ISSN 2723-6471

Implementation of Enhanced Axis Aligned Bounding Box for Object Collision Detection in Distributed Virtual Environment

Elfizar[1,*], Tutut Herawan[2], Sukamto[3]

[1,3]*Department of Information System, University of Riau, Indonesia*

[2]*Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia*

(Received: May 26, 2024; Revised: June 21, 2024; Accepted: July 13, 2024; Available online: August 30, 2024)

Abstract

Axis Aligned Bounding Box (AABB) is a popular method for object collision detection in distributed virtual environment (DVE). Actually, besides AABB several researchers have found better methods but those methods are not used in DVE. The simplicity of AABB is one of reasons. Another side, researchers have found several methods to make a scalable DVE to accommodate huge users. Object-based simulators architecture is one of them. It uses a simulator for an object in DVE. This paper aims to determine the comprehensive effects of implementing AABB in DVE and to enhance AABB implementation in DVE using object-based simulators architecture in order to be able to use other object collision detection methods in DVE. This research had four stages. First stage, we developed DVE application. The second stage was running AABB in DVE. In this stage, we also determined characteristics of AABB implemented in DVE. Third, we implemented AABB in object-based simulators architecture. This architecture used more simulators to manage objects. Finally, we analyzed and compared the results to the common existing DVE. There were three parameters measured in this research: runtime, frame rate and CPU usage of simulation application. The experiment results showed that computation complexity was the most important thing to be considered in DVE because DVE always updates its environment. Adding a few workloads into AABB degraded strictly performance of DVE. The results also showed that the enhanced AABB implementation in DVE using the object-based simulators architecture could rich the user experiences. The average runtime, frame rates and CPU usages using this architecture are 0.001 second, 239 fps and 20.94%, respectively. These achievements are better than common DVE. This novel approach could be used to implement other better collision detection algorithms in DVE such as oriented bounding box (OBB).

Keywords: AABB, Collision Detection, Distributed Virtual Environment, Object-Based Simulators

1. Introduction

Virtual environment (VE) is an environment that imitates the real one and makes user feeling as residing in the real world. Therefore, all activities and situation in the VE should meet the real environment requirements. If 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 simulation, social communities, education, laboratory, games, etc. Even DVE has been used as a powerful tool for autism children training [1], [2], [3]. An aspect in a real world influencing the VE is 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 virtual environment that represent a real world this constraint has to be fulfilled. One of important tasks in VE is to detect collision among objects. Thus, we can formulate that collision detection is a mechanism which is able to detect when and where the objects will collide [4].

Generally, collision detection can be divided into two categories: discrete and continuous. First, discrete collision detection is a method that just detects the collision at a certain time, for instance at timet. Its consequence is this method misses many collision detections between two consecutive configurations. It is called tunneling problem. This method does not require many computations. Therefore, this method gives faster processing. Different from the first method, the continuous collision detection can address the tunneling problem because it uses interpolation algorithm to examine the collision in a continuous movement. The continuous collision detection yields more accurate solution for collision detection. Unfortunately, this continuous collision detection method is slower than discrete method [5]. Generally,

***Corresponding author: Elfizar (elfizar@lecturer.unri.ac.id)**
DOI: https://doi.org/10.47738/jads.v5i3.226
This is an open access article under the CC-BY license (https://creativecommons.org/licenses/by/4.0/).
© Authors retain all copyrights

continuous collision detection uses bounding volume that can be done by using box, sphere, etc. One of the methods including in this category is Axis Aligned Bounding Box (AABB) [6], [7], [8].

Since its reliability in detection process, continuous collision detection methods have been used in many researches to invent the new faster method. Tang et al. [9] had used linear interpolation between model vertices and computed the first time of collision occurred based on hierarchy selection. Curtis et al. [10] had also done basic testing between triangle pairs. Shellshear et al. [11] used parallel computation to accelerate the continuous collision detection. Capability improvement of current processor that uses multi cores inspired that research [12]. Tang et al. [13] yielded parallel collision detection algorithm. That algorithm was run parallel on eight and 16 cores CPU. Each computer gave collision detection speed of 7x and 13x faster than common computer, respectively. Another side, several researches accelerated collision detection speed using Graphics Processing Unit (GPU) [14], [15], [16], [17], [18]. Different from CPU, using GPU processors are very suitable for parallel computation since the number of GPU cores is greater than CPU cores, and GPU has more bandwidth than CPU [19], [20], [21], [22].

Restricted bandwidth of both CPU and GPU made researchers integrate CPU and GPU to compute the collision detection among objects [23]. Kim et al. [24] used four cores CPU and two GPU. The results show that it can accelerate the computation speed from 50% to 80%. Unfortunately, the improvement in these researches is not followed by the implementation of those methods in DVE. The DVE is still using simple method to do the collision detection among objects i.e. AABB. Therefore, the latest virtual environment application generated by developers always use the same method. Qi et al. [25] used AABB to handle collision detection for underground pipelines. Elfizar et al. [26] had found a method to make the DVE able to access by huge users. This architecture is able to make a large scale distributed virtual environment [27]. It is called object-based simulators architecture. It can yield the significant decrease of runtime and memory use. This architecture has more simulators to handle objects residing in DVE [28], [29].

In this paper, we determine the comprehensive effects of implementing AABB in DVE. The characteristics of AABB is find out in order to answer why many DVE developers still use AABB to handle the collision. These characteristics are used to determine the limitation of common existing DVE. To address the limitation of DVE we propose a novel approach to enhance the AABB implementation in DVE. The novelty of the proposed approach is that the AABB is done by many simulators in DVE to handle the collision. There are three contributions of this paper. First, we study characteristics of the common existing DVE. With these characteristics, we are able to find the limitation of DVE in handling the collision. Second, we deliver enhanced AABB method implementation in a DVE. The result can be used by developers to build a large scale DVE and remain able to detect collision detection faster. Finally, we yield the novel approach that can be used by researchers to implement other better collision detection algorithm in DVE. The rest of the paper is organized as follows: Section 2 describes the literatures review: AABB method and the object-based simulators architecture. The experiment scenario, several stages and hardware used in this research are described in Section 3: the research method section. Further, Section 4 presents and describes the experimental results. Finally, the paper delivers the conclusion at Section 5 and the future work at Section 6.

2. Literature Review

2.1. Axis Aligned Bounding Box Method

AABB is one of the continuous collision detection methods. Each object 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) [6]. Assume that we have two balls. If each side of boxes is projected onto each axis, then these two objects can be determined whether they collide or not. Figure 2 shows us two objects that do not collide because projection intervals between both objects in X-axis do not overlap (K1-L1 and K2-L2 interval do not overlap). Mathematically, this condition meets the expression L1 < K2 AND K1 < L2. Further, figure 3 shows two colliding objects because projection intervals overlap on both axis (X and Y). Therefore, collision between two objects occurs in VE if the projection intervals overlap on each axis (X,Y,Z).

ISSN 2723-6471

Figure 1. A box covering an object (in 2D)

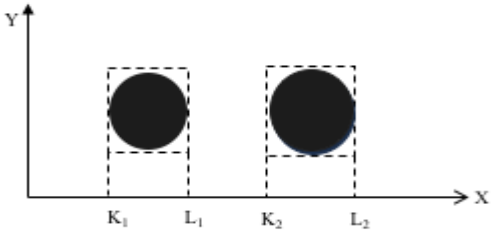

Figure 2. Nonoverlap projection intervals

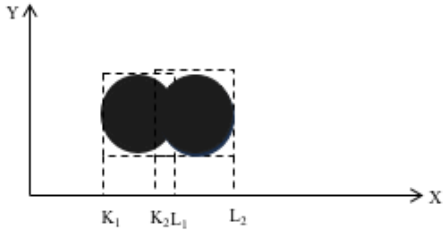

Figure 3. Overlap projection intervals

Based on figure 3, the collision will occur when this expression below is satisfied for the X-axis: L1 > K2 AND K1 < L2. This expression also occurs in other two axis. An important thing to note about that expression is that it is made up of a series of AND which means that evaluation will stop as soon as one of the conditions fails.

2.2. Object Based Simulators Architecture

The object-based simulators architecture uses the concept that simulator is no longer a single process that controls many objects. Each object is acted as a simulator. It is inspired by the concept that we look at DVE as a collection of objects such as trees, animals, houses, boxes, avatars etc. In DVE, several objects are static and the others are dynamic. The dynamic objects are able to interact with other objects. Each object is an independent process in the DVE. It can determine its appearances and behaviors in the environment. For instance, avatar is an object and it is a process that has specific appearances and behaviors. This process maintains how the avatar should walk, handshake, sit, and other activities. The avatar may reside in any region or scene in DVE using its properties. The region just calls this process. This viewpoint is called object-based viewpoint that is very different from the common DVE architecture.

Actually, object-based viewpoint is motivated by the real-world environment. We can easily see person, buildings, animals, trees, etc. They are objects composing the world and each has distinct appearances and behaviors. Those properties are controlled by the object itself [28]. Each object is independent process and separated from other objects. The universe terminology is used to show that objects reside in a continuous space. It differs from the existing DVE which uses a partitioned space. A process that handles the appearances and behaviors of object is called object simulator [29]. Several object simulators are shown by figure 4 (right side).

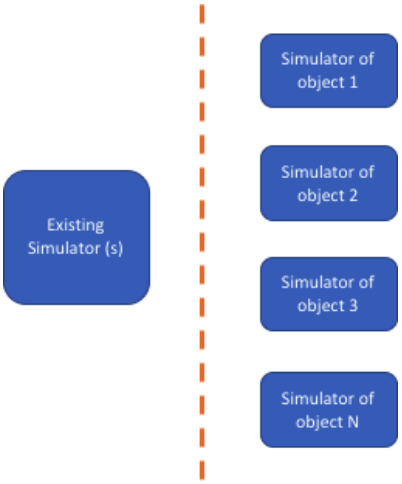

Figure 4. Existing and the object-based simulators

Figure 4 also shows the difference between simulator used by the existing and object-based viewpoints. Based on figure 4, representation of the current viewpoint is the left side of the figure. This simulator simulates many objects. Different from the left side, the right side of this figure represents several simulators based on the number of objects. It aligns to the object-based viewpoint. This model makes object independent in managing itself [29]. The architecture of the object-based simulators (OBS) is shown by figure 5.

 ISSN 2723-6471

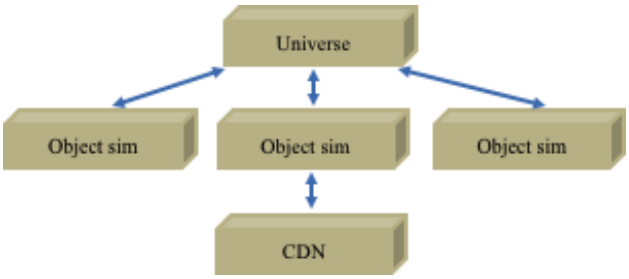

Figure 5. Object based simulators architecture

The universe component is responsible to determine what objects reside in the environment, their location, and their physical properties. It stores object properties in a DVE such as object identifier, position, and physical properties. Object sim is a simulator that has two components in managing an object: scripts and physics engine. Scripts engine is used to run the object scripts. It determines the appearances and behaviors of object. Physic engine runs the physics simulation of the object. This engine ensures that the object in the DVE enforces the physics laws such as gravity pole, collision, etc. Content Delivery Network (CDN) is the same as CDN component in the Sirikata architecture [30]. It stores permanent data and delivers data to the other components. Meshes that are used to display objects are example of the data stored on CDN. Viewers are able to download them to view the VE. The CDN is able to be as simple as a web server. It really needs only to serve requests for files.

There are several advantages of object-based simulator architecture. First, object simulator is an independent process so that it can reside in distributed computers. Therefore, scalability with the additional hardware is not an issue. Second, identification of object is able to be done using simulator ID where existing DVEs still use server ID. It supports the migration of simulator and does not affect the overall simulation process. Third, the object simulator workload is smaller than existing DVE simulators. Therefore, if the number of objects increase in DVE then it does not affect the simulator itself. This advantage eliminates the algorithm of workload distribution among servers as occurred in existing DVE.

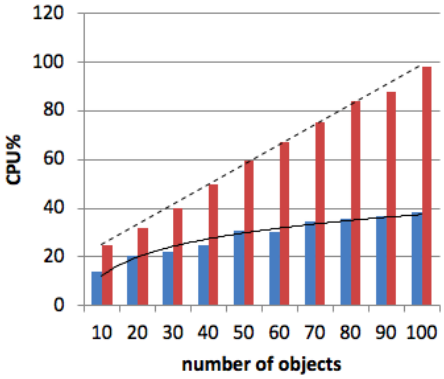

Figure 6. CPU usages of Sirikata and OBS

Elfizar et al. [26] has used Sirikata (existing DVE simulator) to be compared to the object-based simulator. The results show that the object-based simulator scalability and performance are better than the Sirikata. Figure 6 shows the CPU usages of both Sirikata (red) and object-based simulator (blue) for varying number of objects. From figure 6, we note that the trendline of Sirikata CPU usages is linear (dash-line) so that its complexity is O(n). Therefore, the trend-line of the CPU usages of object-based simulator is logarithmic (solid line) so that the complexity of OBS object simulator is O(log n).

3. Research Method

The experiment used core i-7 processor and 8 GB RAM. Further, this experiment had two scenarios. Scenario 1 implemented AABB in the common existing DVE. The scenario 2 implemented AABB in the object-based simulators

ISSN 2723-6471

architecture. After running the scenario 1, we analyzed the implementation of AABB in DVE. What factors AABB still implemented in DVE until now were identified. Further, we run the scenario 2 where the experiment used AABB in object-based simulators architecture. In this scenario we used more simulators based on the number of objects in DVE. Finally, the results between two scenarios were compared.

There are three parameters used in the research i.e. runtime, frame rates and CPU usage. They are used to measure the performance of DVE. We use the simulation frame rate to know the effect of increasing of the number of objects and collisions on the frame rate of simulation. The frame rate of DVE is measured by Glxgears application. The runtime of DVE is measured using simple script that displaying the time required by DVE to handle the collision. Further, the CPU usage is taken from the Activity Monitor terminal available from the operating system.

The DVE developed in this experiment is a 3D environment. The basic elements in this environment are the sky and the ground. The appearance of these elements imitates the real world. The interaction between users and DVE is illustrated by figure 7. User may able to create moving objects. As user creates an object the other users are able to view that object. All users reside in a local area network. If two or more objects collide then the DVE displays that collision. It also displays the collision view and the next appearances of those as collision response. This DVE also has user moving options i.e. translation and pan-tilt option.

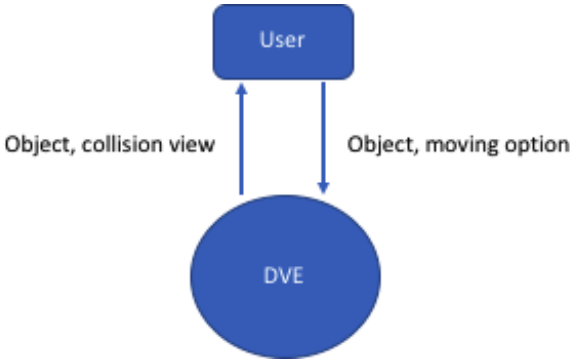

Figure 7. Interaction diagram

The algorithm of developing DVE is shown by figure 8. It begins with initializing the DVE. This stage relates to the preparing of the stuff required by DVE. After display the DVE, users are able to create object (s). The objects used in this research are dynamic boxes falling to the ground with varying sizes and appearances. When a user creates an object in DVE, it can be viewed by other users directly. The appearance of object is different depending on their viewpoint. User uses viewer application to view DVE and it is responsible to display the VE to user. Basically, viewer is an object in the DVE and user can use it to explore the world. But, in this experiment user is not represented by an avatar in the DVE. The viewer application is able to support multi-viewpoint. It means that user is able to have different viewpoint from other users to view the same environment. Further, the next step is collision detection. The falling boxes have high probability to collide with others. The collision is able to occur between object and ground, among objects or combination of them. DVE uses AABB method to detect the objects collision in the environment. If there is collision between two objects then the DVE handles the collision response for both objects. The response can be in term of different positions and appearances of the objects after the collision.

ISSN 2723-6471

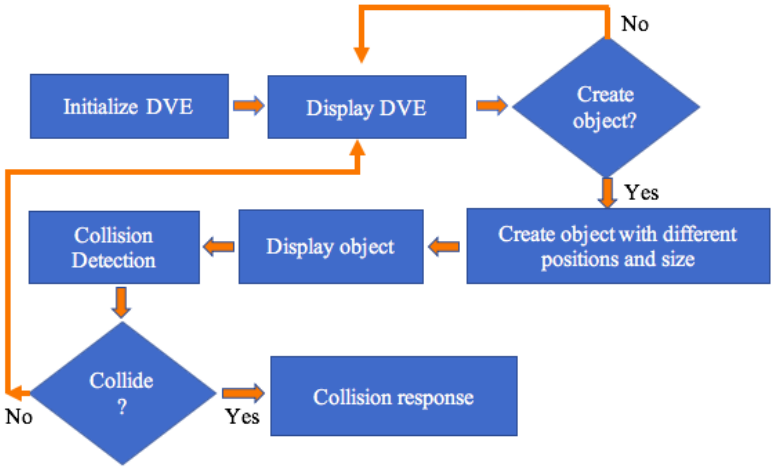

Figure 8. DVE algorithm

4. Results and Discussion

This section describes the experiment results based on two scenarios. Three subsections describe developing DVE, implementing AABB in DVE and implementing AABB in object-based simulators architecture, respectively.

4.1. Developing DVE

DVE is developed by using C, OpenGL and Open Dynamics Engine (ODE) as simulation engine. The application runs on Linux operating system. Figure 9 shows the initial view of DVE. Users can view the environment and create objects using this interface. Further, figure 10 shows two 3D objects created by users in DVE. The objects are boxes with many positions and sizes. These boxes are dynamic objects that falling from a certain high to the ground. Figure 10 also illustrates that DVE has physics activities similar to real world. This DVE has gravity pole and collision detection. Besides that, translation and pan-tilt options are also available to change point of view.

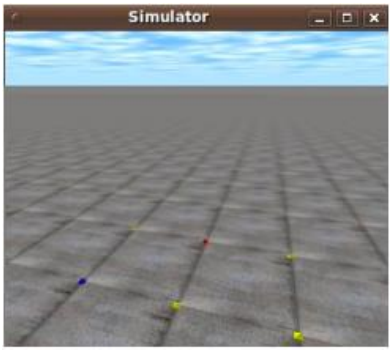

Figure 9. DVE initial view

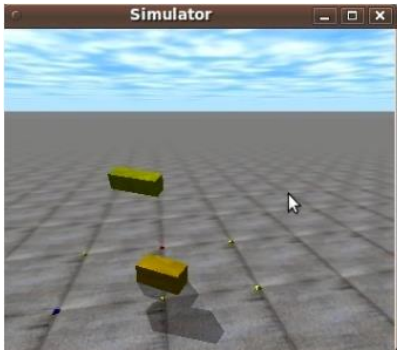

Figure 10. 3D objects created by users

4.2. Implementing AABB in DVE

Principally, implementing AABB on the object is creating a box covering that object as illustrated in figure 11. The box aligns with the coordinates (X,Y,Z). Using the box, the collision between two objects is able to be determined. The collision between two objects is shown by figure 12. Two objects collide when their bounding boxes overlap in X-axis, Y-axis, and Z-axis. In the figure, we can see that pair of objects far from the user position does not collide because there is no overlap of the projections on one axis.

 ISSN 2723-6471

Figure 11. Bounding box of object

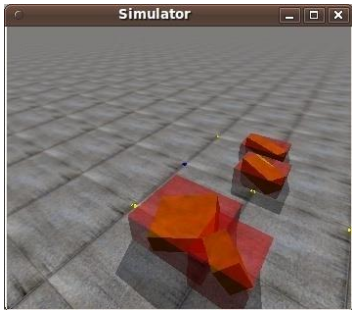

Figure 12. Collision between two objects

Table 1 shows the measurement of runtime, frame rates and CPU usages of the existing DVE. From Table 1 we can see that the runtime, frame rates and CPU usages are almost the same for all experiments. The average runtime, frame rates and CPU usages of DVE are about 0.007s, 234 fps and 50.28% respectively to manage and display the collision detection.

To get more information about using AABB in DVE, we add a workload of counting sum of integer numbers from 1 to 1,000 into AABB processing script. It means that adding the complexity of O(n) to AABB. Adding this workload aims to find out the reason of why DVE developers do not use other better collision detection methods in DVE. They always choose AABB method. The results of runtime, frame rates and CPU usages of additional workload in DVE are shown in table 2. From table 2, we can see that the average runtime, frame rates and CPU usages of DVE are about 0.045s, 153 fps and 52.33% respectively. It means that the performance of DVE is reduced. It is characterized by slow movement of the falling objects in DVE.

Comparing table 1 and table 2, we can see that the application runtime increases significantly for ten times after addition of the AABB workload. Logically, it should not affect the AABB itself since the overall AABB complexity does not change. It means that adding simple C script in AABB process increases almost 0.1 second runtime. Actually, that simple C script to execute it only needs 0.003 second. Thus, we need more time in DVE compared to execute the same script in simple C program.

It also occurs to CPU usages. Adding the scripts increases the CPU usage of DVE. Further, different from table 1the frame rates of DVE decrease for overall experiments. It means that the additional C script affects the performance of DVE. Thus, it decreases the user experience in using the DVE.

Table 1. Runtime, Frame Rates and CPU Usage of DVE (Scenario 1)

Experiment	Runtime (s)	Frame Rate (fps)	CPU Usage %
1	0.008	240	49.52
2	0.009	240	49.65
3	0.008	230	50.01
4	0.010	240	49.3
5	0.006	230	49.8
6	0.005	230	50.57
7	0.008	240	49.45
8	0.005	230	49.7
9	0.005	230	50.2
10	0.006	230	50.8
11	0.008	240	50.33
12	0.005	230	50.47
13	0.005	230	51.3
14	0.006	230	51.76
15	0.005	230	51.38

ISSN 2723-6471

4.3. Implementing AABB in Object Based Simulators Architecture

In object-based simulators (OBS) architecture, we do not use a simulator to handle all objects in DVE. As illustrated by figure 5, this architecture has many simulators to handle objects residing in DVE. In this experiment, the object-based simulators architecture is implemented in client-server network as shown by figure 13. Object simulators, universe and CDN run in server. Clients are viewers, they can view DVE and able to interact with other objects.

Table 2. Runtime, Frame Rates and CPU Usage of DVE (Adding Workload)

Experiment	Runtime (s)	Frame Rate (fps)	CPU Usage %
1	0.040	150	53.22
2	0.040	150	51.95
3	0.050	160	53.38
4	0.048	160	50.89
5	0.050	155	51.14
6	0.060	160	52.73
7	0.050	160	52.64
8	0.043	150	50.32
9	0.040	150	51.87
10	0.040	150	52.36
11	0.050	150	53.29
12	0.050	150	52.96
13	0.040	150	52.78
14	0.040	155	52.83
15	0.040	150	52.62

4.3.1. Runtime Measurement

Table 3 shows the runtime in which the DVE runs the AABB for collision detection. From table 3, we can see that runtime of DVE is from 0.001 to 0.002 second. The average runtime of DVE for scenario 2 is 0.001 second while the average runtime of scenario 1 is 0.007 second. It means that the runtimes of scenario 2 decrease significantly as we compare to table 1.

Figure 14 shows the difference of runtime results between both architectures. The runtime of OBS architecture is smaller than common DVE. The runtime of OBS is almost 0.001 second for all experiments besides of experiment 4. Therefore, its trend line is a constant. Thus, if we compared to scenario 1, the scenario 2 has smaller processing time to handle the collision in DVE. It is the advantage of using OBS because each object is managed by the separated simulator. The processing time of collision handling is addressed by each simulator. Therefore, the runtime of OBS is almost constant and always smaller than the existing DVE.

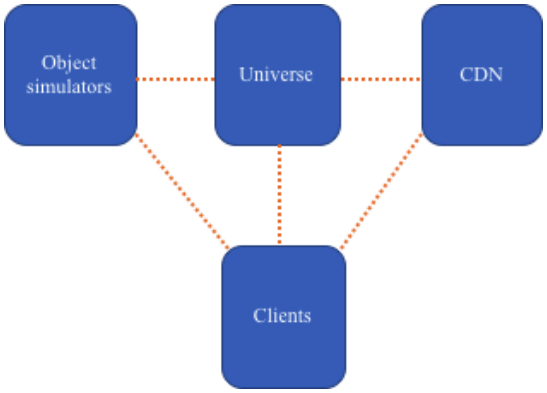

Figure 13. OBS client server architecture

ISSN 2723-6471

Table 3. Runtime of DVE (Scenario 2)

Experiment	Runtime (s)	Experiment	Runtime (s)
1	0.001	9	0.001
2	0.001	10	0.001
3	0.001	11	0.001
4	0.002	12	0.001
5	0.001	13	0.001
6	0.001	14	0.001
7	0.001	15	0.001
8	0.001		

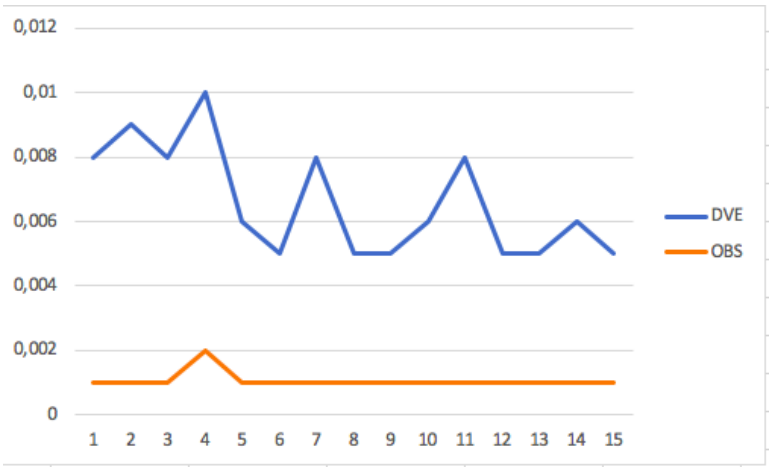

Figure 14. DVE and OBS runtime

Table 4. Frame Rates of DVE (Scenario 2)

Experiment	Frame Rate (fps)	Experiment	Frame Rate (fps)
1	240	9	240
2	240	10	240
3	240	11	240
4	240	12	240
5	236	13	240
6	240	14	240
7	240	15	240
8	236		

4.3.2. Frame Rates Measurement

The frame rates of DVE for scenario 2 are illustrated by table 4. It is measured in frame per second (fps). This frame rates are almost stable for all experiments. Their value is almost 240 fps except the experiment 5 and experiment 8. The frame rates of both experiments decrease to 236 fps. These values are able to deliver smooth environment display for user. If we compare to scenario 1, the OBS has higher frame rates than the common existing DVE. The average frame rates of scenario 1 and scenario 2 are 234 fps and 239 fps, respectively. It means that the OBS is able to deliver better user experiences in running the AABB in distributed virtual environment.

 ISSN 2723-6471

This better performance is also be shown by figure 15. The frame rates of OBS on each experiment are always higher than the common existing DVE. From figure 15 we can see that OBS frame rates tend to constant. It means that the OBS architecture can maintain the performance of DVE. It is not affected by the number of objects reside in DVE because each object is handled by a simulator. The higher frame rates of application the richer experiences able to get by the users. It is important aspect for the 3D application to meet this requirement.

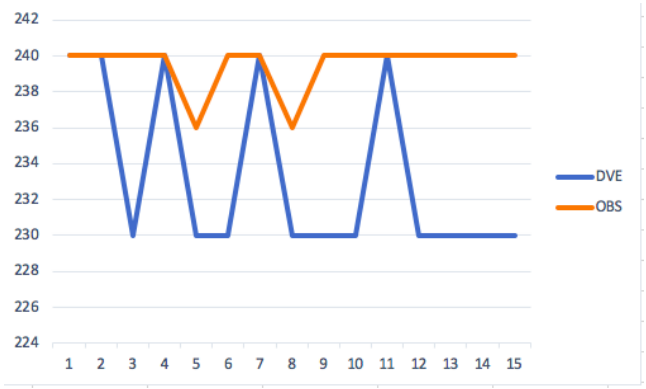

Figure 15. DVE and OBS frame rates

4.3.3. CPU Usages Measurement

Table 5 shows the CPU usage of DVE for scenario 2. From table 5, the average CPU usages of OBS is 20.94%. If we compare to scenario 1, we can see that CPU usages of scenario 2 is smaller than scenario 1 (50.28%). The difference is very strictly, the CPU usages of scenario 2 is half of scenario 1. It means that dividing simulator based on the number of objects is able to decrease the CPU usage significantly.

Table 5. CPU Usages of DVE (Scenario 2)

Experiment	CPU Usage %	Experiment	CPU Usage %
1	20.58	9	21.49
2	20.23	10	20.96
3	20.92	11	21.10
4	21.18	12	20.32
5	20.20	13	20.80
6	20.84	14	21.38
7	20.65	15	21.75
8	21.72		

Figure 16 shows the difference of CPU usages between scenario 1 and scenario 2. All experiments show that CPU usages of OBS always half of the existing DVE. The trendline shows that CPU usages of OBS always smaller than the existing DVE. It means that the performance of OBS is higher than existing DVE in order to manage the objects collision. This OBS performance is also better compared to last research [28]. It explains that the specification of computer used in the research also affects the performance of DVE.

ISSN 2723-6471

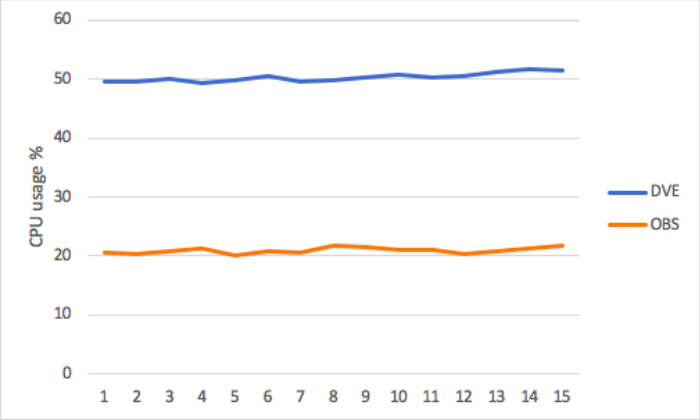

Figure 16. DVE and OBS CPU usages

Based on the runtime, frame rates and CPU usages measurements of implementing AABB in OBS architecture, we can see that the separation of simulator based on the number of objects is able to increase the performance of DVE. In other words, the OBS architecture is able to divide the workload of simulator in handling the object collision in DVE. Automatically, it affects the decreasing of the runtime and CPU usages. It also affects the increasing of DVE frame rates.

5. Conclusions

This paper has run the AABB method in DVE. One of important things found in this research is that the computation complexity must be considered in the DVE. DVE is different from common applications. The DVE always updates its environment. It has to update all objects behaviors and appearances. This characteristic causes the computation time to execute any addition workload given to the AABB method increases significantly. It becomes a reason that many DVE applications still use the AABB to manage the collision detection. Adding workload to AABB also degrades the DVE frame rates. The frame rates become smaller than the common DVE. It yields less user experiences automatically.

To address the problem, OBS architecture has proven that AABB can be implemented better in the DVE. The average runtime of OBS (0.001 second) is smaller than common DVE (0.007 second). The average frame rates of OBS (239 fps) is higher than common DVE (234 fps). Finally, the average CPU usages of OBS (20.94%) is smaller than common DVE (50.28%). Thus, the measurement of three parameters shows that the enhanced AABB implementation in DVE is able to deliver rich user experiences.

6. Declarations

6.1. Author Contributions

Conceptualization: E., T.H., and S.; Methodology: E., and T.H.; Software: E.; Validation: E. and T.H.; Formal Analysis: E., T.H., and S.; Investigation: E.; Resources: E., and T.H.; Data Curation: E., and T.H.; Writing Original Draft Preparation: E., T.H., and S.; Writing Review and Editing: E.; Visualization: E.; All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

The data presented in this study are available on request from the corresponding author.

6.3. Funding

The authors received financial support for the research, authorship, and/or publication of this article.

6.4. Institutional Review Board Statement

Not applicable.

6.5. Informed Consent Statement

Not applicable.

6.6. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Parsons, P. Mitchell, and A. Leonard, "Do adolescents with autistic spectrum disorders adhere to social conventions in virtual environments?," *Autism*, vol. 9, no. 1, pp. 95–117, 2005.

[2] M. Zhang, H. Ding, M. Naumceska, and Y. Zhang, "Virtual Reality Technology as an Educational and Intervention Tool for Children with Autism Spectrum Disorder: Current Perspectives and Future Directions," *Behavioral Sciences*, vol. 12, no. 5, pp.1-33, 2022.

[3] N. Glaser and M. Schmidt, "Systematic Literature Review of Virtual Reality Intervention Design Patterns for Individuals with Autism Spectrum Disorder," *International Journal of Human-Computer Interaction*, vol. 38 no. 8, pp. 753-788, 2022.

[4] B. Sabetghadam, R. Cunha, and A. Pascoal, "A distributed algorithm for real time Multi-Drone Collision free trajectory replanning," *Sensors*, vol. 22, no. 5, pp. 1-21, 2022.

[5] H.A. Sulaiman and A. Bade, "Continuous collision detection for virtual environments: A walkthrough of techniques," *Electronic journal of computer science and information technology*, vol 3, no. 1, pp. 1-7, 2011.

[6] G.V.D. Bergen, "Efficient collision detection of complex deformable models using AABB trees," *Journal of Graph Tools*, vol. 2, no. 4, pp. 1-13, 1997.

[7] D. Hou, X. Wang, J. Liu, B. Yang, and G. Hou, "Research on Collision Avoidance Technology of Manipulator based on AABB Hierarchical Bounding Box Algorithm," in Proc. 5th Asian Conference on Artificial Intelligence Technology (ACAIT), vol. 2022, no. 3, p. 11, 2022.

[8] J. R. Li, R. H. Xin, Q. H., Wang, Y. F. Li, and H. L. Xie, "Graphic-enhanced collision detection for robotic manufacturing applications in complex environments," International Journal of Advanced Manufacturing Technology, vol. 130, no. 12, pp 3291–3305, 2024.

[9] M. Tang, S. Curtis, S.E. Yoon, and D. Manocha, "ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling," IEEE Transactions on Visualization and Computer Graphics, vol. 15 no. 4, pp. 544-557, 2009.

[10] S. Curtis, R. Tamstorf, and D. Manocha, "Fast collision detection for deformable models using representative-triangles," in Proc. The 2008 Symposium on Interactive 3D Graphics and Games, vol. 2008, no. 2, pp. 61-69, 2008.

[11] E. Shellshear, F. Bitar, and U. Assarsson, "PDQ: parallel Distance Queries for Deformable meshes," Graphical Models, vol. 75, no. 2, pp. 69-78, 2013.

[12] X. Zhang and Y. J. Kim, "Scalable Collision Detection Using p-Partition Fronts on Many-Core Processor," IEEE Transactions on visualization and Computer Graphics, vol. 20, no. 3, pp. 447-456, 2014.

[13] M. Tang, D. Manocha, and R. Tong, "MCCD: Multi-core collision detection between deformable models using front-based decomposition," Journal of Graphical Models, vol. 72, no. 2, pp. 7-23, 2010.

[14] L. Xiao, G. Mei, S. Cuomo, and N. Xu, "Comparative investigation of GPU-accelerated triangle-triangle intersection algorithms for collision detection," Multimedia Tools and Applications, vol. 81, no. 1, pp. 3165-3180, 2022.

[15] T. H. Wong, G. Leach, and F. Zambetta, "An adaptive octree grid for GPU-based collision detection of deformable objects," The Visual Computer, vol. 30, no. 5, pp. 729-738, 2014.

[16] P. Du, J. Y. Zhao, W. B. Pan, and Y. G. Wang, "GPU Accelerated Real-Time Collision Handling in Virtual Disassembly," Journal of Computer Science and Technology, vol. 30, no. 5, pp. 511-518, 2015.

[17] Y. Tian, Y. Hu, and X. Shen, "A multi-GPU finite element computation and hybrid collision handling process framework for brain deformation simulation," Computer Animation & Virtual Worlds, vol. 30, no. 1, p. 16, 2019.

[18] P. Du, E. S. Liu, and T. Suzumura, "Parallel continuous collision detection for high-performance GPU cluster," in Proc. The 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, vol. 2017, no. 2, pp. 1-7, 2017.

 ISSN 2723-6471

[19] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast BVH construction on GPUs," Computer Graphics Forum, vol. 28, no. 2, pp. 375-384, 2009.

[20] K. W. Choi, D. Negrut, and D. G. Thelen, "GPU-based algorithm for fast computation of cartilage contact pattern during simulations of movement," in proc. ASME 2013 Summer Bioengineering conference, vol. 2013, no. 6, pp.1-2, 2013.

[21] M. Tang, R. Tong, R. Narain, C. Meng, and D. Manocha, "A GPU-based streaming algorithm for high-resolution cloth simulation," Computer Graphics Forum, vol. 32, no. 7, pp. 21-30, 2013.

[22] Q. Avril, V. Gouranton, and B. Arnaldi, "Collision detection: broad phase adaptation from multi-core to multi-GPU architecture," Journal of Virtual Reality and Broadcasting, vol. 11, no. 6, pp. 1-13, 2014.

[23] S. Pabst, A. Koch, and W. Straber, "Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces," Computer Graphics Forum, vol. 29, no. 5, pp. 1605-1612, 2010.

[24] D. Kim, J.P. Heo, and S. E. Yoon, "HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs," Computer Graphics, vol. 28, no. 7, pp. 1791-1800, 2009.

[25] L. Qi, C. Zhang, G. Xu, and L. Ma, "Autonomous Sweep Modeling and Collision Detection of Underground Pipelines in 3D Environment," in Proc. 2020 5th International Conference on Control and Robotics Engineering (ICCRE), IEEE, vol. 2020, no. 4, pp. 213-217, 2020.

[26] Elfizar, M. S. Baba, and T. Herawan, "1P1O: A large scale distributed virtual environment," Lecture Notes in Electrical Engineering, vol. 520, no. 8, pp 21-29, 2019.

[27] Elfizar, M. S. Baba, and T. Herawan, "A large scale distributed virtual environment architecture," Studies in Informatics and Control, vol. 24, no. 2, pp. 159-170, 2015.

[28] Elfizar, M. S. Baba, and T. Herawan, "Object-based viewpoint for large-scale distributed virtual environment," Malaysian Journal of Computer Science, vol. 28, no. 4, pp. 301-17, 2015.

[29] Elfizar, M.S. Baba, and T. Herawan, "Object-based simulators for large scale distributed virtual environment," Lecture Notes in Electrical Engineering, vol 520, no. 8, pp 11-19, 2019.

[30] B. Czerkawski, "Immersive Learning Experiences through Open Source Virtual Worlds," in Proc. World Conference on Educational Multimedia, Hypermedia & Telecommunications, vol. 2011, no. 6, pp. 3783-3785, 2011.