Complex Geometric Primitive Extraction on Graphics Processing Unit

Mert Değirmenci
Department of Computer Engineering,
Middle East Technical University, Turkey
mert.degirmenci@ceng.metu.edu.tr

ABSTRACT

Extracting complex geometric primitives from 2-D imagery is a long-standing problem that researchers have had to deal with. Various approaches were tried from Hough transform based methods to stochastic algorithms. However, serial implementations lack sufficient scalability on high resolution imagery. As sequential computing power cannot pace up with the increase in size of datasets, researchers are compelled to exploit parallel computational resources and algorithms. In this study, we have merged parallelization capability of GPUs with inherent parallelism on genetic algorithms to cope with the problem of detecting complex geometric primitives on high resolution imagery. We have implemented ellipse detection on commodity graphics processing unit and showed that our GPU implementation achieve high speed-up relative to state of the art CPU by experimental results.

Keywords

Geometric primitive extraction, Genetic algorithm, GPU.

1. INTRODUCTION

Most of the tasks in image analysis involve geometric primitive extraction as a preprocessing step. Therefore, efficiently detecting primitives is an important research topic in the domain of computer vision. Researchers have proposed numerous solutions to detect geometric primitives [Tia96], [Rob98], [Rot94].

Hough transform, HT, based techniques have been widely used for geometric primitive extraction. The idea of classical Hough transform was to perform a mapping from image space to parameter space in order to obtain a function. Optima's of this function correspond to instances of primitives. HT is powerful for detecting simple primitives such as lines. However, computational and memory complexity of classical Hough transform increase exponentially along with the number of parameters [Pen99].

To alleviate problems of HT, randomized Hough transform, RHT, has been proposed by Kultanen et al. [Kul90]. RHT performs converging mapping to parameter space by randomly sampling a number of pixels from image space that will ensure convergence to one point.

RHT is not the only approach that tried to reduce the parameter space of classical Hough transform. Researchers have proposed partitioning parameter space to smaller subspaces by using features of specific complex geometric primitives. Si-Cheng et al. proposed real time ellipse detector that uses analytical properties of ellipses and edge gradient information to divide 5-D parameter space of classical Hough transform [Sic05].

Stochastic approaches were also tried to detect geometric primitives. Ever since geometric primitive extraction has been shown to be an optimization problem [Rot93], researchers have tried optimization algorithms to extract primitives [Rot94], [Pen99].

Genetic algorithm, GA, is one of the most popular stochastic approaches to extract geometric primitives. In the context of natural evolution, genetic algorithm simulates parallel evolution of individuals to find approximate solutions to optimization problems for which no efficient algorithm is known to find the exact solution otherwise. Since primitive extraction can be regarded as an optima search problem, genetic algorithm fit well into the problem of detecting imperfect instances of geometric primitives. Thus, many researchers have developed genetic algorithm oriented methods to detect geometric shapes [Yao05], [Kaw98], [Rot94].

GA based techniques use the feedback obtained from population to cluster solutions around optima's, whereas HT based methods exhaustively search image space irrespective of distribution in parameter space. Although genetic algorithms has inherent strengths over HT based methods, extracting all instances of a geometric primitive cannot be directly translated into problem space of classical genetic algorithm.

Detection of single most observable geometric primitive, on the other hand, can be converted into unimodal optimization problem. However, in real life applications, we need to detect multiple primitives that qualify certain criteria. This corresponds to multi-modal optimization. Researchers have proposed variety of multi-modal optimization stochastic algorithms to extract geometric primitives from 2-D imagery.

Lutton et al. proposed sharing genetic algorithm, SGA, where diversification of population is maintained by promoting fitness of local optima [Lut94]. Fitness of similar individuals is shared to decrease the clustering around global optima while guiding search towards uninhabited areas.

Yao et al. proposed multi-population genetic algorithm [Yao05]. They suggest island oriented model of population where each individual is prompted to live on a matching island. New island generation is also possible if an individual is not close to any existing island.

In this study, we have implemented multi-population genetic algorithm for extraction of complex geometric primitive on graphics processing unit. We have chosen Nvidia's compute unified device architecture, CUDA, as a development platform. Our genetic algorithm implementation on GPU has been tested for detection of ellipses and shown successful improvement over an optimized serial implementation.

2. GEOMETRIC PRIMITIVE EXTRACTION ON GPU

Our implementation of geometric primitive extraction starts by detecting edges of image. Edge image is then partitioned into tiles on which a copy of our genetic algorithm runs. In CUDA , each block works on its corresponding isolated island, and each thread is responsible for an individual on that island, as it can be seen in figure [1]. After loading data, threads behave as individuals who mate with other individuals to produce fitter offsprings for the next generation. However, these responsibilities are distributed after coalesced loads of tiles into shared memory. Current graphics processing units have sixteen kilobytes of shared memory available per block. To comply with limitations of graphics hardware, edges are encoded in bit string.

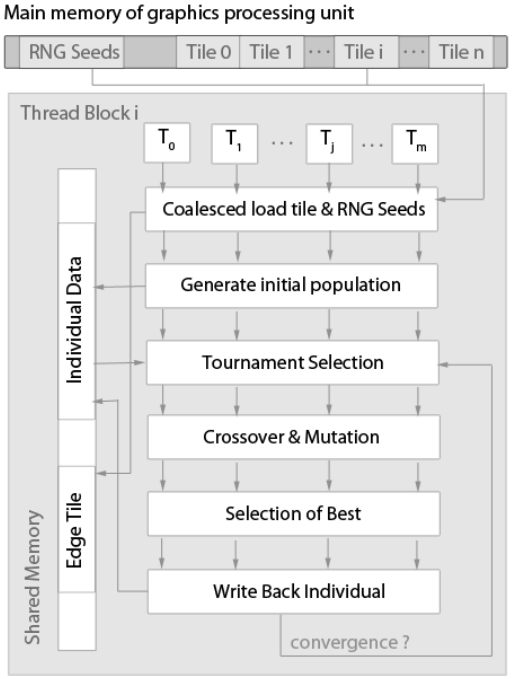

Figure 1. Evolution scheme in compute unified device architecture.

Genetic algorithm needs a random number generator. We have chosen park-miller pseudo-random number generator, RNG, due to its simplicity. Each individual has to load the seed for park-miller RNG from global memory of the device that was previously set using Mersenne Twister RNG. As seen in figure [1], each thread block loads initialization data to shared memory in a coalesced manner. RNG seeds, on the other hand, are kept in the local memory of the threads.

After initialization, genetic algorithm is carried out in parallel. When to stop algorithm is a challenging question in general. An indication of convergence can be checked to terminate the process [Yao05]. More practical applications employ predefined number of iterations before termination [Won09]. This is a tradeoff between accuracy and computational complexity. We have chosen to terminate the genetic algorithm after fixed number of epochs determined by empirical results.

Last step is to merge the results of thread blocks to find potential primitives. Individuals write the best result they have obtained to global memory of graphics processing unit. Results are then copied back to CPU memory, where they are combined to find candidate primitives sequentially. Merging of results is required only if segmentation is overlapped on image. If not, thread blocks can threshold the population and output a fixed number of qualified individuals.

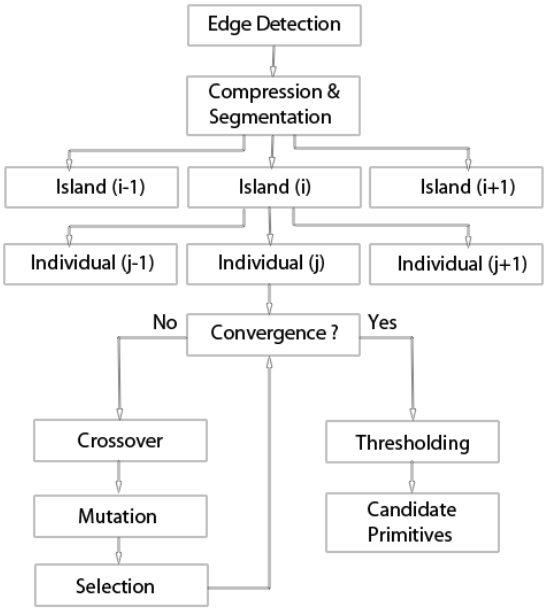

Figure 2. Architecture of primitive extractor on graphics processing unit.

Figure [2] shows phases of geometric primitive extraction on graphics processing unit. We will explain details of each step in the following sections of this document.

Edge detection

Edge detection is an important part of our process. Falsely detected edges lead to unreliable primitive extractor. There are various edge detectors available. Since we do not have enough memory for contour images, our expectation from an edge detector is to produce minimal response to a true edge.

Canny's edge detection algorithm is commonly used in the domain of computer vision. Its aim is to produce single response for an edge while maintaining the purpose of detecting all edges in the image. In this study, Canny's algorithm is selected for GPU implementation.

Yuancheng et al. have already implemented Canny's algorithm on CUDA [Yua08]. They have released source code of their implementation. However, their implementation is partial since it makes fixed number of iterations to find the connected components at the last stage of Canny's algorithm. They have tested their edge detector implementation on Lena, and reported limited improvement on edges for more than four iterations to find connected components. As it can be seen from figure [3], one needs more adaptive implementation to detect all edges.

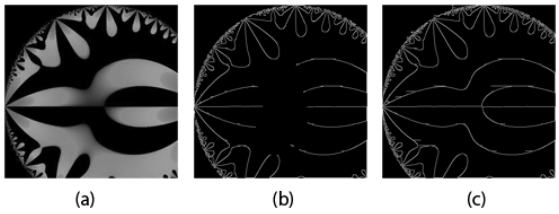

Figure 3. Original 512x512 image (a), edge image after 4 iterations (b), after 90 iterations

In our implementation of Canny's algorithm, we have used parallel breadth first search algorithm to detect connected edges, proposed by Pawan et al. [Paw07]. This adaptation has also accelerated Yuancheng's implementation, which is illustrated in figure [5]. Details of algorithm to find connected components are given in figure [4].

```
1 : For each definite edge do in parallel
2 :     Set frontier [ edge ] <- True , visited [ edge ] <- False
3 : End For
4 : For each edge do in parallel
5 :     If frontier [ edge ] is True
6 :         Set frontier [ edge ] <- False , visited [ edge ] <- True
7 :         For each neighbour,potential, and not visited edge
8 :             Set frontier [ not visited edge ] <- True
9 :         End For
10 :    End If
11 : End For
```

Figure 4. Algorithm to find connected edges on graphics processing unit.

Individual representation

There are different proposals for chromosome encoding of an individual for geometric primitive extraction. Yao et al. suggested encoding minimal number of points to uniquely define geometric primitive [Yao05], whereas Lutton et al. proposed storing parameters of the equation of geometric primitive [Lut94].

Both approaches have their own advantages and disadvantages. The former needs re-computation of parameters of primitive on each fitness evaluation, while keeping search focused on existing edge points and primitives. The latter, on the other hand, is computationally efficient on fitness evaluation while spending considerable time on non-existent primitives.

Since we have implemented ellipse detection on graphics processing unit, computation of ellipse parameters is negligible compared to time spent on fitness evaluation. Therefore, we have decided to encode a chromosome of an individual with five points. General ellipse equation has five coefficients to be determined. Note that, the latter approach should be considered on more complex shapes, where number of unknowns is large.

Initial population generation

One possible population generation is each thread composing its own individual based on edge points on block's shared memory. A thread can compose an individual by randomly sampling edge points from block's tile. One exceptional case is where there is not enough number of edge points on the tile. In that situation, block is terminated immediately since there is not enough evidence to hypothesize the existence of geometric primitive on the tile. Roth et al. has applied the same scheme of initial population generation, but noted that approach usually leads to ill-defined representation of entire geometric primitive [Rot94].

An alternative generation of initial population is total random point generation irrespective of edge existence on that point as suggested by [Pen99]. Although this approach is less computationally demanding, it does not consider existing edges to compose a candidate geometric primitive. However, number of points on chromosome is insignificant considering the points on the primitive it represents.

We have chosen second approach since first results in bank conflicts between threads. Thus, each thread generates a random chromosome consistent with tile dimensions.

Fitness Evaluation

Most of the proposed methods for fitness evaluation of ellipse simply matches template on the boundaries of it. Variations exist, however, in calculation.

Mainzer suggests punishment of displacement from boundary of an ellipse [Man02]. It is intuitive to distinguish edges near the primitive and the ones far from it. Value of the fitness function is given in eq.(1), where c is 0.7 and E(x,y) is 1 if there exists an edge pixel on point (x,y) 0 otherwise.

$$\sum_{x,y}\left(MAX(E(x+i,y+j)-(|i|+|j|)/c)_{\forall\, i,j}\right) \quad (1)$$

Yao et al. suggest two measures of fitness concurrently converging on the optima [Yao05]. They name these fitness measures as similarity and distance. Similarity determines how much the actual pixels match the perimeter of an ideal complete ellipse. Distance, on the other hand, is a measure of how far or close the actual pattern to the ideal ellipse is.

We have implemented fitness evaluation suggested by Mainzer [Man02]. If we had the capability of storing contour of tile on the shared memory of our GPU, this approach would be less time consuming since contour images can supply the distance from edge. In our implementation, we have checked neighborhood of boundary edge of ellipse. Since most time consuming task of genetic algorithm is fitness evaluation, contour image usage on larger shared memory can directly reduce bank conflicts, effectively increasing bandwidth of transaction between shared memory and co-processors.

Evolution

Selection and diversification dictate the main process of evolution. Selection eliminates individuals of low fitness value, promoting fitter individuals. However, it is important not to cluster all solutions around global optima while using elitism for selection. Since our genetic algorithm implementation divides the population into islands, side-effects of elitism have been eliminated.

Diversification is realized by crossover and mutation. Crossover mates two individuals to produce fitter offsprings. In CUDA, every thread selects her own mate to crossover randomly. We have implemented tournament selection on GPU as in Pospichal's study [Pos09]. However, their adaptation of tournament selection is deterministic since every thread mates with one next to it. In our implementation, we have used RNG to select a mate, which is more intuitive for stochastic algorithms but problematic in terms of bank conflicts.

Although other selection algorithms are possible, they are more computationally demanding on CUDA. For example, roulette wheel selection requires that every thread branch divergently in a loop, which immediately causes divergence of whole warp.

Mutation, in our implementation, simply changes a random bit of a chromosome with low probability of occurring. There are different mutation methods proposed for primitive extraction. Yao et al. suggested localized mutation operator that utilizes a trace tracking algorithm to find potential ellipse [Yao05]. Yin, on the other hand, proposed flipping a bit of a pixel such that it remains in the image [Pen99]. We have adapted Yin's implementation of mutation due to its lower processing requirements.

3. EXPERIMENTS & RESULTS

In this section, we inspect various aspects of our implementation and describe our results. Most of our experiments are focused on comparison of sequential and parallel implementations of ellipse detection.

A moderate improvement on canny edge detector has been achieved in our study. CUDA implementation of Yuancheng has been improved in terms of reliability and efficiency. Figure [5] shows efficiency gained by our adaptation of Pawan's parallel breadth first search algorithm for finding connected edges [Paw07]. Difference between two GPU implementations stems from work distribution among threads. In Yuancheng's implementation, each thread executes its own breadth first search,

while in our implementation every thread is responsible for a single edge.

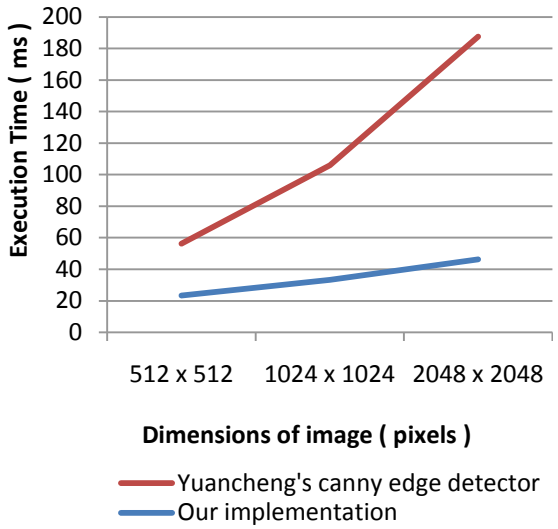

Figure 5. Comparison of our canny edge detector implementation with Yuancheng's implementation on CUDA.

Our ellipse detector is effective against high resolution imagery. Ellipse detection algorithm has been tested with over 600 images contained in our database. CPU implementation has been tested on Intel Core™ i7 at 2.67 GHZ, while GPU implementation has been tested on Nvidia GeForce GTX 260 graphics card. Figure [6] shows us that sequential computational resources could not exhibit scalability accomplished by our parallel implementation of ellipse detection. Note that GPU is not fully utilized for a 512 x 512 image since there are more processors than image segments.

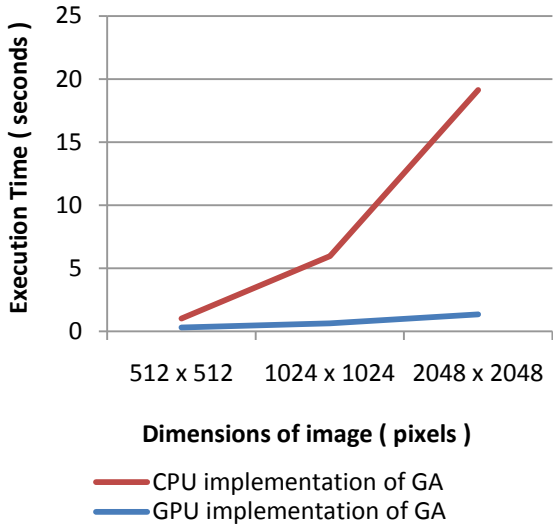

Figure 6. Comparison of sequential implementation of ellipse detection versus parallel implementation on CUDA.

Figure 7. 1024 x 1024 image from our test database (a), image with highlighted ellipse (b).

To test our ellipse detector, we have constructed image database from both synthetic and real world images. Figure [7] shows a real-world image contained in our database along with result of the experiment. To test the accuracy, geometric properties of contained ellipses have been recorded. Table [1] shows statistical results obtained from ellipse detection experiments on GPU.

As input size is multiplied by four, average running time is doubled for small inputs. The reason is under utilization of GPU on low resolution images. Also note that, accuracy of our computation decrease as we increase number of ellipses. This is the result of clustering on global optima. Although we have adapted multiple island model of genetic algorithm, test cases where more than one solution falls into single island did not produce accurate results. To overcome this problem, variations of genetic algorithm, such as sharing genetic algorithm, can be implemented on CUDA.

Dimensions of Images (pixels)	Number of Images	Average Number of Ellipses	Average Running Time (s)	Accuracy (%)
512x512	124	1.3	0.311	82.4
1024x1024	241	2.4	0.627	81.9
2048x2048	248	5.2	1.340	77.3

Table 1. Statistical results obtained from experiments on our ellipse detector.

4. CONCLUSION

In this study, we have shown an implementation of geometric primitive extraction on graphics processing unit. Genetic algorithm has been fully utilized on GPU side, while CPU's computation time is saved. We have achieved up to 15x speed up relative to our sequential implementation of genetic algorithm on state of the art Intel CPU. Main

problem of our GPU implementation is low shared memory per multiprocessor. Usage of bit strings to represent image has produced bank conflicts during fitness evaluation, which is the most costly process of genetic algorithm. Advantage of our sequential implementation is the use of contour images to store edges. Such a data structure to store edge images is expected to accelerate fitness evaluation on GPU side. But current memory limitation has forced us to use bit strings. This problem might be alleviated by image compression techniques. Hardware solutions, on the other hand, are also possible for this kind of problem. As graphics processing hardware scales rapidly, more efficient ellipse detector can be implemented easily.

Experimental results confirmed the trend in parallelization of algorithms in the domain of computer vision. Scalability of current parallel architectures transforms many domains, including computer vision, into an era of parallel computation. Our study was aimed to contribute to this transformation.

5. ACKNOWLEDGMENTS

The author acknowledges the support of METU-TAF Modsimmer, and thanks Prof. İşler for his rigorous assistance to this paper.

6. REFERENCES

[Tia96] Tianzi Jiang; Song De Ma, Geometric primitive extraction using tabu search, Pattern Recognition, 1996., Proceedings of the 13th International Conference on , vol.2, no., pp.266-269 vol.2, 25-29 Aug 1996.

[Rob98] Robert A. McLaughlin, Randomized Hough Transform: Improved ellipse detection with comparison, Pattern Recognition Letters, Volume 19, Issues 3-4, Pages 299-305, ISSN 0167-8655, March 1998.

[Rot94] Roth, G.; Levine, M.D., "Geometric primitive extraction using a genetic algorithm," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol.16, no.9, pp.901-905, Sep 1994.

[Pen99] Peng-Yeng Yin, A new circle/ellipse detector using genetic algorithms, Pattern Recognition Letters, Volume 20, Pages 731-740, Issue 7, July 1999.

[Kul90] Kultanen, P.; Xu, L.; Oja, E., "Randomized Hough transform (RHT)," Pattern Recognition, 1990. Proceedings., 10th International Conference on , vol.i, no., pp.631-635 vol.1, 16-21 Jun 1990.

[Sic05] Si-Cheng Zhang, Zhi-Qiang Liu, A robust, real-time ellipse detector, Pattern Recognition, Volume 38, Issue 2, Pages 273-287, February 2005.

[Rot93] G. Roth and M. D. Levine, "Extracting geometric primitives," Comput. Vision. Graphics Image Processing: Image Understanding, vol. 58, pp. 1-22, 1993.

[Yao05] Yao J., Kharma N., Grogono P.,A multi-population genetic algorithm for robust and fast ellipse detection , Pattern Analysis & Applications, vol. 8 pp. 149-162,2005.

[Kaw98] Kawaguchi, T.; Nagata, R.-I., "Ellipse detection using a genetic algorithm," Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on, vol.1, no., pp.141-145 vol.1, 16-20 Aug 1998.

[Lut94] Lutton E, Martinez P, A genetic algorithm for the detection of 2D geometric primitives in images. In: Proceedings of the 12th international conference on pattern recognition, Jerusalem, Israel, 9–13 October 1994.

[Yua08] Yuancheng Luo; Duraiswami, R., Canny edge detection on NVIDIA CUDA, Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on, vol., no., pp.1-8, 23-28 June 2008.

[Paw07] Pawan Harish and P. Narayanan. Accelerating large graph algorithms on the gpu using cuda. pages 197–208. 2007.

[Man02] Mainzer T., Genetic algorithm for shape detection, Technical report no. DCSE/TR-2002 06, University of West Bohemia, 2002.

[Pos09] Pospichal P. , and Jaros J. , GPU-based Acceleration of the Genetic Algorithm, from contest GPUs for Genetic and Evolutionary Computation, 2009.

[Won09] Wong M. L. , Parallel multi-objective evolutionary algorithms on graphics processing units, Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, pages 2515-2522, 2009.