Thesis Report

An Alternate to Hawk Eye using the Graphics Processing Unit

By:

Radin Ahmed Ehsan – 12241011

Supervisor:

Professor Zahidur Rahman

Spring 2013

<u>DECLARATION</u>

This is to certify that the Thesis entitled “An Alternate to Hawk Eye using the Graphics Processing Unit” is submitted to the department of Computer Science and Engineering of BRAC University in partial fulfillment of the Bachelor of Science in Computer Science. The Thesis comprises only our original work and due acknowledgement has been made in the text to all other material used.

Dhaka, 28th April, 2013

……………………………………………………..
Signature of the Supervisor

Professor Mohammad Zahidur Rahman
Chairman
Department of Computer Science &Engineering
....

……………………………………………………
Radin Ahmed Ehsan

STUDENT ID:12241011

CERTIFICATE

This is to certify that the Thesis entitled “An Alternate to Hawk Eye using the Graphics Processing Unit”, submitted to the department of Computer Science and Engineering of BRAC University in partial fulfillment of the Bachelor of Science in Computer Science is a record of my own work carried out by me. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

April 28, 2013

TABLE OF CONTENTS

0. Abstract:

The more powerful and advanced our processing units become our hunger for more computation power increases. While we are pushing the Central Processing Unit (CPU) to its last limit in the hope to get more computational power, the Graphical Processing Unit (GPU) is sitting idle most of the times. The GPU is immensely powerful in terms computation power and as most modern day GPU's have multi-core architecture; parallel computing can be performed easily. This paper will look into the prospect of improving Hawk Eye, a technology currently used as decision aid system in cricket and tennis in terms of accuracy. For the system NVDIA GPU has been used on CUDA platform.

Keyword: GPU, GPGPU, CUDA, NVDIA, Hawk Eye, Simulation, Parallel Processing.

1. Introduction:

Hawk Eye which is currently used for decision aid system is an image based curve fitting technology. Multiple cameras are used to feed the system the data on the actual flight path of the ball such that the final flight path of the ball can be predicted.

The Hawk Eye uses a high speed camera with typical frame rate of 106 frames per second to track the ball after it has been released from the hand of the bowler through to the stump. The images received are then used to triangulate and predict the final position of the ball had it not been stopped by the pads. In each frame sent from each camera, the system identifies the group of pixels which corresponds to the image of the ball. It then calculates for each frame the 3D position of the ball by comparing its position on at least two of the physically separate cameras at the same instant in time. A succession of frames builds up a record of the path along which the ball has travelled. It also "predicts" the future flight path of the ball and where it will interact with any of the playing area features already programmed into the database.

The system however fails to provide much accuracy as it does not take into consideration factors that may affect the trajectory. Also the amount of deviation that one bowler have may significantly be larger than any other bowler causing the ball to turn more during the very last part of the flight. Hawk Eye fail to take into consideration the spin of the ball, the environmental factors like wind speed, pitch condition and flight of the ball.

We propose a massive change over the existing technology where we will simulate the trajectory of the ball based on running a set of physical simulations that will take the environmental factors into consideration and thereby will provide a substantially higher level of accuracy to the predictions.

The process which will determine the ball’s trajectory through the pad will utilize the images taken from cameras and will attempt a simulation per frame until the ball crosses the stump. All of the trajectories from each simulation run will be averaged together to provide the most accurate approximation of the ball’s path.

We are focusing on the use of the Graphics Processing Unit (GPU) as a processing platform as this allows us the liberty of utilizing the parallel architecture of the GPU to run all of the per frame simulations in parallel which will allow us to complete the simulation in almost real time. [4] [5]

2. Related Work

The only work done in this exact field is Hawk Eye, but it is patented and not much has been published about it. However papers have been published about image processing and physics simulation. This paper focuses on merging the two and coming up with a result that can be used in the field to provide a more accurate judgmental tool to be used in games like cricket, tennis and more recently football. [1]

The papers on parallelization mostly focus on the performance comparison of running such applications on different configuration machines and the performance difference of running on GPU and CPU.

3. Structure of Solution

The video taken from the high speed video has to be first separated into frames and then used for further processing. For the purpose of keeping the project simple the large numbers of image frames are considered as input which will be concurrently analyzed in the multiple cores of the GPU. These images will be used for calculations.

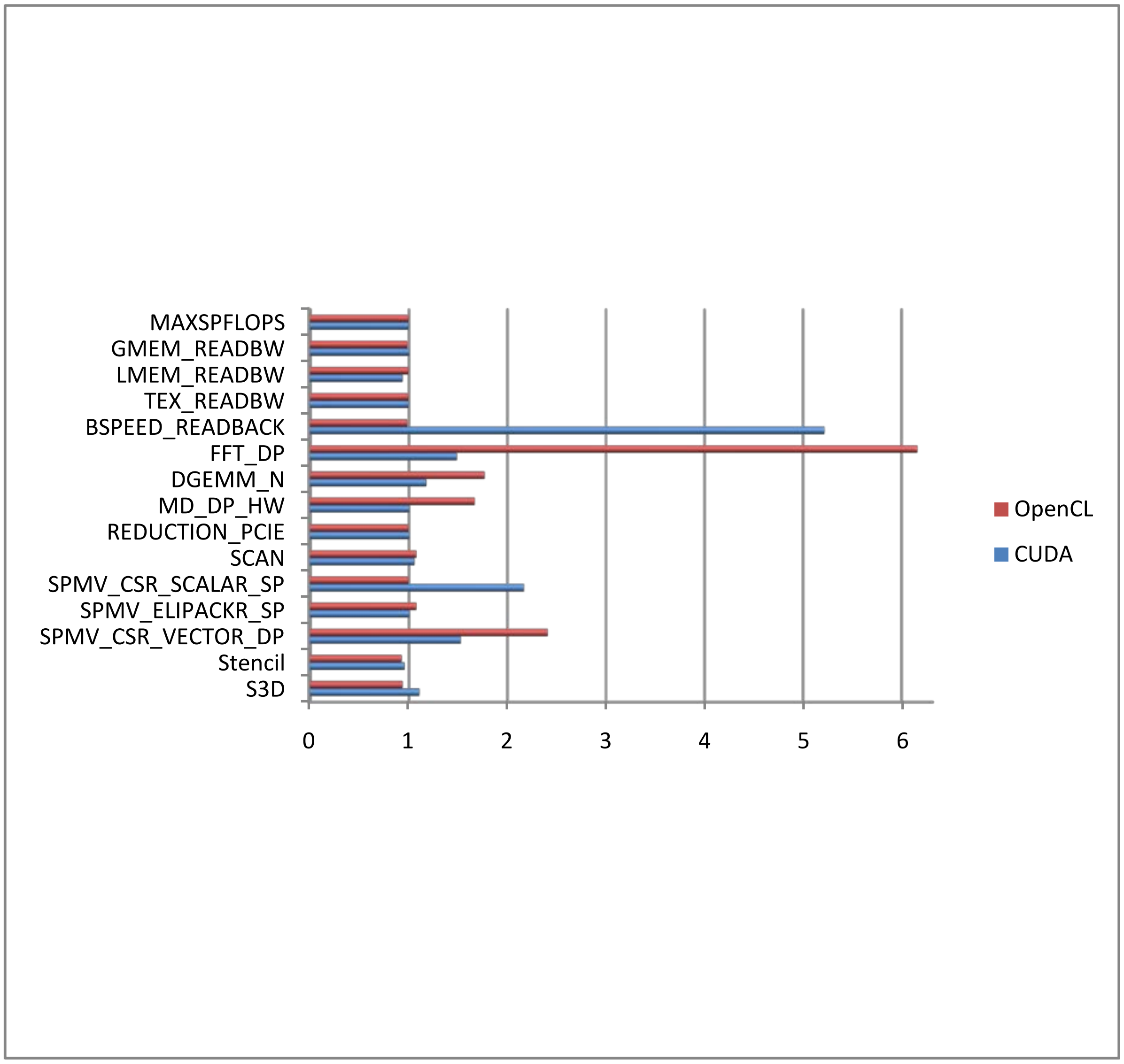
The computation was done on a 64 bits Windows 7 Ultimate desktop with Intel Core 2 Duo, E7400 2.80GHz CPU and NVDIA GTX 560 GPU which supports CUDA.

Visual Studio 2010 was used as default IDE.

3.1 Choice of computation platform

CUDA is the computation platform that will be used with C++ as the main programming language.

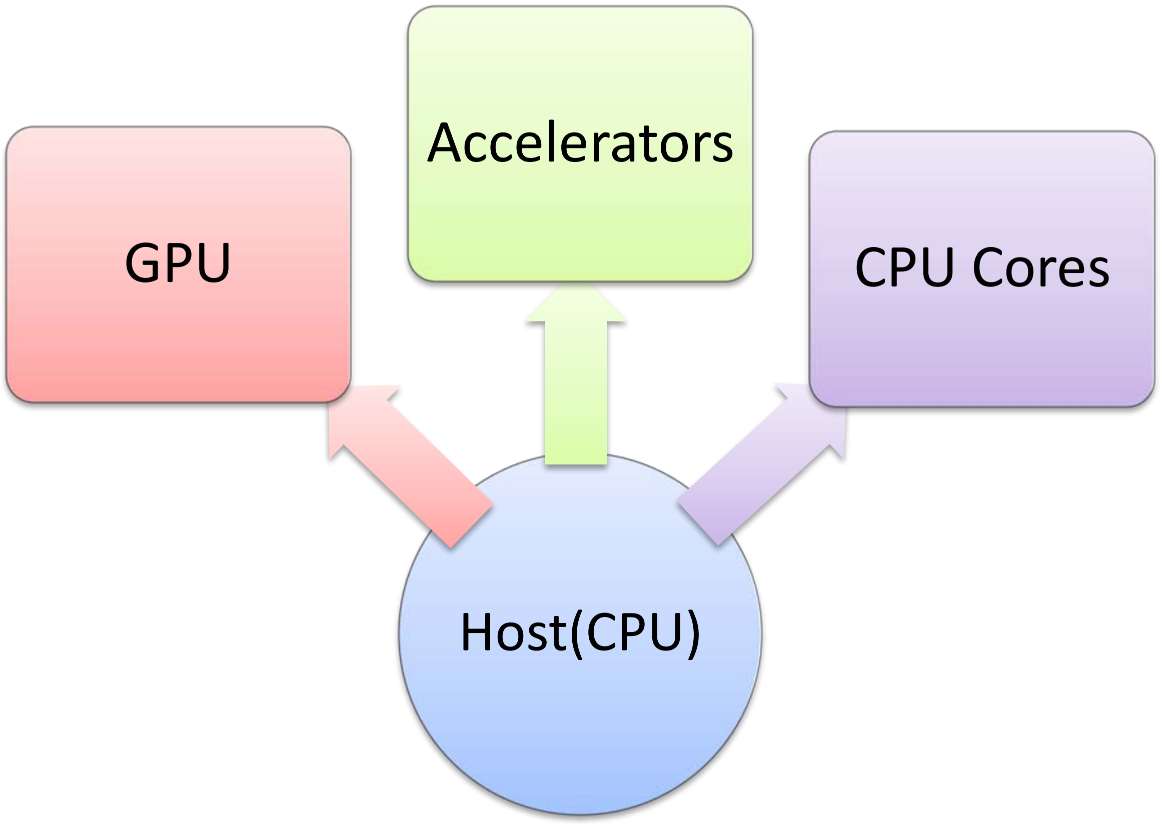
There are two popular platforms for GPU computing available now; one is the Open Computing Library (OpenCL) and the other is the NVDIA Corporations CUDA SDK. The two platforms are very close to each other in terms of performance with CUDA taking the marginal lead. In terms of device compatibility, the OpenCL is much further developed than the NVDIA's CUDA. Unlike CUDA OpenCL supports all programmable graphics processors but CUDA on the other hand only supports NVDIAGPUs'. OpenCL as its name would suggest is open source whereas CUDA is a proprietary SDK by NVDIA Corporation. The chart below demonstrates the performance differences between CUDA and OpenCL in details.



[3]

The reason why CUDA is the choice for this thesis over OpenCL is because CUDA has a slight edge in terms of performance. The documentation of CUDA is much better than that of OpenCL which makes it much easier to learn and pick up. It also makes it faster to start working with since time is a constraint in this thesis. However, if compatibility and openness is concerned, then it is definitely more advantageous to use the OpenCL Application Programming Interface.

OpenCL API as well as CUDA APIs comes with compilers which can compile and execute GPU run able programs and retrieve data from it. It models the programs as a set of input data and a set of instruction data which is sent to the device from the host which does the calculations internally. So essentially, the host is doing nothing but delegating the task to the devices and providing instructions for it and fetching the computed data from the devices which it displays to the users.

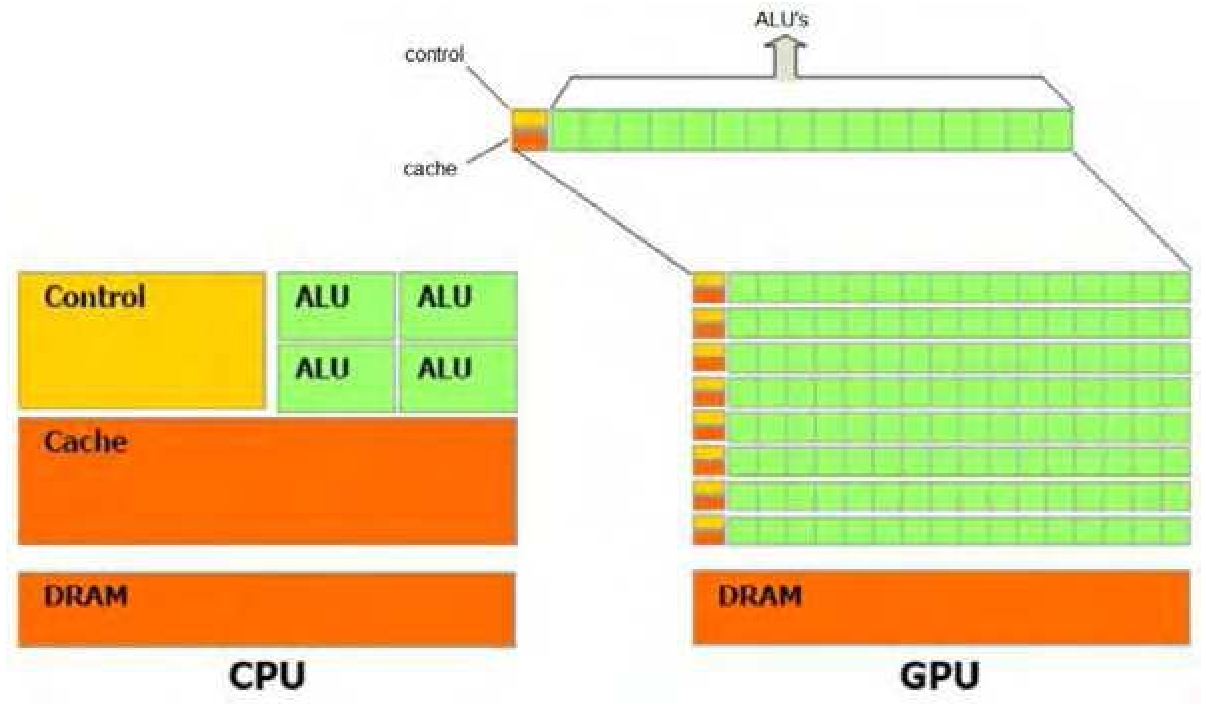


3.2 GPU as a Device to Generic Processing

Over the past 10 years, hitherto, it has seen the evolution of the GPU's as specialized hardware to process graphics and video output, and massive parallel processing of data for general computing. The power of data processing of GPU's has grown much faster than the CPU, and the main reason for this rapid growth of GPU's with respect to the CPU is due to the fact that the GPU's were born with the focus of intensive computing, with respect to data processing and massive parallel computing, as just the minimum requirements necessary to meet the needs of the scenario of computer graphics, like rendering, shadows in 3D scenes and others.

Thus the design of the GPU takes into account the existence of more transistors dedicated to a better process control and data flow, which depicts the main elements: ALU, cache, and DRAM control for a CPU and a GPU.

Many applications that process large data sets organized in a matrix/vector can use a model of parallel computing. In 3D rendering processes large arrays of pixels and vertices are organized so that they can be processed in parallel using threads. Similarly, applications of image processing, encoding and decoding, video scaling, stereo vision, artificial neural networks and pattern recognition can be processed in data blocks and pixels by parallel threads. In fact, many algorithms, even outside the area of image processing, can be accelerated through parallelization of data processing, specially signal processing, simulation of physical effects, computer models of financial or biological applications.

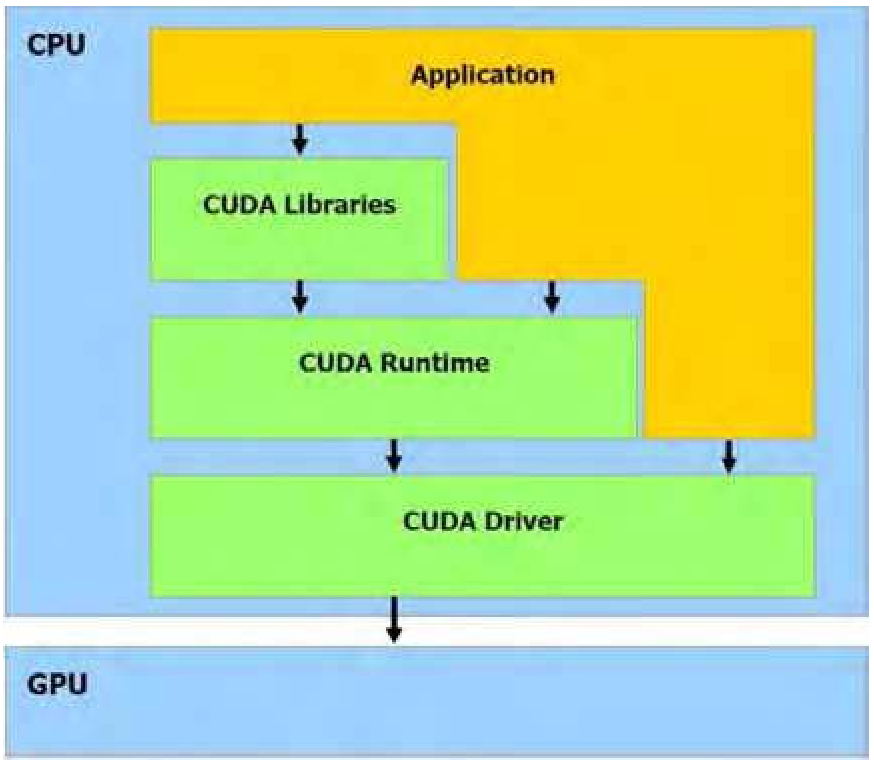


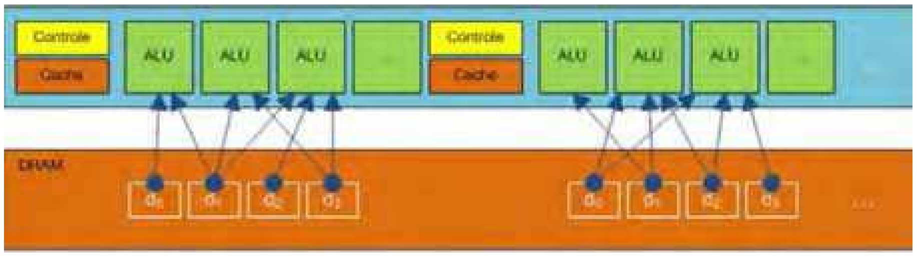
3.3 CUDA - Compute Unified Device Architecture

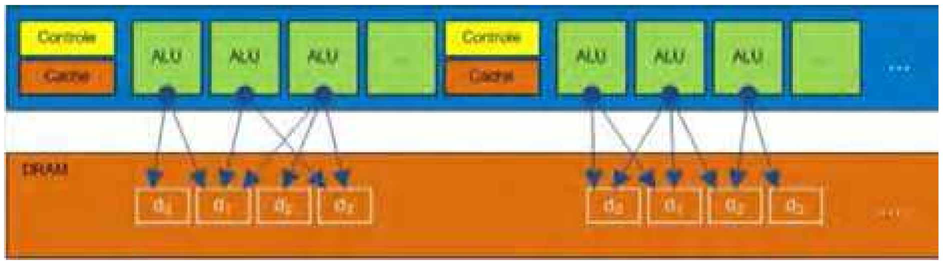
The development of applications that use the GPU as a device for "unconventional" parallel data processing, i.e., not specifically the graphics processing like rendering, is increasing. However the use of a GPU as a device that requires an adjustment of the traditional graphics card pipeline's, forcing the developer to take responsibility for certain control points in these processes, through graphics libraries that have an API for GPU's to become programmable, is annoying.

CUDA is a new architecture of hardware and software that was developed with the main objective of managing the parallel processing of data within the GPU device without the need to make the mapping of the routines and take responsibility for the execution of the pipeline system, through API chart. We have the software stack environment of CUDA, not necessarily for 4 layers of software and these are: (a) application, which is implemented by the browser software that makes use of GPU as a device data processing; (b) CUDA Library is a set of mathematical libraries, such as CUBLAS, an extension of a BLAS library functions algebra implemented in FORTRAN and CUFFT's a fast Fourier transform of 1, 2 and 3 dimensions; (c) where the CUDA runtime routines of other graphics libraries like OpenGL and DirectX are accessed to be processed on the GPU; and (d) CUDA Driver API that is the direct communication with the GPU. In order to facilitate the development of computing solutions for general purpose, not just graphic, CUDA provides the GPU direct memory access to both writing and for reading, just as a conventional CPU works.

The data is read from or written to memory by the ALUs. In this architecture there is a parallel data cache and a shared memory, which has a high-speed access for both, writing and reading. The applications benefit from this structure by minimizing over fetch and round-trips of DRAM and reduce the need/dependence on the bandwidth of DRAM access.







[10]

3.4 Image Processing

The image processing provides a substantial bottleneck in the parallelization process as using the Open Computer Vision (OpenCV) library does not allow for any GPU parallelizable algorithms to be used alongside their provided pattern recognition algorithms. In order to overcome this bottleneck, we have come up with a conceptual algorithm which improves upon the performance of the pattern recognition techniques used in the OpenCV library.

The algorithm we have come up with supplies the background subtracted image to the GPU and provides a color threshold as the parameter. The GPU goes through all the pixels and marks the regions which most closely fit the provided color threshold. The image data is retrieved and the marked areas are made to fit inside a circular region. The region which most closely fits the shape is selected as the ball. This is a brief overview of the process-

» Background Separation Carried out in the CPU

» Image Regions marked in GPU on a per-pixel basis
» Noise Reduction applied as a Gaussian Blur
» Image region selected based on closest area match

This process speeds up the detection of the regions but still gets bottlenecked during background separation and the final fitting. The third and fourth phase is only necessary since there is no means of gathering data from other GPU cores during the process. Since getting the final shape requires the merging of all the regions and finding the out most closely matched shape, we cannot skip the last phase and as such will have to face the bottleneck, however, the O(n) process of iterating through the pixels of the image can be turned into O(k) time thanks to the massive parallelization of the GPU which provides a boost in the response time. As for the noise reduction, it is currently being carried out during the third phase in the CPU, however, we are working on getting it within the GPU so that this O(n2) process can be improved in terms of performance.

Once the ball has been detected, the position of the ball is approximated by using the stumps as a relative standard and the displacement of the ball is approximated per frame by checking the difference of the ball from the previous frame. To obtain the 3D positioning of the ball at least 2 different camera images need to be processed. One camera needs to take the image from the Z axis which provides the X and Y values of the ball's position and the other camera needs to take the Image from the Y axis which provides the Z and X positions of the ball. Using the X axis as the standard axis, we can scale the Y and Z values to more accurately match the actual position of the ball.

Obtaining the position and the velocity is the rather easy part which can be achieved just by simple pattern detection. However, in order to detect the spin, we need to have a special pattern imprinted in the ball which can be used as a reference to obtain the spin velocity and the direction of spin of the ball. In order to detect and define the pattern, we need to re-detect the image with a stricter threshold so that the pattern inside the ball is detectable. The same pattern detection process is applied again on the extracted region and the pattern is detected. It is then compared with the previous frame’s pattern and the difference is calculated to be the angular velocity of the ball which is the spin velocity as well.

3.5 Physical Simulation

Given the massive parallelization capabilities of the GPU the best way to utilize the power is by running multiple instances of the same simulation at the same time. The physics behind the trajectory is the same as the physics of the projectile. The other factors of consideration are the spin and the swing of the ball.

We will be using equations to find out the trajectory of the ball. For a successful prediction we need to know the following:

1) Where the ball will first drop.
2) What will be the height of the ball when it makes impact with the batsman.

All other variables which include wind speed, pitch condition are required to calculate the spin and swing.

For finding out the distance on where the ball will first drop, we will use:

$$d = \frac{v\cos\theta}{g}\left(v\sin\theta + \sqrt{(v\sin\theta)^2 + 2gy_0}\right)$$ [7]

The length of the entire pitch is known to be 22 yards which is equivalent to 20.12 meters. If we subtract the value of d from 20.12 meters we will be able to find out the distance the ball has to cover before making the impact. From that we can figure out how long the ball will be at flight from the time it has dropped for the first time till it makes impact with the pad. This is done using this formula:

$$t = \frac{d}{v\cos\theta} = \frac{v\sin\theta + \sqrt{(v\sin\theta)^2 + 2gy_0}}{g}$$ [7]

The height of the ball while making impact is found using this formula.

$$y = y_0 + x\tan\theta - \frac{gx^2}{2(v\cos\theta)^2}$$ [7]

From the equations it is understandable that the formulas and equations that are to be solved do not require any iteration making it atomic. The attached code is a simple model of this trajectory calculation made in java which is showing that the computation is of constant time. Thus we will not be able to take advantage of the multiple core of the GPU. For this we will run simulation of multiple instances at the same time so that we can harness the power of the GPU.

The simulations that are being run will each give result. We will use these results and compare it with the original trajectory that we have till the pad. The simulation that is the closest with the original trajectory will be considered to be the most accurate and that will be our prediction.

4. Pitfall

- Spin and Swing: While working with the simulation of ball we figured out that there is not any single equation for finding out the spin of the ball. Therefore it is hard to predict where the ball will make its impact in case of spin bowlers. For pace bowlers there is no equation for finding out the swing of the ball which makes it difficult to predict the impact. Since our computation power is not limited, we will have a brute force approach for finding out the spin and the swing of the ball.

- Image Processing: Although the ball is divided into two parts with a seam in the middle, but it is monochromatic which will makes it difficult for us to find out the angular movement of the ball. If one side of the ball was marked with a cross, it might have been easier for us to track the angular movement.

  The background does not stay static as the game is progressing which makes it difficult to separate the ball. The heads of the players are spherical in shape which increases the complexity of finding the ball.

- Algorithmic Complexity: There are issues of bottleneck with the physical simulation since the physics simulation algorithm is of complexity O(k). Advantages of multi-core architecture can only be taken if the complexity is of O(n) or higher.

- Independence: We have to ensure that the simulations we are running are independent of each other if we are to utilize the full power of the GPU [4].

- Reverse Swing: Finding out the swing and reverse swing that the bowler produces will be a very tough job as the phenomenon is not fully understood yet by researchers. [6]

- Frame Separation: Due to the lack of time the video feed could not be separated frame by frame. Most open source tools can only divide frame for camera speed up to 60fps whereas the typical frame rate of cameras used in Hawk Eye is 100+. [2]

5. Conclusion

The future of computing is about parallelization. It is difficult for the CPU to handle so much processing whereas the GPU sits idle most of the time. Therefore it would be wise to delegate the processing of CPU to the GPU to ensure smoother and better processing in minimal time.

The solution is always scalable and can be mapped to other problems of similar nature.

6. Future Work

In the future I will try to implement this in cloud and try to gain more knowledge so that the physics of spin, swing and reverse swing can be added in to the system making it a full solution to the problem that exist with the existing Hawk Eye System.

7. Reference

[1]: Hawk-Eye. http://en.wikipedia.org/wiki/Hawk-Eye

[2]: The Ball Tracking Chronicle. http://cricketingview.blogspot.com/2011/07/ball-tracking-chronicle.html

[3]: OpenCL Gains Ground On CUDA. http://www.hpcwire.com/hpcwire/2012-02-28/opencl_gains_ground_on_cuda.html

[4]: R. D. Chiara, U. Erra, V. Scarano, and M. Tatafiore. Massive simulation using GPU of a distributed behavioral model of a flock with obstacle avoidance. In B. Girod, M. A. Magnor, and H.-P. Seidel, editors, VMV, pages 233–240. Aka GmbH, 2004.

[5]: Shane Ryooy, Christopher I. Rodriguesy, Sara S. Baghsorkhiy, Sam S. Stoney, David B. Kirk and Wen-mei W. Hwuy. Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. In proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming.

[6]: David Jamesa, Danielle C. MacDonaldb, John Harta. The effect of atmospheric conditions on the swing of a cricket ball. Procedia Engineering. Volume 34, 2012, Pages 188 - 193.

[7]: Trajectory of a projectile. http://en.wikipedia.org/wiki/Trajectory_of_a_projectile

[8]: What Is CUDA. http://developer.nvidia.com/cuda/what-cuda

[9]: OpenCV. http://opencv.willowgarage.com/wiki/

[10] Gustavo Poli and José Hiroki Saito: Parallel Face Recognition Processing using Neocognitron Neural Network and GPU with CUDA High Performance Architecture