Analysis of Axis Aligned Bounding Box in Distributed Virtual Environment

Elfizar
Department of Information System
University of Riau
Pekanbaru 28293, Indonesia

Sukamto
Department of Information System
University of Riau
Pekanbaru 28293, Indonesia

ABSTRACT
Axis Aligned Bounding Box (AABB) is the simple method for object collision detection, but it has limitation in detection process. In decades, some better methods have been generated such as Oriented Bounding Box (OBB) and HPCCD. Unfortunately, these methods are not used in DVE. This paper aims to analyze why most DVEs still use AABB in detecting objects collision in the environment. This research begins with developing the suitable DVE. The DVE should make many users collaborate with each other, and it has physics activities such as gravity pole, movement, etc. Each user is able to create objects and they should be visible to other users. To detect the object collision, AABB is implemented in the DVE. Further, to analyze the collision detection process and the performance of DVE, there are two parameters used, i.e. runtime and frame rate of simulation application. The experiment results show that adding the computation workload into AABB on DVE increases the runtime significantly compared with regular application. The lack of performance is also shown by the application frame rates in which strictly decrease so that the DVE performance degrades.

General Terms
Distributed Virtual Environment, Distributed Simulation

Keywords
AABB, Collision detection, Distributed Virtual Environment

1. INTRODUCTION
Virtual Environment (VE) is environment that imitates the real environment and makes user feel as residing in the real world. Some activities and situation in this environment should meet the real environment requirements. As the VE involves some users locating in different places geometrically, it is known as Distributed Virtual Environment (DVE). Currently, DVE has been used widely in many applications such as training, education, games, social communities, etc. Even DVE has been used as a powerful tool for autism children training [1].

An aspect in a real world influencing VE is a constraint that two objects are not able to occupy the same point in a space at the same time. Generally, object representation in VE does not allow penetration between objects. Therefore, to develop a simulation environment that represent a real world this constraint should be satisfied. One of important tasks is to detect collision among objects. Collision detection is a mechanism that is able to detect when and where the objects will collide [2].

Collision detection can be classified into two categories, i.e. discrete and continue. Discrete collision detection is a method that just detects the collision at a certain time, for instance at time *t*. Its consequence is that this method misses many collision detections between two consecutive configurations. It is called tunneling problem. Discrete collision detection does not require many computations so that the process is faster.

Continue collision detection can address the tunneling problem because it uses interpolation algorithm to examine the collision in a continue movement. It yields accurate solution for collision detection. Unfortunately, this method is slower than discrete method [3]. One of approaches used in continue collision detection is bounding volume that can be done by using box, sphere, etc. Axis Aligned Bounding Box (AABB) [4] is a method included in this category.

Because of its reliability in detection process, continue collision detection methods have been used by many researches to invent the new faster method. [5] have used linear interpolation between model vertices and computed the first time of collision occurred based on hierarchy selection as well as done basic testing between triangle pairs [6].

Another approach that has been used to accelerate the continue collision detection is using parallel computation [7]. It is inspired by the capability improvement of current processor/CPU that uses multi cores. [8] have yielded parallel collision detection algorithm which run parallel on computers with eight cores CPU and on 16 cores. Each computer gives collision detection speed of 7x and 13x faster, respectively.

Besides using multi processors, acceleration of collision detection speed has been done by using some Graphics Processing Unit (GPU). In contrast to the CPU, GPU processors are very suitable for parallel computation. It is caused by the number of GPU cores is greater than CPU cores, and GPU has more bandwidth than CPU [9-12].

Because bandwidth of both CPU and GPU is restricted, it is required to integrate CPU and GPU in order to compute the collision detection among objects. [13] have used four cores CPU and two GPU. The results show that acceleration can be achieved from 50% to 80% compared with using just CPU for the same test model.

Unfortunately, the improvement in these researches is not followed by the implementation of the resulted methods into DVE. DVE is still using simple method to do the collision detection among objects such as measuring distance between objects [14], and AABB. It affects the DVEs generated by developers in which they give inaccurate collision detection in their environment.

This paper aims to analyze the implementation of AABB in DVE and to find the answer that why many DVEs still use simple method especially AABB in order to do the collision detection in the environment. Even there are many other reliable methods available.

The contribution of this paper is a thorough analysis of AABB implementation in DVE. The result can be used by other researches to find out a method that is more accurate than AABB and suitable to be used in DVE.

Following this section, Section 2 describes AABB as a method used in this research. The proposed model is presented in Section 3. Further, the results and analysis are presented in Section 4. Finally, Section 5 delivers the conclusion.

2. AABB METHOD

Each object in VE is covered by a box as illustrated in 2D by Figure 1. In three-dimensional (3D), this box is drawn aligning with each axis in coordinate system (X,Y,Z). Hence, it is called Axis Aligned Bounding Box (AABB) [4].

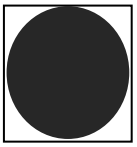

Fig 1: A box covering a ball object in 2D

When each side of box is projected onto each axis, then two objects can be determined whether they collide with each other. Figure 2 shows two objects that do not collide in 2D because projection intervals of both objects in X-axis do not overlap (K_1-L_1 and K_2-L_2 intervals do not overlap).

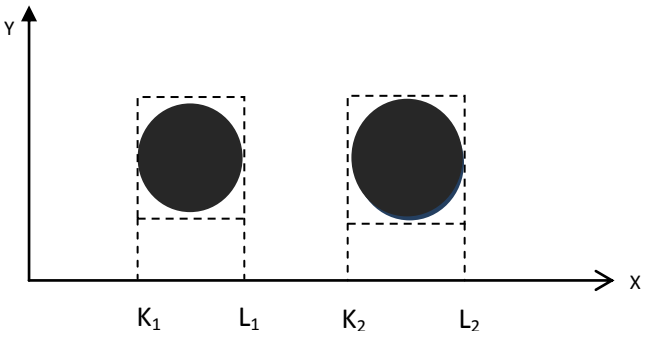

Fig 2: Non-overlapping projection intervals

Further, Figure 3 shows two colliding objects because projection intervals overlap on both axis (X, and Y). Therefore, two objects are called colliding in VE when projection intervals overlap on each axis (X,Y,Z).

3. THE PROPOSED MODEL

3.1 DVE Analysis and Design

Using the DVE, user is able to create one or more objects. User can create dynamic objects that can move in the environment. Objects created by a user can be visible by other users. In other words, each user has the same display in the same interface, time and point of view for the number of objects, and what happening to the environment.

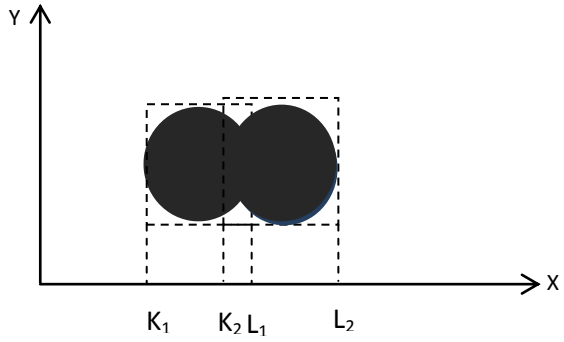

Fig 3: Overlapping projection intervals

Because there are many objects residing in VE, the possibility of object collision is very high. Hence, VE should have a method to handle the collision. The AABB method is used for this task. Another important thing is VE should meet the real condition. For example, VE should have gravity pole, and when an object collides with other objects then the result of this collision will change the position and physics of objects.

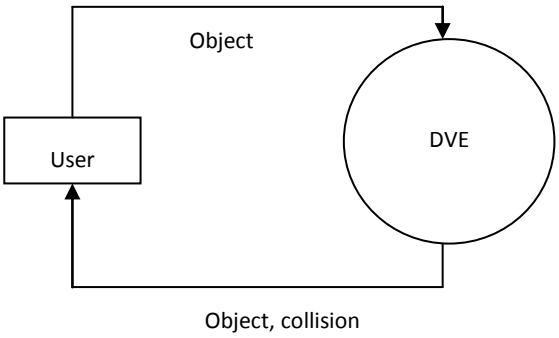

Fig 4: Interaction between user and application

Based on all requirements above, interaction between user and DVE application can be illustrated by Figure 4. Furthermore, Figure 5 shows algorithm of the application. As user creates an object in VE, the application does the collision detection between the object and other object using the AABB method.

3.2 Implementation of AABB in DVE

Experiments run on computers with dual core processors, and memory of 1 GB RAM. DVE is developed by using C, OpenGL, and Open Dynamics Engine (ODE) as simulation engine. The implementation is run by using Linux operating system.

This research uses two scenarios. The first scenario is running the AABB as collision detection method in DVE, and the second scenario is adding the workload to the AABB and then run it in DVE. The second scenario means that the DVE uses another method, which has more workload than AABB.

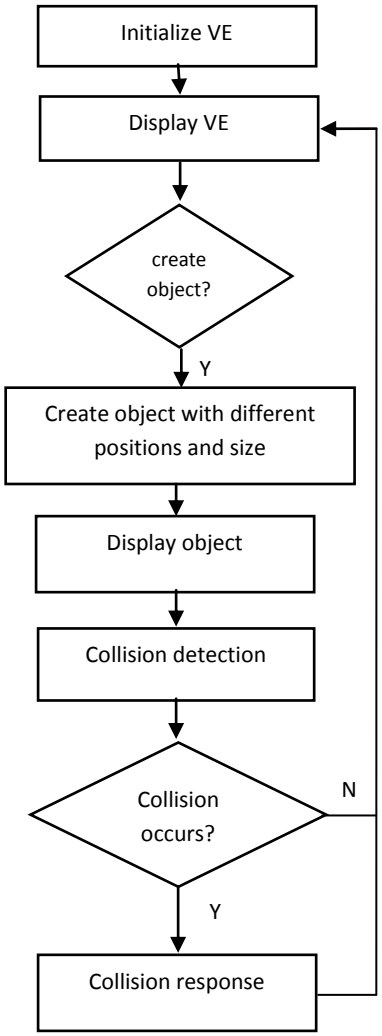

Fig 5: Application algorithm

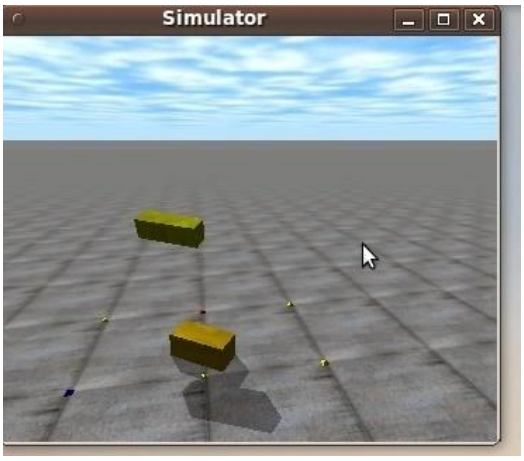

Fig 6: Two 3D objects created by user

4. RESULTS AND DISCUSSION

With the application, Figure 6 illustrates two 3D objects created by users in VE. The objects are boxes with varying positions and sizes. These boxes are dynamic objects falling from a certain high to the ground. Figure 6 also shows that VE has physics activities as real world. For instance, it has gravity pole and collision detection. User can change point of view by using translation or pan-tilt options.

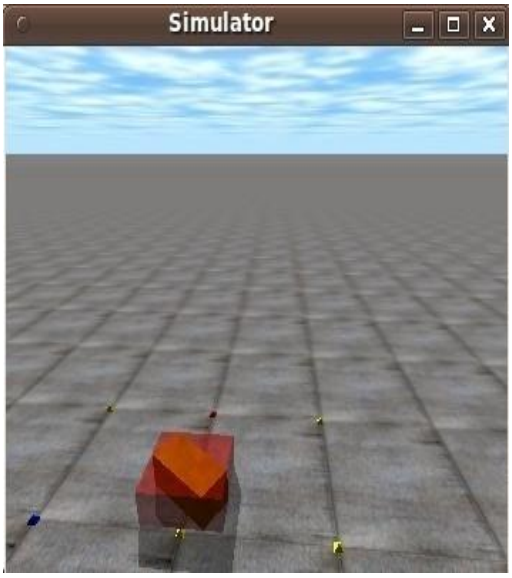

Fig 7: Bounding box of an object

Using AABB on object creates a box covering that object as illustrated in Figure 7. From the figure, the box is made align with three coordinates (X,Y,Z). The collision between two objects is determined by using their boxes. Furthermore, Figure 8 shows two pair of objects that collide and not collide with each other. Two objects are called colliding when their bounding boxes projections overlap in three coordinates: X-axis, Y-axis, and Z-axis. As seen in the figure, pair of objects that is far from the user position does not collide because the projections of objects on one axis do not overlap. It is different from another pair of objects that collides because all projections onto three coordinates overlap.

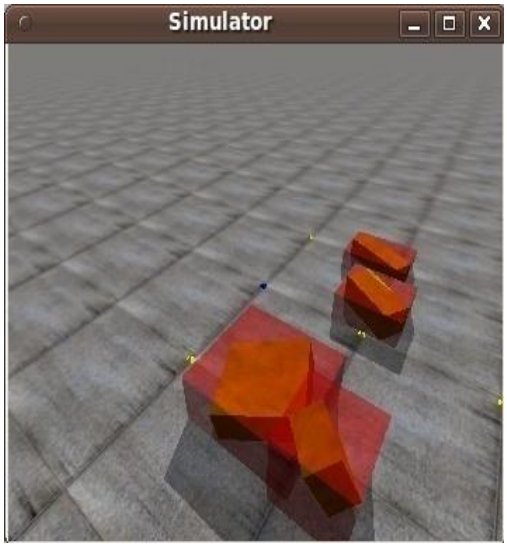

Fig 8: Two pairs of objects with and no collision

To analyze DVE, there are two parameters used in the research, i.e. runtime and frame rate for both scenarios. The workload addition for second scenario is counting sum of integer numbers from 1 to 1,000. Adding the complexity of O(*n*) to AABB should not affect the AABB itself because the overall AABB complexity does not change. Unfortunately, the application runtime of second scenario increases significantly compared with first scenario for ten times experiment as shown in Table 1.

Table 1. Runtime measurement

Experiment	Scenario I (s)	Scenario II (s)
1	0.01	0.1
2	0.01	0.1
3	0.015	0.12
4	0.015	0.1
5	0.01	0.1
6	0.015	0.1
7	0.01	0.12
8	0.015	0.1
9	0.015	0.1
10	0.01	0.1
Average	0.0125	0.104

Further, Figure 9 illustrates the difference between two scenarios. There is addition almost 0.1 second to the second scenario, even the simple C program to execute the workload addition only needs 0.03 second. Thus, execution of addition workload in DVE needs much time compared with simple C program.

Table 2. Application frame rates

Experiment	Scenario I (fps)	Scenario II (fps)
1	240	160
2	240	160
3	245	160
4	240	155
5	240	150
6	240	160
7	240	160
8	230	165
9	245	165
10	240	165
Average	240	160

Table 2 shows the frame rates of application for both scenarios. The measurement is conducted for ten times experiment. Figure 10 also depicts the difference between both scenarios. In contrast to the runtime, the frame rates of the second scenario decrease significantly compared with the first scenario in order to execute the addition workload. There is the disparity of 80 frames per second (fps). The workload addition given to second scenario causes the performance of DVE degrades and automatically reduces user experience in using the DVE.

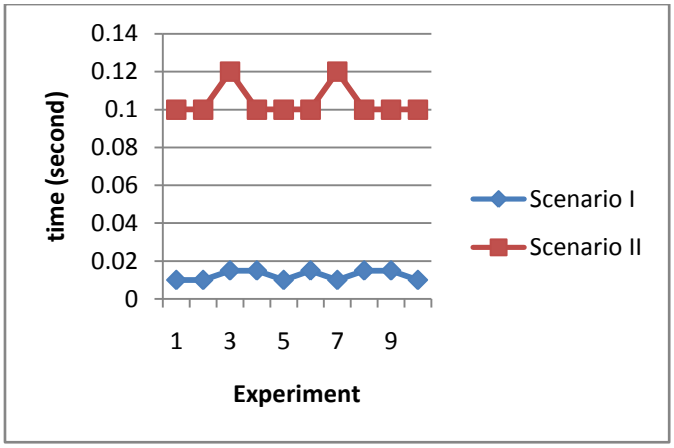

Fig 9: Application runtime for both scenarios

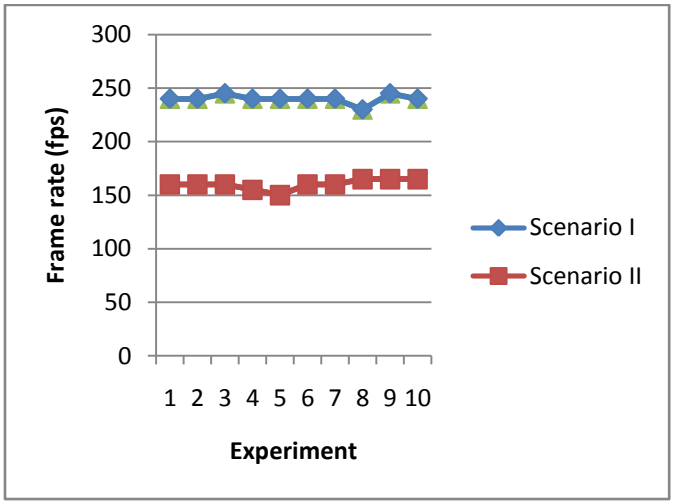

Fig 10: Application frame rates for both scenarios

5. CONCLUSION

This paper has been analyzed AABB method implemented in DVE. There are two parameters used in the research: application runtime and frame rates. To compare the performance of DVE, this research uses two scenarios i.e. DVE uses AABB method, and DVE uses AABB method with addition workload.

DVE application always updates its environment. It is busy to update their objects behaviors and appearances. This characteristic causes the computation time to execute the addition workload given to the AABB method increases significantly. Even the addition workload only requires a little time when it is executed by using the simple C program.

In contrast to the application runtime, adding workload to AABB method causes DVE frame rates decrease. The frame rates of the second scenario are lower than the first one. It makes the performance of DVE degrade and finally gives less user experiences.

As finding of the research, those two results give the answer of that why the modification methods resulted by the last researches are not used in the current DVE. Using modification method that has higher complexity than AABB increases the runtime of DVE significantly. Unfortunately, it also decreases the DVE frame rates. In fact, the case is that AABB method has many limitations.

In future work, investigating a better method used in DVE is needed. To address the increasing of runtime and decreasing of application frame rate, separating the collision detection engine from simulator will be considered in order to reduce the simulator workload.

6. ACKNOWLEDGMENTS

This research is funded by University of Riau in research scheme of Fundamental Research. Thanks to reviewers for the valuable feedbacks to this paper.

7. REFERENCES

[1] Parsons, S., Mitchell, P., and Leonard, A. 2005. Do adolescents with autistic spectrum disorders adhere to social conventions in virtual environments?. Autism, 9(1), 95-117.

[2] Bergen, G.V.D. 2004. Collision detection in interactive 3D environments. Morgan Kaufmann publishers, San Francisco.

[3] Sulaiman, H.A., and Bade, A. 2011. Continuous collision detection for virtual environments: A walkthrough of techniques. Electronic journal of computer science and information technology, 3(1), 1-7.

[4] Bergen, G.V.D. 1997. Efficient collision detection of complex deformable models using AABB trees. Journal of Graph Tools, 2(4), 1-13.

[5] Tang, M., Curtis, S., Yoon, S.E., and Manocha, D. 2009. ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. IEEE Transactions on Visualization and Computer Graphics, 15(4), 544-557.

[6] Curtis, S., Tamstorf, R., and Manocha, D. 2008. Fast collision detection for deformable models using representative-triangles. In Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, 61-69.

[7] Shellshear, E., Bitar, F., and Assarsson, U. 2013. PDQ: parallel Distance Queries for Deformable meshes. Graphical Models, 75(2), 69-78.

[8] Tang, M., Manocha, D., and Tong, R. 2010. MCCD: Multi-core collision detection between deformable models using front-based decomposition. Journal of Graphical Models, 72, 7-23.

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

[10] Choi, K.W., Negrut, D., and Thelen, D.G. 2013. GPU-based algorithm for fast computation of cartilage contact pattern during simulations of movement. In Proceedings of ASME 2013 Summer Bioengineering conference.

[11] Tang, M., Tong, R., Narain, R., Meng, C., and Manocha, D. 2013. A GPU-based streaming algorithm for high-resolution cloth simulation. Computer Graphics Forum, 32(7), 21-30.

[12] Avril, Q., Gouranton, V., and Arnaldi, B. 2014. Collision detection: broad phase adaptation from multi-core to multi-GPU architecture. Journal of Virtual Reality and Broadcasting, 1-13.

[13] Kim, D., Heo, J.P., and Yoon, S. E. 2009. HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. Computer Graphics, 28(7), 1791-1800.

[14] Nassiri, N., Powell, N., and Moore D. 2010. Human interactions and personal space in collaborative virtual environments. Virtual Reality, 14, 229-240.