International Journal of Innovative Computing

Journal Homepage: http://se.fsksm.utm.my/ijic/index.php/ijic

BROAD PHASE COLLISION DETECTION USING GRAPHICS PROCESSING UNIT

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

Fawwaz Mohd Nasir
Faculty of Computing, Universiti Teknologi Malaysia,
UTM Skudai, Johor,
Malaysia

***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 1.2x faster compared to the sequential implementation.**

Keywords — Collision detection, broad phase, bounding volume, parallel computing, GPU programming.

I. INTRODUCTION

Collision detection for computer graphics applications refers to the problem of determining whether there are any intersections between virtual objects [1] and to report any collisions [2] so that they could be handled accordingly. It is an important component in computer simulated environment to ensure that virtual objects behave in a physically-plausible manner. Without collision detection, objects can penetrate each other and can lead to physically incorrect behaviour. In order to efficiently implement collision detection, a two-phased collision detection approach is frequently adopted which comprises of the broad phase followed by the narrow phase collision tests [1, 3-8]. The first phase which is the broad phase collision detection is responsible for the quick and efficient removal of the object pairs that are not in collision [5]. The narrow phase on the other hand, performs tests in more detail and is performed on objects identified as having the potential to collide in the broad phase level. Collision detection is one of the bottlenecks in real time rendering loop [1, 9, 10]. A conventional way to accelerate collisions tests during broad-phase collision detection is based on bounding volumes approach [1, 11].

Hence, this paper focuses on the implementation of the broad phase collision detection using the all-pair test. Sphere bounding volume was used to perform collision tests. Graphics processing unit (GPU) programming using the Compute Unified Device Architecture (CUDA) was applied in this research to determine the differences between the sequential and parallel algorithm executions in the implementation of the collision detection process.

This paper is organized into six different sections. The first serves a brief introduction to the topic. The second section discusses on the background of this research. This is followed by the research framework in the third section. Experimental layout is discussed in the fourth section. The results and discussions are presented in fifth section and the final section concludes this paper.

II. BACKGROUND OF THE RESEARCH

In a dynamic environment involving multiple interacting objects, collision detection is known for its computationally intensive process. Different approaches have been proposed to improve the process, for example by culling away non-interacting objects during the broad-phase collision detection level or the use of specific data structure such as the bounding volume hierarchy (BVH) or space decomposition.

A two-phase collision detection approach is usually employed in a dynamic simulation involving n-body objects to reduce the total number of pairwise collision tests. This involves the broad-phase level and narrow-phase collision detection level. More than often, the 3D collision tests are reduced into suitable parallel problems; for example by projecting the 3D bounding volume information onto the X, Y and Z-axis respectively and feeding the reduced 1D collision tests into parallel threads. The next sub-section will discuss related issues regarding collision detection and its adaptation into parallel implementation.

A. 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, especially in if it involves dynamic objects. However, collision detection remains as a fundamental problem since it forms a bottleneck in many of these applications [1, 6, 10]. In order to ensure that the application can run smoothly, various approaches has been introduced to reduce the cost involved; particularly on collision tests. One of them is the two-phase collision detection scheme that was mentioned earlier. Different approaches can be adopted during the broad-phase level; generally they fall under either object-space approach, or image-space approach. The bounding volume approach is one of the examples of object-space approach and it can further be used to subdivide the objects into a bounding volume hierarchy (BVH). This is known as object decomposition. On the other hand, the whole scene can also be decomposed into smaller 3D grids and this is known as scene decomposition. An example of scene decomposition is by using octree. Figure 1 shows the general approaches that were mentioned here.

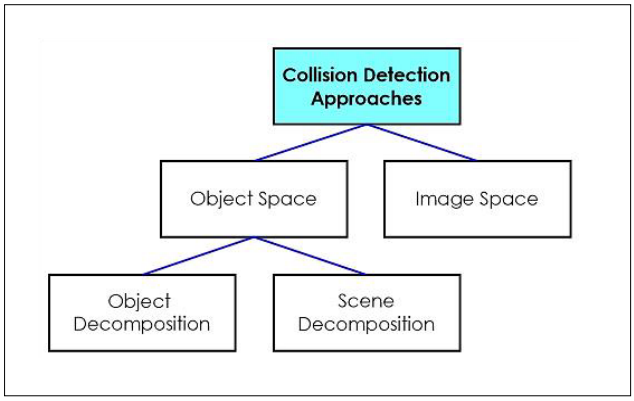

Fig. 1. Collision detection approaches.

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 worst 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 worst case algorithm complexity.

B. Hardware Development

Referring to Moore's law, the chip performance would double every 18 months [12]. 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 approach quickly becomes unreliable - further increase in the number of transistors 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 [13].

In order to continue delivering performance improvement and due to cost constraints 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. The number of cores is likely to increase at the rate corresponding to the Moore's law [14]. However, multiple core processors can only be beneficial for problems that were carefully adapted for parallel, core-limited tasks [15]. 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) [14]. 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. 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.

C. Adaptation of Parallel Collision Detection

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. Earlier algorithms were mostly implemented on the CPU, and later involving the use of GPU without direct GPU manipulation [16]. Sequential pairwise checks that are traditionally done between a large number of interacting objects usually consume huge amount of resources, but the nature of pairwise collision checks is readily adopted for parallel implementation [17]. Parallel collision detection implementation were investigated along with the introduction of the multi-core CPU and later extended to GPU implementation.

Parallel BVH construction and computations were normally used for collision detection between rigid and deformable objects [14, 18, 19], rendering-related processes such as ray-tracing [10, 20] and visibility culling. Parallel broad-phase CD was also investigated for collision checking as part of sampling-based path planning [21, 22]. A framework for the fast construction of surface area heuristic (SAH)-based BVHs that is particularly designed for the Intel multi-core architecture for ray-tracing purposes has been proposed by [20]. In this case, CPU implementation was chosen since it dealt with control-intensive tasks such as BVH construction rather than compute-intensive tasks.

Multiple core GPUs have been exploited for parallel collision detection and distance computation for rigid and deformable objects [19], bounding volume hierarchies (BVH) construction and related computation [23]. It offers many advantages especially if the scene involves a large number of dynamic and interacting objects where collision culling based on GPU helps to reduce collision tests [24, 25].

As discussed above, the choice of either to use parallel implementation on the CPU and GPU has been explored by a number of researchers. It was mentioned that the parallel CPU implementations were more suitable for control-intensive tasks while the parallel GPU implementations were more suitable for compute-intensive tasks [26]. GPU is used for compute-intensive tasks since it is said to have higher throughput compared to the CPU and higher speed for real-time applications (in terms of frames per seconds (FPS)). Therefore, it is seen as an opportunity to utilize the both multi-core CPU and GPU as can be seen from hybrid CPU/GPU spatial subdivision collision detection technique [18] and algorithm that is suitable for both multi-core CPU and many core GPU [27].

III. RESEARCH FRAMEWORK

Figure 2 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.

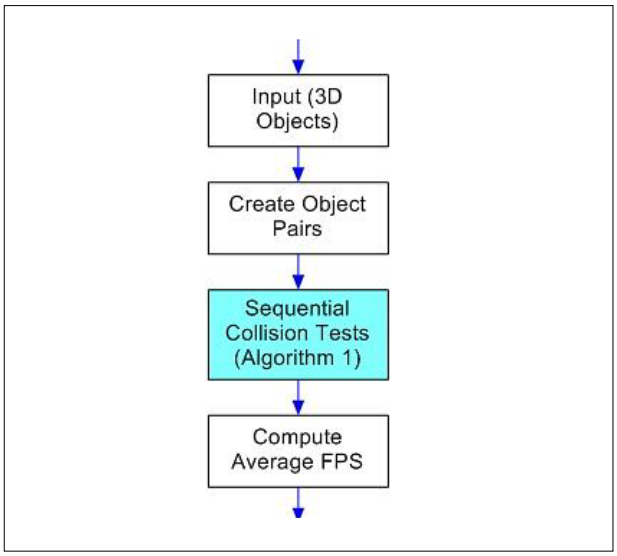

Fig. 2. Sequential approach research framework.

Bounding spheres were used throughout this research as the objects' bounding volume. 3D data files were used as input for the 3D objects used in the research framework shown above. Objects that have the potential to collide with each other are paired up together, creating a list of object pairs. Each pair is then tested for collisions. During collision test process, the radius of bounding sphere pair is summed up together to check for collision. An intersection (a collision) between a pair of bounding sphere only occurs if the distance between the spheres' centre is less than the sum of the radius of both spheres (refer to Figure 3 and Algorithm 1). 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.

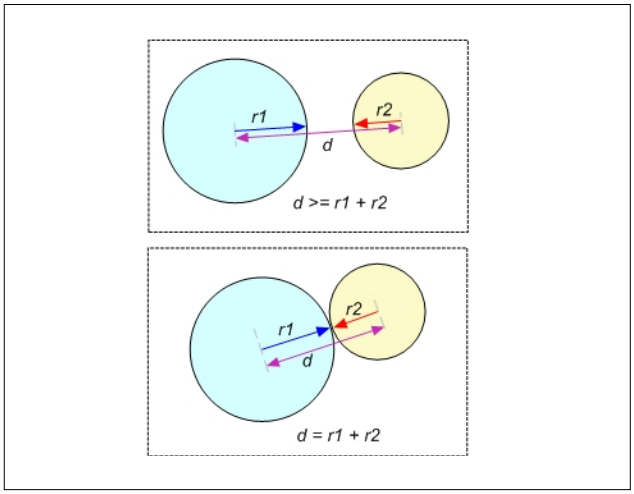

Fig. 3. Collision detection approaches.

Algorithm 1 shows the broad phase collision detection algorithm for sequential approach. The radius of bounding sphere 1 and bounding sphere 2 is represented by r_1 and r_2 respectively. The distance between the spheres' centre is denoted by d.

Algorithm 1: Collision detection for sequential approach

1. Pair objects that have the potential to collide
2. Function to test paired objects for collision
   - 2.1. Sum r_1 and r_2
   - 2.2. Compute d
   - 2.3. If $d \leq (r_1 + r_2)$ then return true
     Else return false
3. Compute FPS

Figure 4 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 main 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.

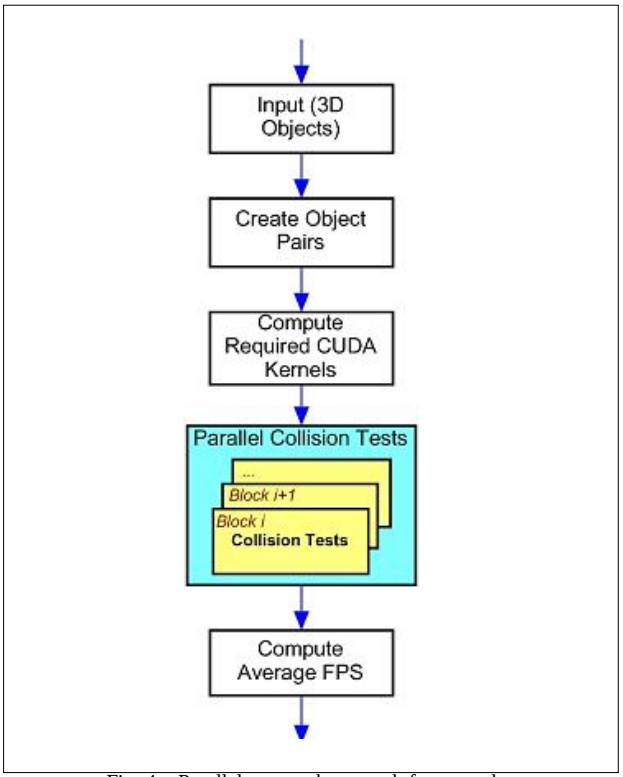

Fig. 4. Parallel approach research framework.

Figure 5 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 during the kernel launch.

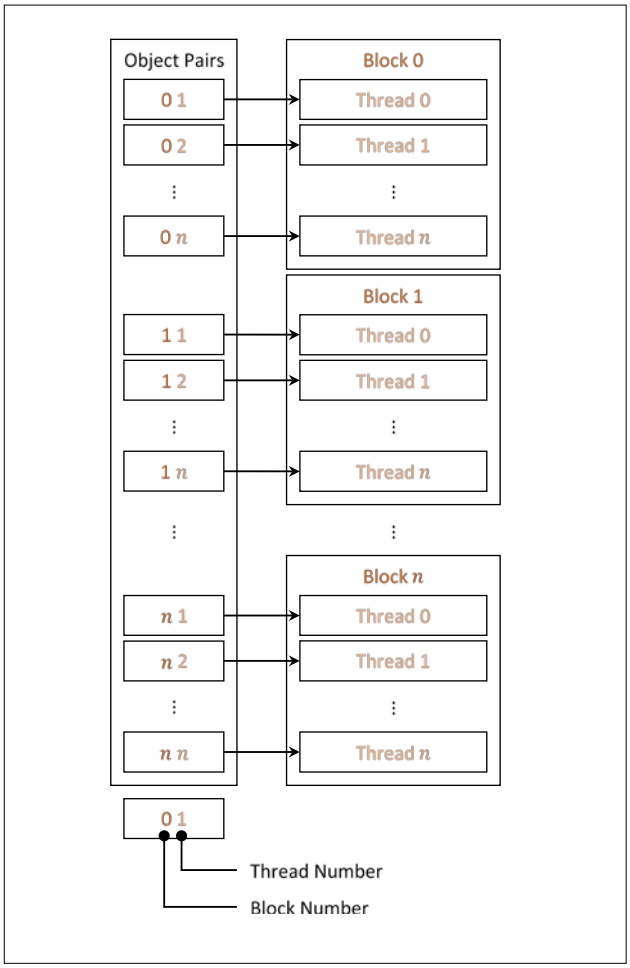

Fig. 5. Workload distribution.

The parallel broad-phase collision detection algorithm that was used is shown as Algorithm 2. Note that r_1 and r_2 are used to represent the radius of bounding sphere 1 and bounding sphere 2. Notation d on the other hand, represents the distance between the spheres' centre. Also, notice that a CUDA kernel is used in step 3 to distribute the workload. A kernel is a function that executes for a certain number of times, in parallel, depending on the number of blocks and threads specified during the kernel call.

Algorithm 2: Collision detection for parallel approach

1. Pair objects that have the potential to collide
2. Compute required number of CUDA blocks and threads
3. CUDA kernel to test paired objects for collision
   - 3.1. Sum r_1 and r_2
   - 3.2. Compute d
   - 3.3. If $d \leq (r_1 + r_2)$ then return true
     Else return false
4. Compute FPS

Performance comparisons between the sequential and parallel approach were made based on the frame rates from both computation. In computer graphics, frame rate refers to the speed at which the image is refreshed. Usually frame rates were measured in seconds (frames per second, FPS). 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.

The simulation used for experiments was written using C++ programming language and is developed using Microsoft Visual Studio 2008 integrated development environment (IDE). The Open Graphics Library (OpenGL) application programming interface (API) was used to render the scene (see Figure 6). NVIDIA CUDA Toolkit 5.5 was 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 consumer processor which is one of NVIDIA's CUDA capable GPUs.

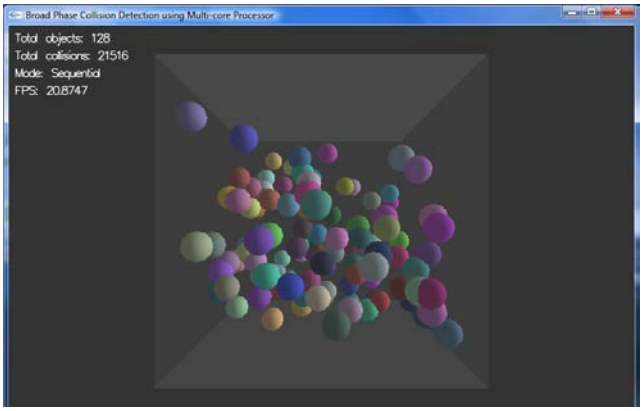

Fig. 6. The scene.

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. 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. Average readings will be calculated based on recorded data. This process is performed in order to increase the precision of the FPS value. Higher FPS reading is preferred since it supports real-time application. Figure 7 shows the average FPS captured for both sequential and parallel broad-phase collision detection.

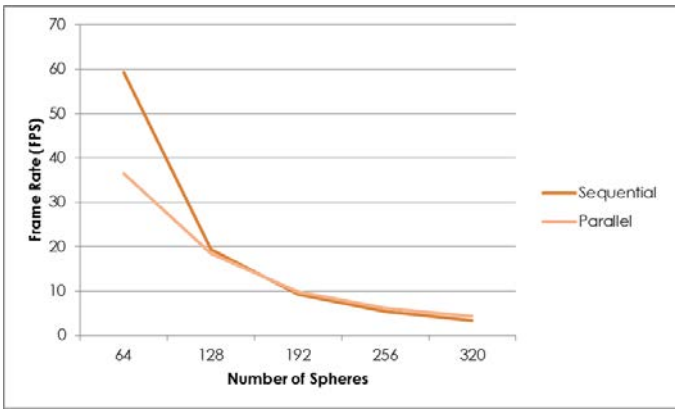

Fig. 7. Frame rates (FPS values) for sequential and parallel approach.

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 [28].

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	0.62
128	19.43	18.39	0.95
192	9.35	9.73	1.04
256	5.36	6.20	1.16
320	3.42	4.34	1.27

* average FPS and speed-up values were rounded up to second decimal

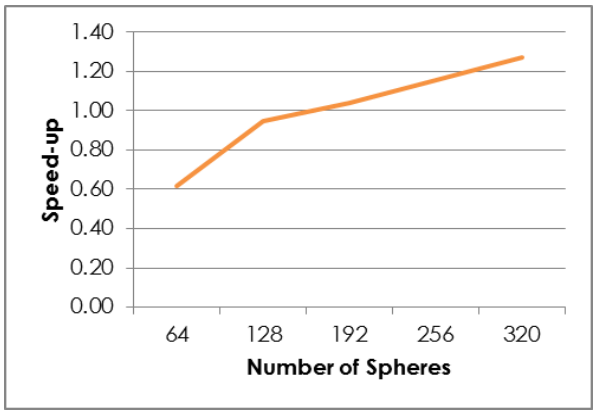

Fig. 8. Parallel approach speed-up.

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 8 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.

VI. CONCLUSION

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

We would like to express our special thanks to Universiti Teknologi Malaysia (UTM) and Ministry of Higher Education (MOHE) for related support and arrangements. This research was partly supported by RUG research grant 16H01.

REFERENCES

[1] Suaib, N.M., Bade, A., and Mohamad, D. 2013. 'Hybrid Collision Culling by Bounding Volumes Manipulation in Massive Rigid Body Simulation', TELKOMNIKA, 11, (6), pp. 8

[2] Bade, A., Suaib, N.M., and Daman, D. 2007. 'Collision detection in virtual environment', in Daman, D., Sunar, M.S., and Zamri, M.N. (Eds.): 'Advances in Computer Graphics & Virtual Environment' (Penerbit UTM Press), pp. 103 - 122

[3] Suaib, N.M., Bade, A., and Daman, D. 2008. 'Collision Detection: A Survey Of Techniques And Applications', in Daman, D., Bade, A., and Suaib, N.M. (Eds.): 'Collision Detection for Real-Time Graphics: Series of Techniques' Penerbit UTM Press, pp. 1-16

[4] Grand, S.L. 2008. 'Broad-Phase Collision Detection with CUDA', in Nguyen, H. (Ed.): 'GPU Gems 3'. Addison-Wesley.

[5] Avril, Q., Gouranton, V., and Arnaldi, B. 2010. 'A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures'. Proc. Virtual Reality International Conference (VRIC 2010), Laval, 7 – 9 April 2010, pp. Pages

[6] Weller, R. 2013. 'New Geometric Data Structures for Collision Detection and Haptics'. Springer International Publishing

[7] Kockara, S., Halic, T., Iqbal, K., Bayrak, C., and Rowe, R.: 'Collision detection: A survey', in Editor (Ed.)^(Eds.): 'Book Collision detection: A survey' (2007, edn.), pp. 4046-4051

[8] Hubbard, P.M. 1995. 'Collision detection for interactive graphics applications'. IEEE Transactions on Visualization and Computer Graphics, 1 (3), 218-230.

[9] Suaib, N.M., Bade, A., and Mohamad, D. 2015. 'Sphere-encapsulated Oriented Discrete Orientation Polytopes (S-Dop) Collision Culling for Multi-, Rigid Body Simulation', Jurnal Teknologi (Sciences & Engineering), 75, (2), pp. 7

[10] Lehericey, F., Gouranton, V., and Arnaldi, B. 2013. 'Ray-Traced Collision Detection : Interpenetration Control and Multi-GPU Performance'. Proc. 5th Joint Virtual Reality Conference of EuroVR - EGVE, Paris. pp. Pages

[11] Weller, R., Mainzer, D., Srinivas, A., Teschner, M., and Zachmann, G. 2014. 'Massively Parallel Batch Neural Gas for Bounding Volume Hierarchy Construction'. Proc. 11th Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS, Germany, 24 - 25 September 2014 2014 pp. Pages

[12] Stallings, W. 2010. 'Computer Organization and Architecture: Designing for Performance'. Pearson, 8/E ed.

[13] Padua, D. 2011. 'Encyclopedia of Parallel Computing' Springer, 4/E ed.

[14] Tang, M., Manocha, D., Lin, J., and Tong, R. 2011. 'Collision-streams: fast GPU-based collision detection for deformable models'. Proc. Symposium on Interactive 3D Graphics and Games, San Francisco, California.

[15] Pase, D.M., and Eckl, M.A. 2006. 'A Comparison of Single-Core and Dual-Core Opteron Processor Performance for HPC'. Proc. The 7th LCI Conference on High Performance Clustered Computing, Oklahoma, May 1-4, 2006

[16] Vassilev, T.I. 2012. 'Collision Detection for Cloth Simulation using Ray-tracing on the GPU', International Journal on Information Technologies & Security, 4, (4), pp. 10

[17] Avril, Q., Gouranton, V., and Arnaldi, B. 2011. 'Dynamic adaptation of broad phase collision detection algorithms', in Editor (Ed.)^(Eds.): 'Book Dynamic adaptation of broad phase collision detection algorithms'. pp. 41-47

[18] Pabst, S., Koch, A., and Straßer, W. 2010. 'Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces', Computer Graphics Forum, 29, (5), pp. 1605-1612

[19] Lauterbach, C., Mo, Q., and Manocha, D. 2010 'gProximity: Hierarchical GPU-based Operations for Collision and Distance Queries', Computer Graphics Forum, 29, (2), pp. 419-428

[20] Wald, I., Woop, S., Benthin, C., Johnson, G.S., and Ernst, M. 2014. 'Embree: a kernel framework for efficient CPU ray tracing', ACM Trans. Graph., 33, (4), pp. 1-8

[21] Geleri, F., Tosun, O., and Topcuoglu, H. 2013. 'Parallelizing Broad Phase Collision Detection Algorithms for Sampling Based Path Planners' in Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, Feb. 27 2013-March 1 2013.

[22] Erbes, R., Mantel, A., Schömer, E., and Wolpert, N. 2013. 'Parallel Collision Queries on the GPU', in Keller, R., Kramer, D., and Weiss, J.-P. (Eds.): 'Facing the Multicore-Challenge III'. Springer Berlin Heidelberg, pp. 84-95

[23] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. 2009. 'Fast BVH Construction on GPUs', Computer Graphics Forum, 28, (2), pp. 375-384

[24] Liu, F., Harada, T., Lee, Y., and Kim, Y.J. 2010. 'Real-time collision culling of a million bodies on graphics processing units', ACM Trans. Graph., 29, (6), pp. 1-8

[25] Lo, S.-H., Lee, C.-R., Chung, I.-H., and Chung, Y.-C. 2013. 'Optimizing Pairwise Box Intersection Checking on GPUs for Large-Scale Simulations', ACM Trans. Model. Comput. Simul., 23, (3), pp. 1-22

[26] Wald, I. 2012. 'Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture', IEEE Transactions on Visualization and Computer Graphics, 18, (1), pp. 47-57

[27] Zhang, X., and Kim, Y.J.: 'Scalable Collision Detection Using p-Partition Fronts on Many-Core Processors', Visualization and Computer Graphics, IEEE Transactions on, 2014, 20, (3), pp. 447-456

[28] Cheng, J., Grossman, M., and McKercher, T. 2014. 'Professional CUDA C Programming'. John Wiley & Sons, Inc.