Broad Phase Collision Detection using Multi-core Processor

Norhaida Mohd Suaib
UTM VicubeLab, Faculty of Computing,
Universiti Teknologi Malaysia, Johor, Malaysia,
81310 UTM Skudai
haida@utm.my

Fawwaz Mohd Nasir
Faculty of Computing,
Universiti Teknologi Malaysia, Johor, Malaysia,
81310 UTM Skudai
fawwaz_bmn@yahoo.com

Abstract—**Collision detection is a very important component in computer graphics applications. However, due to its high algorithm complexity, collision detection usually forms a bottleneck in many of these applications causing the simulation performance to deteriorate. Earlier algorithms for collision detection are sequential in nature. The multi-core processor technology is seen as an opportunity to reduce and eliminate this bottleneck by parallelizing the collision detection algorithm. Therefore, this paper implements the sphere bounding volume in the broad phase collision detection using the sequential and parallel approach separately, in order to identify the simulation performance differences between both approaches. The algorithm used to implement the broad phase collision detection involved the all-pair test where it is based on the comparison of the objects' bounding volume to determine if collision occurs. As an extension, this paper utilizes the graphics processing unit to implement the parallel approach. The implementation of the broad phase parallel collision detection shows improved frame rate for larger number of objects involved up to 1.2 x faster compared to the sequential implementation.**

Keywords-component; collision detection; broad phase; parallel computing

I. INTRODUCTION

Collision detection refers to the problem of determining whether moving objects collide, in other words, whether the corresponding shapes intersect [1]. Without collision detection, objects can penetrate each other. In order to efficiently implement collision detection, a two-phased approach is used which are the broad phase followed by the narrow phase [2]. The first phase which is the broad phase is responsible for the quick and efficient removal of the object pairs that are not in collision [3]. The narrow phase on the other hand, performs tests in more detail and is performed on objects that have the potential to collide [2]. Tests made during the first phase use the traditional way of testing collisions that are based on bounding volumes. It is one of the bottlenecks in real time rendering loop [4].

Hence, this paper focuses on the implementation of the broad phase approach of the collision detection using the all-pair test. The sphere is used as the bounding volume to perform collision tests. Graphics processing unit (GPU) programming using the Compute Unified Device Architecture (CUDA) is applied in this paper to determine the differences when the sequential and parallel algorithm is executed in the implementation of the collision detection. An NVIDIA GeForce G105M graphics card was used for the experiments.

This paper is organized into six different sections. The first section gives an in-depth discussion on the issues relating to the traditional collision detection. This is followed by the research framework in the second section. The experimental layout is discussed in the third section. The results and discussions are presented in fourth section and the final section concludes this paper.

II. TRADITIONAL COLLISION DETECTION ISSUES

Collision detection refers to the process of determining if two objects collide with each other. It is a very important component in computer graphics applications. However, it remains as a fundamental problem since it forms a bottleneck in many of these applications. Earlier algorithms for collision detection are sequential in nature. They are designed for single core processor. One of those sequential or traditional algorithms is the brute-force algorithm. This algorithm tests every object for collision. It has an algorithm complexity of $O(n^2)$. The next algorithm is the sort and sweep algorithm. Although it is simple to implement, it has an average complexity of $O(n \log n)$ and its worse case complexity is $O(n^2)$. Spatial subdivision can also be used as one of the algorithm in broad phase collision detection. This algorithm has the same complexity as to the previous mentioned algorithm where its average algorithm complexity is $O(n \log n)$ with $O(n^2)$ being its worse case algorithm complexity.

Referring to Moore's law, the chip performance would double every 18 months [5]. This means that programs will automatically run faster on newer processors. The design goal for the processors in the late 1990's and early 2000's was to increase the clock rate. This was accomplished by increasing the number of transistors on the smaller chip. Unfortunately, this method becomes unreliable around 2002 because by further increasing the number of transistors it causes the increase in power dissipation of the processor chip, beyond the capacity of inexpensive cooling techniques. Therefore, opportunities for improving the raw performance of individual

processor has become very limited because of this problem which is commonly known as the power wall [6].

In order to continue delivering performance improvement and due to cost constrains on the need for inexpensive cooling techniques, the multi-core processors were introduced. A multi-core processor can run more operations and system processers at the same time, compared to a single-core processor. It can complete a complex task in a short period of time. For instance, with a weak processor, decoding a high definition movie and playing it can take up to more than three hours [7]. However, with a dual-core processor, it can be done within an hour besides having the ability to do something else in the background [7]. The number of cores is likely to increase at the rate corresponding to the Moore's law [8]. This means that programs will not get any faster unless the ever-increasing number of cores is effectively utilized. Therefore, software developers are trying to make use of this multi-core technology to improve the performance of their application.

Modern GPUs offer higher peak throughput compared to the central processing units (CPUs) [8]. This has been proven by many organizations where the developers have achieved an increase in performance when using the GPU to perform computations traditionally handled by the CPU. Although not all applications can see this kind of speed-up, but 100 times speed-up and beyond have been seen by hundreds of developers [9]. Therefore, by using the GPU to implement the algorithm for the broad phase approach of the collision detection, it is expected that there will be a performance improvement since CPUs has lower throughput compared to the GPUs. Besides, using the concept of parallelism can also improve the algorithm performance compared to using the traditional sequential approach.

The broad phase collision detection is one of the bottlenecks of the real time rendering loop due to its algorithm complexity. This will eventually lower the frames per second (FPS) since it affects the overall performance of the application. The limited ability in improving individual processor since its interception with the power wall has resulted in the introduction of the multi-core processors. Therefore, this is seen as an opportunity to utilize the multi-core GPU especially to overcome the bottleneck since there was very little study regarding the implementation of collision detection on multi-core processors. GPU is used since it is said to have higher throughput compared to the CPU and better results are expected in terms of FPS when using GPU compared to CPU.

III. RESEARCH FRAMEWORK

Figure 1 illustrates the research framework for the implementation of the broad phase collision detection in sequential computing. The algorithm used is the all-pair test which is a brute force approach of collision detection. The all-pair tests checks for collision between objects by testing whether the objects bounding volume intersect with each other.

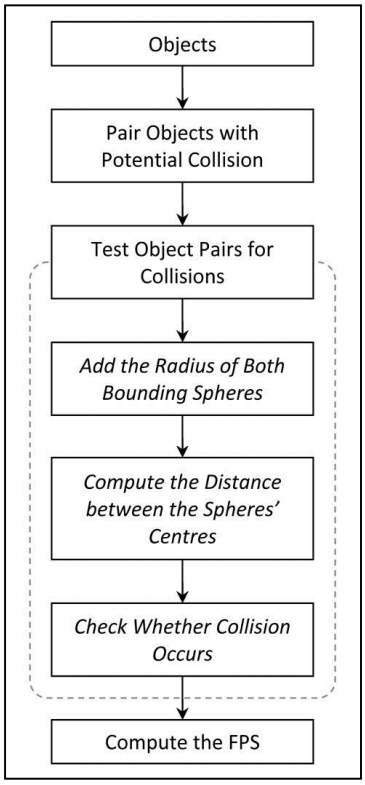

Figure 1. Sequential Computing Research Framework

This paper uses the bounding sphere as the objects' bounding volume. The input of the research framework shown above is the objects. Objects that have the potential to collide with each other are pair up together, creating a list of object pairs. Each pair is then tested for collisions. During the test process, the radius of bounding sphere pair is summed up together. Then, the distance between the pairs' centre is calculated. The calculated values are then compared with each other to check whether the bounding spheres intersect. An intersection between a pair of bounding sphere does not occur if the distance between the spheres' centre is greater than the sum of the radius of both spheres. Else, if is the other way around, then the intersection occurs. By the end of the collision testing, the output, which is the simulation performance measured in FPS for the whole process is calculated. Note that for this sequential computing approach, the object pairs are tested one after another in a sequential manner.

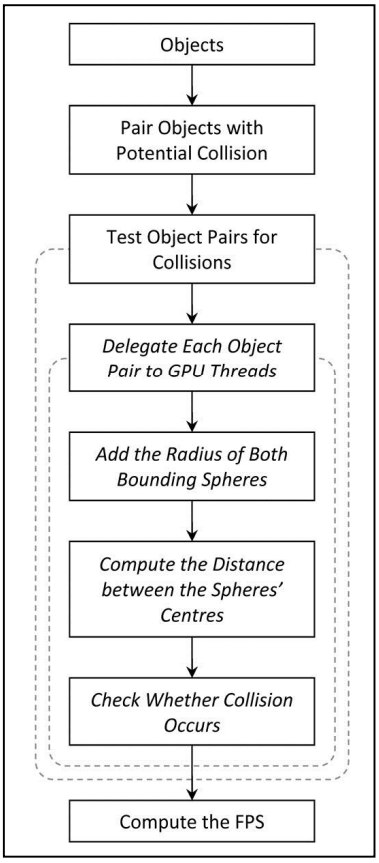

Figure 2. Parallel Computing Research Framework

Figure 2 on the previous page represents the framework for conducting the parallel computing on the broad phase collision detection. The framework is quite similar to the framework for sequential computation. The only difference is that instead of checking for bounding spheres intersection sequentially, intersections are checked concurrently. Meaning, the object pairs created in the earlier step will be tested for collision in a parallel fashion by delegating the job between several GPU threads or cores. Figure 3 shows how the workload, which in our case is the object pairs, is delegated in a CUDA capable NVIDIA GPU. The workload is distributed by first specifying the number of blocks and the number of threads per block needed to complete the task. These numbers depend on the amount of object pairs that will be tested for collision. CUDA will then distribute the workload based on these specified numbers.

To compare the differences between the sequential and parallel approach, comparisons are made on the frame rates from both computation. A frame rate is the frequency where unique consecutive images were produced by the imaging device. In computer graphics, it refers to the speed at which the image is refreshed. Usually frame rates were measured in seconds. The higher the frame rate, the smoother the motion image being displayed. A lower frame rate causes the motion image to look choppy or jumpy. Therefore, in our case, parallel computing is expected to produce a higher frame rate when compared to the sequential approach.

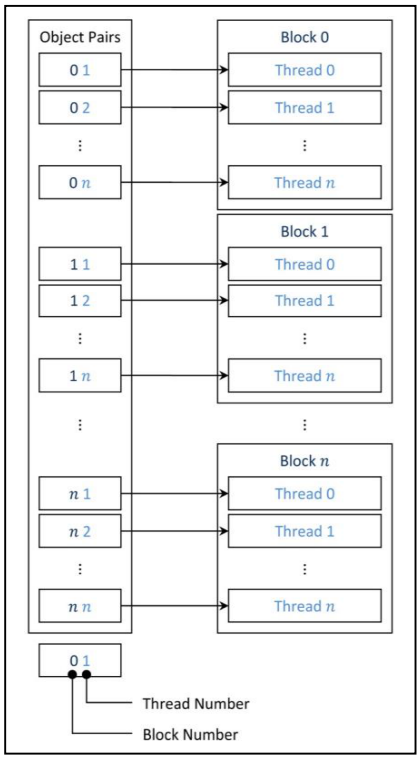

Figure 3. Workload Distribution

The program for this paper is written in C++ programming language and is developed using an integrated development environment (IDE) named Microsoft Visual Studio 2008. The Open Graphics Library (OpenGL) application programming interface (API) is used to render the scene. Besides, NVIDIA CUDA Toolkit 5.5 is also used to allow direct access to the GPU. Currently, the toolkit only supports Visual C++ 9.0 compiler (part of Microsoft Visual Studio 2008 IDE) or later for programs written in C++ and developed in the Microsoft Windows operating system. Apart from that, this paper utilizes the NVIDIA GeForce G105M processor which is one of NVIDIA's CUDA capable GPUs.

IV. EXPERIMENTAL LAYOUT

The experiment is conducted by adding different number of spheres to the scene. The chosen set for the number of spheres is 64, 128, 192, 256 and 320. Also, besides using different number of spheres, different approaches were also used to conduct the intersection test. The set of numbers mentioned was tested with these approaches, which are the sequential and parallel approaches. A set of 200 FPS readings is then recorded into a text file by a user triggered event. Separate files are created to record the readings for both the sequential and parallel approaches. The readings will be summed up together and will be divided by the total number of FPS records to get the average. This process is performed in order to increase the

precision of the FPS value. Note that the greater the number of FPS readings, the higher the precision.

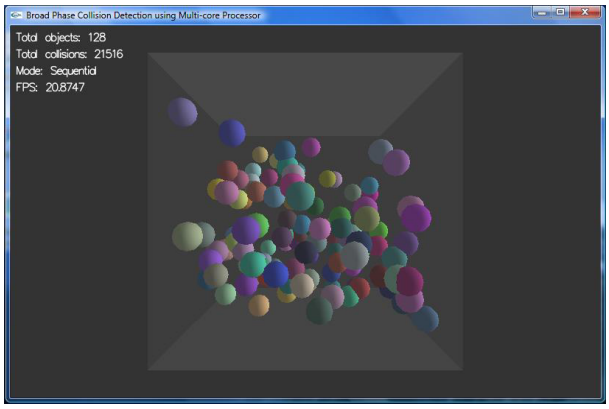

Figure 4. The Scene

V. RESULTS AND DISCUSSIONS

Table 1 shows the calculated average of the recorded real-time FPS for both the sequential and the parallel approaches. The effects of using different number of spheres were also documented in the table. The number of spheres used is actually a multiple of 16. In CUDA, the smallest executable unit of parallelism is 32 threads, which is called a wrap [10]. The graphics card used performs memory transfers and instruction dispatch at half-wrap or 16 threads. Therefore, in order to maximize the graphics card occupancy, multiples of 16 is used. In general, from what is shown in the table and Figure 5 below, the parallel approach is faster than the sequential approach especially when the number of spheres is higher.

TABLE I. AVERAGE FPS FOR DIFFERENT NUMBER OF SPHERES AND APPROACHES

Number of Spheres	**Average FPS**		**Speed-up**
	Sequential Approach	***Parallel Approach***	
64	59.31	36.49	0.6152
128	19.43	18.39	0.9465
192	9.354	9.730	1.041
256	5.359	6.198	1.157
320	3.420	4.342	1.270

The amount of speed-up gained in the parallel approach when compared to the sequential approach was also calculated. In parallel computing, speed-up refers to how much faster the parallel approach is compared to the sequential approach. It is calculated by dividing the average FPS of the parallel approach by the average FPS of the sequential approach. Notice that from Table 1 shown above and Figure 6 on the next page, this value increases as the number of spheres increases. This shows that the parallel approach works better when there are higher numbers of spheres. In other words, the GPU is at its best performance if there are higher numbers of tasks to be performed.

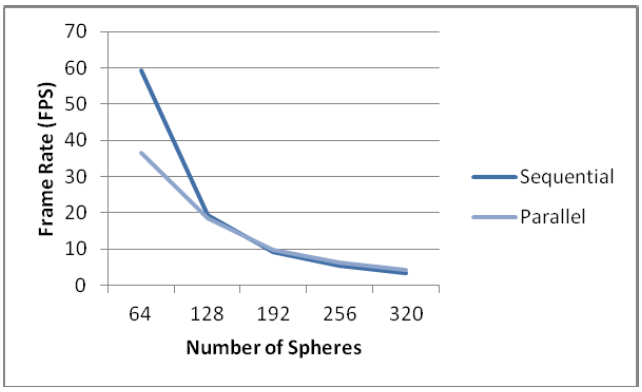

Figure 5. Sequential and Parallel Approach Frame Rate

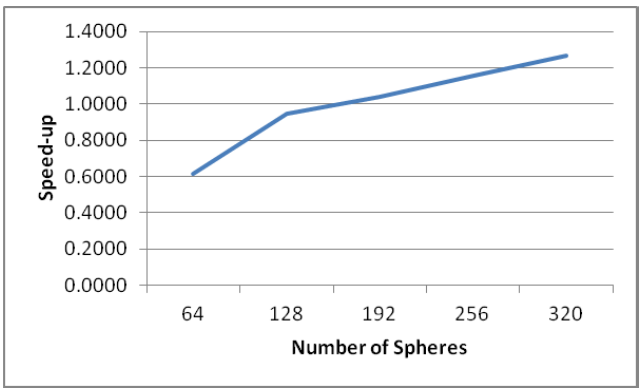

Figure 6. Parallel Approach Speed-up

VI. CONCLUSIONS

This paper has investigated the implementation of the broad phase collision detection in both sequential and parallel approaches. As mentioned earlier, the purpose of this paper was to investigate the difference between executing the broad phase collision detection algorithm in the sequential and parallel approaches. Since the processor architecture hit the power wall, the multi-core processors were introduced to continue delivering performance improvement. Therefore, in order for the applications to run faster, they need to utilize the multi-core technology. The performance of the computer graphics applications, especially that involves collision detection can be improved since collision detection forms a bottleneck in many of these applications. Generally, from the outcome of the conducted test, the parallel approach works better than the sequential approach when there are higher amount of work that needs to be completed. Also, the amount of speed-up correlates to the amount of spheres added to the scene.

ACKNOWLEDGMENT

This research is supported by Universiti Teknologi Malaysia (UTM) and Ministry of Higher Education (MOHE), in collaboration with Research Management Centre (RMC), UTM, and partly supported by Research University Grant (RUG) Program, Tier 1 (research grant 09H18) and RUG Tier 2 (research grant 05J21).

REFERENCES

[1] Klawonn, F. *Introduction to Computer Graphics: Using Java 2D and 3D*. London: Springer. 2008.

[2] Grand, S. L. Broad-Phase Collision Detection with CUDA. In: Nguyen, H. ed. *GPU Gems 3*. Boston: Addison-Wesley. 697 – 721; 2008.

[3] Avril, Q., Gouranton, V. and Arnaldi, B. A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures. *Proceedings of Virtual Reality International Conference (VRIC 2010).* April 7 – 9, 2010. Laval: Laval Virtual. 2010. 95 – 100.

[4] Malmsten, M. and Klasen, S. Practical Collision Detection on GPU: A Case Study Using CInDeR. Retrieved February 27, 2013 from http://fileadmin.cs.lth.se/graphics/ theses/projects/gpucd/report.pdf. n.d.

[5] Stallings, W. *Computer Organization and Architecture: Designing for Performance (Eighth Edition).* New Jersey: Pearson. 2010.

[6] Karras, T. Thinking Parallel, Part 1: Collision Detection on the GPU. Retrieved February 27, 2013 from https://developer.nvidia.com/content/thinking-parallel-part-i-collision-detection-gpu. 2012.

[7] Paul M. Why dual-core is faster than single-core processors. Retrieved March 17, 2013 from http://www.helium.com/items/1021144-why-dual-core-is-faster-than-single-core-processors. 2013.

[8] Tang, M., Manocha, D., Lin, J. and Tong, R. Collision-Streams: Fast GPU-based Collision Detection for Deformable Models. *Proceedings of I3D '11 Symposium on Interactive 3D Graphics and Games.* February 18 – 20, 2011. San Francisco: ACM. 2011. 63 – 70.

[9] Keane, A. "GPUs are Only Up to 14 Times Faster than CPUs" Says Intel. Retrieved March 24, 2013 from http://blogs.nvidia.com/2010/06/gpus-are-only-up-to-14-times-faster-than-cpus-says-intel/. 2010.

[10] NVIDIA Corporation. *CUDA C Best Practices Guide: Design Guide.* n.p.: NVIDIA Corporation. 2013.