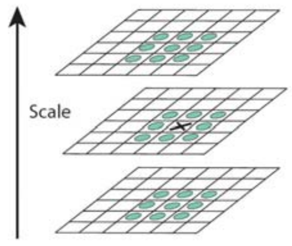

Fig. 3: Non-Maximal Suppression

4.3. Orientation Detection

In order to achieve invariance to image rotation each detected interest point is assigned a reproducible orientation. Extraction of the descriptor components is performed relative to this direction so it is important that this direction is found to be repeatable under varying conditions. Like SIFT, conjunctions are made within a scale dependent neighborhood of each interest point detected by the Fast-Hessian.But in SURF algorithm integral images used in conjunction with filters known as Haar wavelets are used in order to increase robustness and decrease computation time. Haar wavelets are simple filters which can be used to find gradients in the x and y directions, as illustrated in Figure 4.

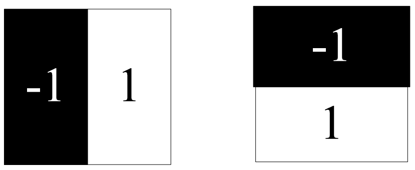

Fig. 4: Haar Wavelets..

In Bay's paper, to determine the orientation, Haar wavelet responses of size 4σ are calculated for a set pixels within a radius of 6σ of the detected point, where σ refers to the scale at which the point was detected. Within our kernel orientation_kernel the branching logic slow down efficiency because circle estimation.

The resulting 2D vectors are weighted by a Gaussian with $\sigma = 2.5s$ and then sorted by orientation. The vectors are summed in a sliding window of size $\frac{\pi}{3}$, and the orientation is taken from the output of the window with the largest magnitude.

4.4. Feature Vector Calculation

To construct the feature vectors, axis-aligned Haar responses are computed on a 20s × 20s grid. The lattice points of the grid are aligned with the feature orientation, and the Haar response vector is rotated by this angle as well. Normalization is done on the GPU using another simple reduction to compute the vector magnitude. The kernel carry out above calculations named descriptors_kernel.

5. Result

We ran our implementation on a GeForce 8800 GT using test images acquired from http://www.robots.ox.ac.uk/~vgg/ research/affine/. Those OpenCL kernels were programmed with OpenCL 1.0 in *.cl files separately, the comparative program achieved with OpenSURF Library 1.0. All of them worked on the so common inexpensive computer (equipped with Intel Core2 Quad Q6600 2.4GHz and 4G RAM). The result showed in Figure 5, by the way our times do not include the time to transfer the image to the graphics card, since this only affects latency, not throughput. For accuracy, we calculated the same data on the same computer ten times. Results show in table II. On average, we observed the 8800 to be about 5 times faster than the CPU. The OpenCL version SURF algorithm can really reach the interactive frame rates. Figure 6 shows the final matched image pair.

TABLE II. COMPARATIVE RESULTS ON CPU& GPU(TIMES IN MS)

No.	**OpenSURF-Based CPU SURF**	**OpenCL-Based GPU SURF**
1	219.3	43.7
2	231.1	44.6
3	207.9	40.2
4	232.4	47.4
5	240.2	41.2
6	223.6	40.8
7	233.7	46.6
8	238.5	42.6
9	269.2	40.5
10	225.5	44.6
Average	232.1	43.2

To visualize this comparison a figure was provided.

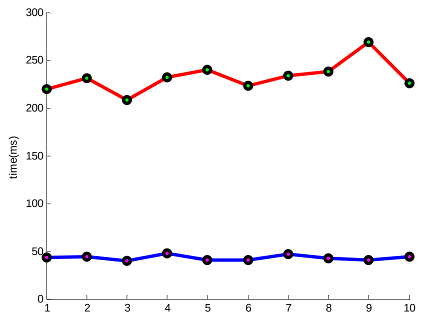

Fig. 5. The visualization of comparative result(Red line:CPU SURF Blue line: GPUSURF)

Fig. 6. The final matched image pair

6. Conclusion

We have presented a GPU implementation of the SURF feature extraction algorithm that achieves interactive frame rates. The present work shows great promise for future development. There are several possible directions for future work.

First of all, due to lack of time, no optimization was done to the OpenCL kernels. And the data structure should be redesigned to transfer data as much as possible from CPU to GPU to take advantage of efficiency of GPU.

Second, in the future, we plan to use some kind of DSP processors to establish portable real-time 3D model reconstruction system. The OpenCL based implementations for those algorithms used in following reconstruction process should be studied as well.

7. Reference

[1] Jonathan Lisic Discriminant Adaptive Nearest Neighbors Classification Implementation with OpenCL and OpenMP,http://CSI702.net/CSI702/image/DANN_ON_OPENCL.pdf

[2] NVIDIA: CUDA web page, http://www.nvidia.com/object/ cuda_learn.html, 2009.

[3] KHRONOS: OpenCL overview web page, http://www.khronos.org/ opencl/, 2009.

[4] NVIDIA Corporation. OpenCL Programming Guide for the CUDA Architecture Version 2.3. Santa Clara : NVIDIA, 2009.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features, European Conference on Computer Vision, 1:404~417, 2006.

[6] D.G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91~110, 2004.

[7] Luo Juan O Gwun A comparison of SIFT, PCA-SIFT and SURF. Journal of Image Processing (IJIP),4(3):143~152,2008

[8] GOHARA D.: OpenCL tutorials web page, http://www.macresearch. org/ opencl, 2009.

[9] Wikipedia: OpenCL. Wikipedia. http://en.wikipedia.org/wiki/ OpenCL. May 05, 2010 [Cited: May 05, 2010.]

[10] OpenCL API 1.0 Quick Reference Card. s.l. Khronos Group, 2009

[11] Timothy B. Terriberry, Lindley M. French, and John Helmsen GPU Accelerating Speeded-Up Robust Features, DPVT, 2008