Bounding Volume Hierarchies for Collision Detection

Chapter · March 2012

DOI: 10.5772/35555 · Source: InTech

2 authors:

Hamzah Asyrani Sulaiman
Technical University of Malaysia Malacca
113 PUBLICATIONS **262** CITATIONS
SEE PROFILE

Abdullah Bade
Universiti Malaysia Sabah (UMS)
107 PUBLICATIONS **291** CITATIONS
SEE PROFILE

Bounding Volume Hierarchies for Collision Detection

Hamzah Asyrani Sulaiman[1] and Abdullah Bade[2]
[1]*University Teknikal Malaysia Melaka, Durian Tunggal, Melaka,*
[2]*University Malaysia Sabah, Kota Kinabalu, Sabah,*
Malaysia

1. Introduction

In virtual environment world, performing collision detection between various 3D objects requires sophisticated steps to be followed in order to properly visualize their effect. It is challenging due to the fact that multiple objects undergo various motion depending on the application's genre. It is however an essential challenge to be resolved since it's many use in the computer animation, simulation and robotic industry. Thus, object intersection between rigid bodies has become one of the most important areas in order to bring realism to simulation and animation.

Rigid bodies stand for geometric models that are fixed and assumed to remain static until there is some force being applied on it. In collision detection case, when two geometric models have collided, the system would notice that both objects couldn't change it dimensions and sizes. Any deformation to rigid bodies is neglected because of this behaviour and the collisions only affect location or movement of both objects. Since in the early era of 3D simulation and animation, problems prevailed in detecting object interference parts, and numerous attempts by researchers have been made to find the solution of the collision detection between rigid bodies. Baraff has made one of the earliest researches concerning detecting object interference between rigid bodies (Baraff, 1989).

Later in 1993, M.C. Lin conducted a research of detecting object interference between two rigid bodies. M.C. Lin defined that there are two types of contact between rigid bodies that could be identified. They are tangential collision and boundary collision (Lin, 1994). Tangential collision happens when there is intersection between two surfaces at 90 degrees at geometric contact point. It means that the collision happens either from 90 degrees from above, bottom, right or left surfaces of corresponding polygons. Meanwhile boundary collision occurred when there is one object wants to check for potential collision from inside the object boundary. For example, a circle has it owns boundary made of certain radius. If one point inside this radius has intersected with another circle, then boundary collision has occurred. (Lin, 1994)

Whilst, Redon explained that there are two common types of performing collision detection between rigid bodies namely discrete collision detection (DCD) and continuous collision detection (CDC) (Redon et al., 2002). DCD is performed by sampling the object motion towards the object that is going to be intersected and detect the object interpenetrations (Tu and Yu, 2009, Rocha and Maria Andre'ia Formico, 2008, Kockara, 2007, Bade et al., 2006, Klosowski et al., 1998, Baciu, 1998, Cohen et al., 1995, Garcia-Alonso et al., 1994, Cohen et al., 1994, Gilbert and Foo, 1990, Baraff, 1990). DCD is the approach that has been used by researchers to perform collision detection in term of speed. CDC, on the other hand, computes from the first time of contact when object collided. It is much slower in comparison to DCD method because CCD focuses on accuracy of collision detection.

In most 3D applications, DCD is the most useful compared to CCD. The differences between these two are on their manipulation of time and object movement. DCD is preferred because of its simplicity and fast collision detection compared to the CCD algorithm. DCD checks the intersection within a fix time instant while CCD algorithm uses the object trajectory. As checking for object trajectory requires future object location to be known in very small time frames, the CCD algorithm cannot be performed as fast as DCD algorithm. However, CCD is useful for accurate collision detection where high precision is needed. It also has a small false positive (collision miss) compared to the DCD algorithm. As an example, an application such as clothes, medical, hair or any deformable bodies' simulation requires collision of the object to be detected at high precision state where it involves accurate collision detection algorithm. Fast collision detection algorithm on the other hand is useful for computer games development where the response of potential colliding objects need to be known as fast as possible.

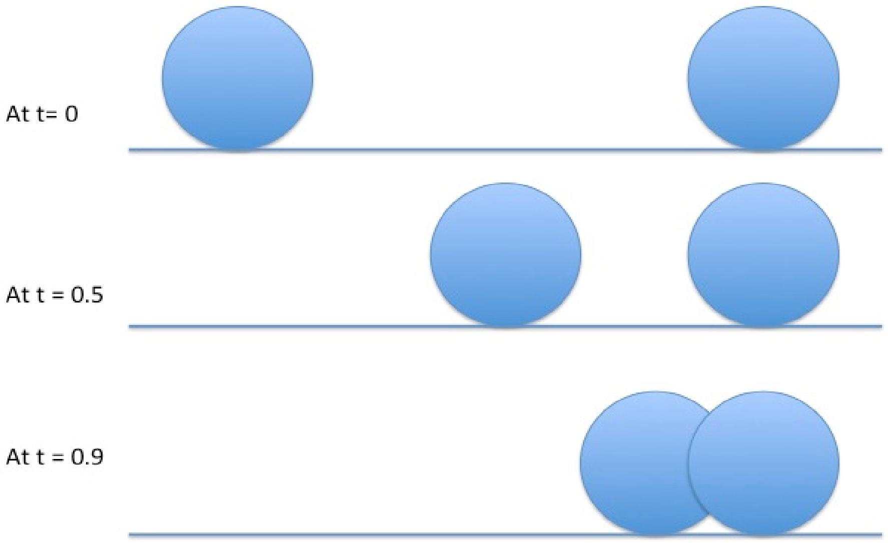
In DCD algorithm, hierarchical representation for object is commonly used (Tu and Yu, 2009, Larsson, 2009, Chang et al., 2008, Liu et al., 2007, Nguyen, 2006). By enveloping the object in virtual environment with hierarchical representation, the detection time could be reduced by performing collision checking for specific node in hierarchical tree. Bounding-Volume Hierarchies (BVH) is the most fashionable approach used by researchers to perform collision checking. It represents the object into hierarchical manners where each node contains single Bounding-Volume (BV) that enclosed set of triangles (Larsson and Akenine-Moller, 2008, Sobottka and Weber, 2005, Chang et al., 2008, Liu et al., 2007).

Figure 1 illustrates the different between CCD and DCD algorithm.

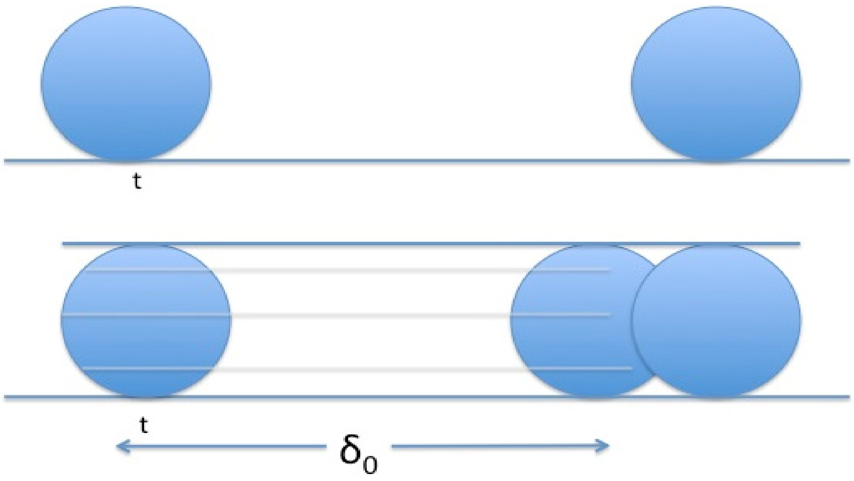
In order to perform fast DCD approach, BVH itself must be able to have very efficient node so the traversal algorithm for the BVH could traverse as fast as possible avoiding any unnecessary node that does not involve in collision. Thus, researchers has come out with solution by using heuristic, it could produces balance BVH tree where until specific level of BVH tree. Once the balance BVH tree had reached, the primitive-primitive testing inside leaf nodes will be tested against another BVH tree. However, it still suffers from limitation of performing fast response to the collision detection due to balance level that had been generated is not well-organized where some nodes contains unbalance set of triangles or size of BV.

2. Hierarchical approaches

Virtual environment is composed of many objects that could be static or in motion, where each objects may have thousands of primitives. Testing object interference between these primitives could become troublesome as the number of pair intersection tests that need to be performed is exploding (Bergen, 2004). Hence, spatial data structures is becoming one of the



(a)



(b)

Fig. 1. (a) Discrete Collision Detection - During the detection, object might be sampling for example, every 0.1 seconds. While (b) Continuous Collision Detection - During the detection, every object movement is calculated in order to find the accurate time of contact.

efficient solution for accelerating collision detection checks in massive environments (Bergen, 2004). According to (Bergen, 2004) for n objects there are $\binom{n}{2} = \frac{1}{2}n(n-1)$ potentially collided pairs. There are two types of spatial data structures being used for collision detection: spatial division and BVH. Spatial partitioning divides the spaces of virtual environment and causes the environment to be divided into cells while BVH only divides each object that exists in virtual environment. However for spatial partitioning case, when it comes to large-scaled simulation, the tree depth increases and it will slow down the performance of the simulation.

Furthermore, since spatial partitioning divide the cell into smaller cells, it cannot cover the object very tightly as bounding-volume could. Thus, the accuracy of detecting object interference between environments that use spatial partitioning might decrease since it might report false positive intersection. Bounding-volume hierarchy provides a better solution to the researchers by providing simpler and tighter tree hierarchy. Figure 2 depicts the BVH tree.

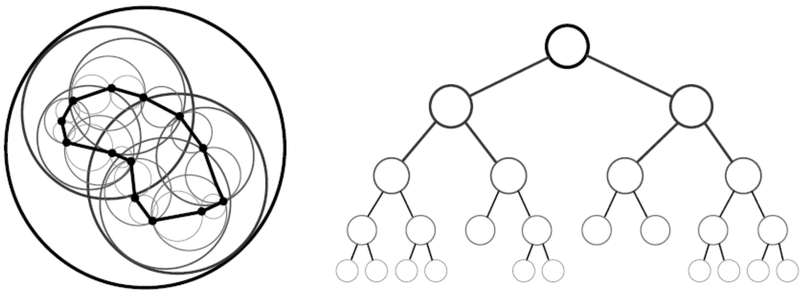


Fig. 2. The left hand side image shows a BVH with Sphere BV while on the right hand side image, shows unbalanced hierarchical form using binary type hierarchy

2.1 Hierarchy characteristics

Figure 3 describes in general of hierarchical tree characteristic. Haverkort suggested useful characteristics of hierarchy tree using bounding-volumes but also suitable for any kind of hierarchical (Haverkort, 2004).

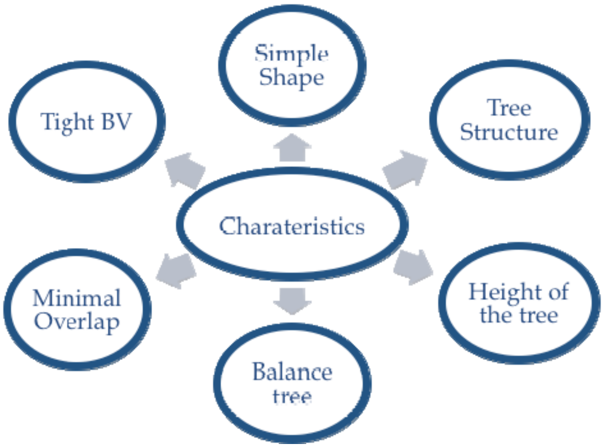


Fig. 3. Hierarchical Tree characteristics

As illustrated in Figure 3, the main important attribute is to find a suitable bounding-volume (BV) for the targeted application. If the application requires an accurate collision detection scheme especially for medical and fluid simulation, then tight BV is needed. For computer games development and fast response simulation, simple and low count surfaces BV is the most suitable. Examples of them include the Axis-Aligned Bounding-Box (AABB), Oriented Bounding-Box (OBB), and Sphere. This is due to the reason that when intersection occurs, it first checks for the intersection points between bounding volume without calculating the complex primitives in geometric models. Then, when there is an intersection between top-level bounding-volume (root of hierarchy), further checks between children nodes of hierarchy will be conducted down through the tree until the correct intersection is found. However, there is trade-off between simple and tight bounding-volume. Simple bounding-volume seems to perform faster intersection test while tight bounding-volume goes for the accuracy but slow intersection test.

The structure of the tree is also one of the important elements in tree building since it determines the speed of the tree traversal when collision needs to check part of the primitives bounded by the BV. For example, the binary-type tree is faster than quad-type tree in term of reaching the bottom of the tree. The binary-type tree only needs to check for two nodes that are either left or right node compared to the quad-type tree that needs to checks four nodes. But in term of the accuracy, sometimes it depends on the intersection itself as quad-type tree can spend less time than binary-type tree if the quad-tree managed to find the collision at earlier level of the tree hierarchy. The number of primitives reside in the nodes also depends on well-organized algorithm that is used to split up the upper node into several nodes (depends again on the type of the tree).

It also needs to be determine the height of the tree by own algorithm as the process can be lengthy and could also ultimately leads to infinite tree height (when set of triangle failed to split themselves and repeat the process all over again). Faster traversal can be achieved if the algorithm used to split up the tree is successful at dividing the nodes into certain level (less is good) and carried out primitives-primitives testing or one BV one triangle testing. One BV one-triangle testing is the primitive checking using the BV itself. Whereas the triangle testing could lead to a faster collision checking but could also bring false positive result if the BV is not a suitable one. Faster traversal from root to specific node can be achieved if the height of the hierarchy is small but it might not always be sufficient.

Creating a balance tree hierarchy for the object is an important task for collision detection in order to improve result speed. As the height of the tree is getting deeper and longer, the tree must be able to well balance on it as it affects the traversal processes. Giving an example of binary-tree traversal system when the left node is deeper than the other one (right node), it will slow down the process of collision checking at the particular nodes. Thus it is essential that the tree hierarchy is kept balanced out with respect to nodes, volume and density.

The design and the construction of the hierarchy tree must also have minimal overlap between its volumes. This is to make sure that the process of tree traversal does not check for the same intersection between nodes multiple times. The last characteristic is to use appropriate tight BV with an appropriate hierarchy tree type for our application. Tight BV is used for accurate collision checking between other objects that are also bounded by the tight BV. K-Dop, Elipsoid, and Convex Hull are the examples of the BV that can tightly bounded the object but mainly used for the accurate collision detection.

2.2 Bounding-volume

Generally in any simulation, computation is very high when detecting interference between two rigid bodies. For example, a car model with 10,000 polygons surface hit the building that has 40 polygons surface is waiting for signal that object has been intersected. Each polygon surface will be checked one by one to find which parts of the car and the building has come into intersection. Even though the computer may be able to calculate in milliseconds, but imagine if the simulation environment consists of multiple objects that each of them has tens of thousands polygon surfaces. So, one of the way to overcome this is by implementing the bounding-volume.

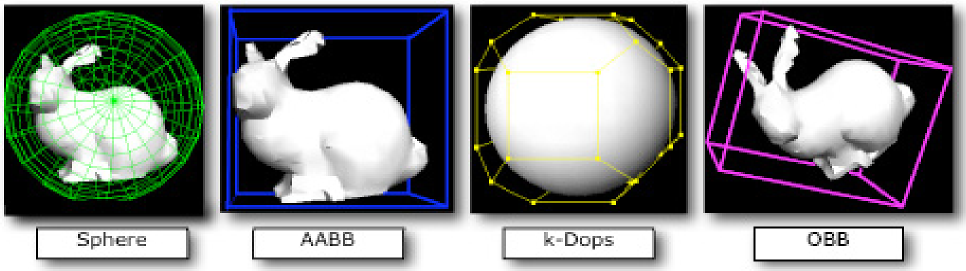


Fig. 4. Common Bounding Volume in previous researches (Suaib et al., 2008, Bade et al., 2006).

The purpose of using BV is to reduce the computational cost to detect object interference. If the object performs primitive-primitive testing without applying BV, it could consume longer time, as it needs to check each triangle with other object triangle set. However, time to check for each collision can be reduced through enveloping highly complex object with BV. Instead of using single BV, BVH could help performing collision detection better than a single BV. BVH provides a hierarchical representation that could split the single BV into certain level before performing primitive-primitive testing for accurate collision detection. It can also be used for fast collision detection method by stopping at certain level using stopping function or criteria and approximately response to the collision as object that has been collided. It depends heavily on what kind of application that been developed as some application prefers speed and others accuracy.

At present time, there are several famous bounding volumes such as spheres (Weller and Zachmann, 2009, Madera et al., 2009, Chang et al., 2009, Rocha and Maria Andre'ia Formico, 2008, Larsson et al., 2007, Spillmann et al., 2007, Benitez et al., 2005, Gareth and Carol, 2004, Bradshaw and O'Sullivan, 2002), Axis Aligned Bounding Box (AABB) (Tu and Yu, 2009, Zhang and Kim, 2007, Weller et al., 2006), Oriented Bounding Box (OBB) (Tu and Yu, 2009, Chang et al., 2009, Gottschalk et al., 1996), Discrete Oriented Polytope (k-DOP) [3], new type of bounding volume; Oriented Convex Polyhedra (Suaib et al., 2008, Bade et al., 2006), and hybrid combination bounding volume (Tu and Yu, 2009, Kockara, 2007). Most large scale 3D simulations used bounding box because it iss simple, require small space of storage, fast response of collision, and easy to implement (Lin and Manocha, 2004). Figure 4 illustrates most commonly used bounding volume.

2.3 Hierarchical tree building strategies

There are three different ways to construct a tree: top down, bottom up, or insertion method. Top down methods can be presented as a growing tree of two or more subsets while the bottom up grouped the subsets to form internal node until they becomes root node. Insertion methods can be implemented by inserting one object at one time into the tree. Each type has its own unique characteristic as shown Figure 5.

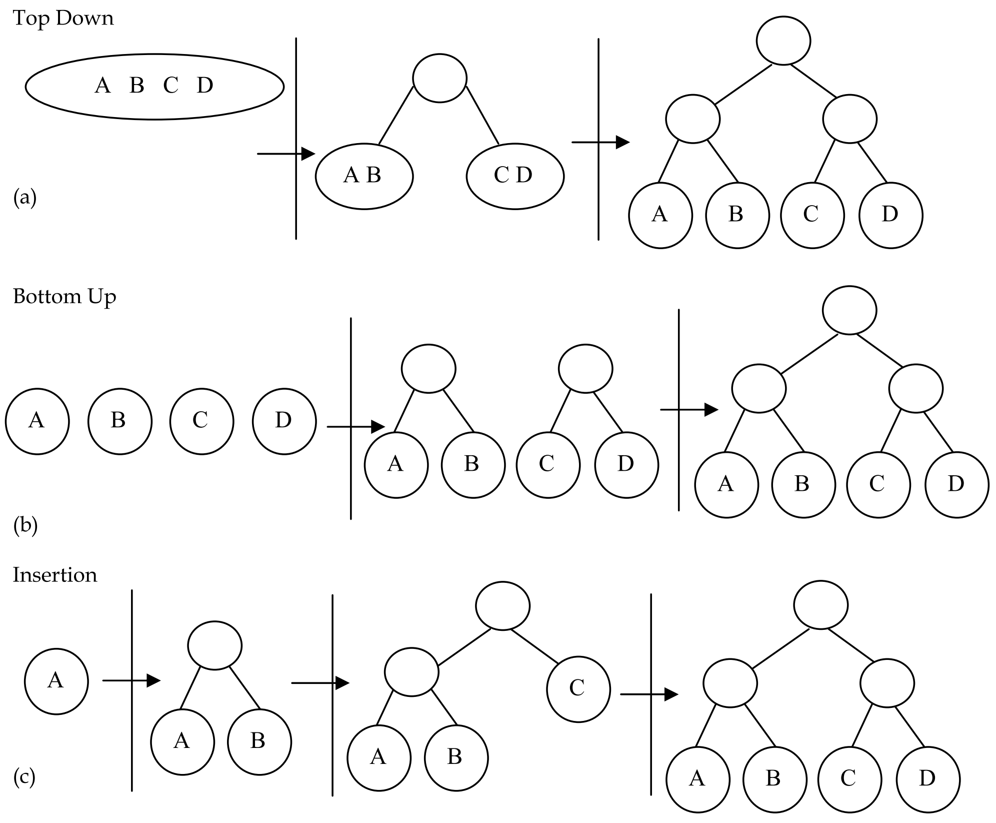


Fig. 5. Hierarchy tree based of four objects using (a) top-down, (b) bottom-up, and (c) insertion construction (Ericson, 2004).

Among the three, top-down approach is the most popular technique to be used to construct hierarchy tree. (Gottschalk, 2000, Gottschalk et al., 1996) proposed an OBB-tree while (Tu and Yu, 2009) constructed binary AABB BVH that use top-down hierarchy where each set of primitives is tightly bounded with bounding-volume. However, depending on the application type, they might not always produce the best result. From Figure 5, top-down approach seems to fully cover the objects with large bounding-volume for example sphere. Then, it is recursively partitioned into two separate parts that has smaller spheres, that is connected to the root of the hierarchy. These two separate parts will then keep partitioning until a particular stopping criteria is reached or only a single primitive is available. Top-down approach commonly use splitting algorithm in order to find the correct splitting of the tree.

In contrary, bottom up approach are more complicated to put into as it has slower construction time but it is efficient in producing the best tree (Omohundro, 1989). From Figure 5, each primitive will be bounded with the smallest bounding-volume. Then, these volumes will be grouped together to perform the upper nodes called leaf nodes. However, from this point, some merging criterion or clustering rule must be followed in order to merge two or more leaf nodes into parent nodes. Afterward, these nodes will be enclosed with a bounding-volume and replace the original set of nodes. This procedure will continue until a single bounding-volume is left that will become root of the hierarchy tree. Bottom-up approach regularly use merging algorithm when merging two or more leaf nodes into parent nodes.

The third hierarchy type is the insertion method or incremental method. Starting from an empty tree, single primitive will be inserted one by one at a time by finding the insertion location. It depends fully on the cost metric. Incremental insertion algorithm as presented (Goldsmith and Salmon, 1987) in Figure 6 was the first algorithm to construct incremental hierarchy tree. It used cost function to control the insertion or primitives into hierarchy tree. First the algorithm estimates the cost of inserting primitives at different places in the hierarchy, and then the algorithm automatically picks the lowest cost and the cheapest place to insert primitives into tree. There are three rules that need to be followed when a primitive needs to be inserted into as tree (Goldsmith and Salmon, 1987) :-

1. *p* can become the child of parent *o*
2. *p* can be combined with another *q* of a group *o* to establish new group called *o′* which automatically becomes child of *o*.
3. *p* can becomes child of parent *o′* of a group *o* recursively. Then, partially formed hierarchy is traversed until either case 1 or 2 is reached.

However, the problem of this method is that it can become worst as it depends on the insertion order of the nodes. Various insertion methods can be designed but it is challenging to find the best one. Apart from that, a well-balanced hierarchy tree can become a totally an unbalanced tree (worst case) if no proper insertion algorithm is used.

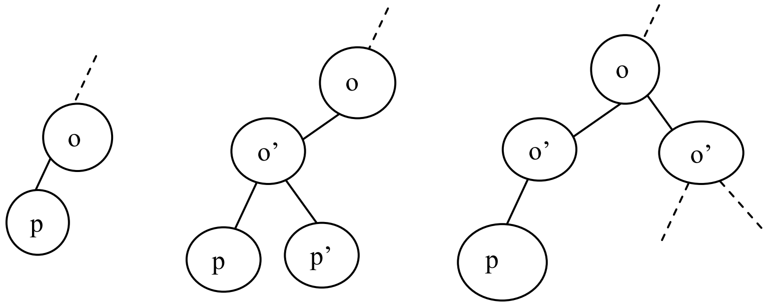


Fig. 6. Rule 1(left) group o has p as a child. Rule 2 (middle) merging two primitives to create new o′. Rule 3(right) recursively insert primitive into parent nodes (Goldsmith and Salmon, 1987).

2.4 Partitioning strategies and splitting algorithm

The simplest way to partition an object is to split the object equally with respect to local coordination axes of the object. The split can be done according to the median-cut algorithm style or several other strategies as follows(Ericson, 2004):-

- *Minimize the sum of the volumes (or surface areas) of the child volumes*. Minimizing the sum of value can effectively minimize the probability of intersection - tightness bounding volume must be used.
- *Minimize the maximum volume (surface area) of the child volumes.* Divides the child volume into equal size by making larger volume as small as possible.
- *Minimize the volume (surface area) of the intersection of the child volumes.* Complex to be implemented as it depends on the tightness of bounding volume.
- *Minimize the separation of the child volumes.* Where each child will be separated, it helps to decrease the probability of both children being traversed at the same time.
- *Divide primitives equally between the child volumes*. Divide the object into two equal parts as mentioned earlier.
- *Hybrid combination of multiple bounding-volumes*

(Müller et al., 1999) had explained the splitting criteria that have been used to partition a set of primitives into two subsets until certain stopping criteria are reached. In this optimized hierarchical method, each object will be divided into several pieces called subsets. By using a certain cost function, it can be ensured that the best possible splitting plane is used. However, the splitting procedure is applied only to the bounded object. This means that no BV will be decomposed into pieces.

Normally, root of the hierarchy is created by sorting each primitive using its centres. A cost function is then used to determine the best possible partitions that split the object volume into two parts or subsets. The splitting parts or subsets constitute the left and right side of the hierarchy. Then splitting algorithm continues to repeat the above process recursively until there is only one big BV enclosing the whole object.

2.4.1 Choosing the splitting point

There are some options available to determine splitting point along the axes (Ericson, 2004) (please see Figure 7):-

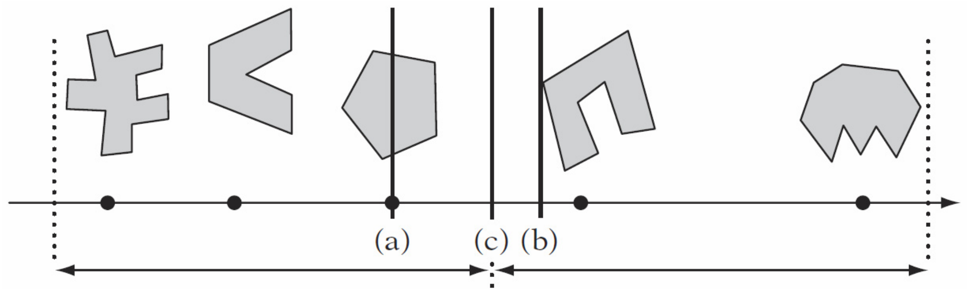


Fig. 7. An example of (a) Object median splitting (b) Object mean splitting (c) Spatial median splitting based on (Ericson, 2004)

- *Median of the centroid coordinates* (object median) – split at the object median (distributed parts) resulting a well-balanced tree.
- *Mean of centroid coordinates* (object mean) – (Klosowski et al., 1998) stated that better performance is obtained using object mean compared to object median. It was claimed that splitting at the object mean gives smaller volume trees.
- *Median of the bounding-volume projection extents* (spatial median) – splitting the volume into two equal parts.

Meanwhile, (Gottschalk et al., 1996) constructed OBB trees using top-down approach. The first procedure is to enclose all primitives by an oriented bounding box, and the recursive partition is made possible by using certain rules. Their subdivision rule proposed as applied to the object is to find the longest axes and the splitting point is determined through an orthogonal plane on one of its axes. Objects are partitioned according to the side of the splitting point but this does not involve the primitive. Figure 8 and 9 depicts the splitting example (Gottschalk et al., 1996).

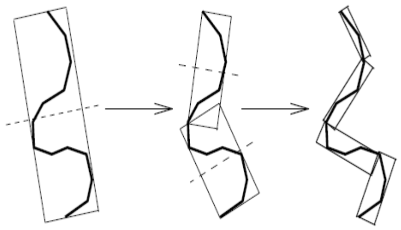


Fig. 8. Splitting along the farthest axis of object bounded with OBB using centre points (Gottschalk et al., 1996).

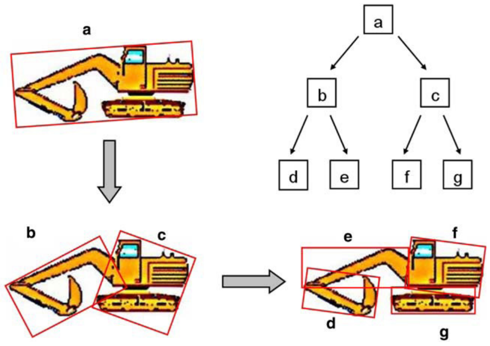


Fig. 9. OBB tree recursively divides the object into two parts using binary tree and top-down approach (Kamat and Martinez, 2007).

2.4.2 Merging algorithm

Merging algorithm has been described in (Erleben et al., 2005), and its implementation can be found from the Open Tissue website. The algorithm started by building graph of the data structure. Nodes in the graph correspond to the primitives and the edges on the other hand correspond to the nearest neighbour relations. Here, an edge in the graph indicates that two

BVH nodes are good candidates to be merged into group. A heuristic function is used to determine collisions and large primitive BVs into one single BV. A collision here means that any edge between two colliding nodes in the graph must be added into the graph. Figure 10 depicts the operation of grouped BVs.

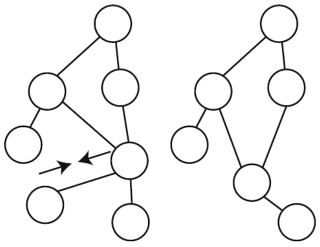


Fig. 10. Edge collapsed and merged into one single node.

3. Bounding volume hierarchies

BVH is simply a tree structure that represents geometric models with specific bounding volumes. It works like a tree that has a root (upper division), a group of leafs (middle division) and a leaf (last division). Each node has it bounding-volumes that cover the children nodes. The main idea of BVH is to build a tree that has a primary and secondary root where each of the secondary nodes is stored as leaf. BVH allows intersection to occur without searching for non-colliding pairs from the hierarchy tree. For example, given two objects with their BVH, when root of the hierarchies do not intersect, the calculation will not be done for both objects. However, when roots of both hierarchies intersect, it will check for intersection between roots of one of the hierarchy's tree with the children of the other hierarchy's tree. In this case, it recursively checks again whether there is intersection between both objects at middle level until the correct intersection is found. Figure 1 previously shows how object is partitioned into several parts. While Figure 11 depicts the basic algorithms for detecting collision between two hierarchies (Kamat and Martinez, 2007, Nguyen, 2006)

```
Beginning at the root nodes of two given trees
1.    Check for intersection between two parent nodes
2.    If there is no intersection between two parents
3.        Then stop and report "no collision"
4.    Else check all children of one node against all
      Children of the other node
5.    If there is intersection between any children
6.        Then If at leaf nodes
7.            Then report "possible collision"
8.        Else go to Step 4
9.    Else skip and report "no collision"
```

Fig. 11. BVH traversal algorithm proposed by (Nguyen, 2006) for collision detection between two BVH

3.1 BVH construction

The construction of BVH is started by selecting type of the tree. There are multiple types of BVH construction available. One of the most popular tree constructions is binary tree, which has following nodes definition:-

a. **Nodes** – A node may contain specific value or a condition that represents a separate data structure or a tree of its own. In BVH, it contains bounding-volume with their tree ID. Each node may become parent, child or leaf nodes (in computer science tree grow downward where the root is at the top of the tree). The node that produces child nodes is called parent node, superior node or ancestor node. The length starting from root node down to the leaf node determines the height of the tree. The depth of the tree is the length of the path to its root.
b. **Root Nodes** – The topmost node in binary tree is called root node. Actually, root node does not have any parents. It is the starting node where all the operations of the tree will begin normally. Root Node is connected to each child nodes downward by using branch or subtree connector. Branch of subtree connector is one of the important elements that are connecting each node with their parent nodes.
c. **Leaf Nodes** – The bottommost node in binary tree is called the leaf nodes. It does not have any child nodes and mostly contain the last value or conditions. In BVH, it may contain few triangles or maybe one triangle.
d. **Internal Nodes** – The functional node where it has both parent nodes and child nodes. It is not leaf nodes as it still has child nodes and linked by parent node.
e. **Subtree** – A portion of a tree data structure that can be viewed as a complete tree in itself. It represents some other parts of the tree that can be manipulated by user in programming.

Binary approach in top down fashion is selected, as it is the most preferred approach by researchers. It can traverse faster and is very efficient compared to other version of BVH tree. There are few common operations involved in construction and implementation of binary tree for BVH construction. Among others are enumerating all the items, searching for an item which is bounding-volume, removing a whole section of a tree for non-intersected area when performing collision detection (called pruning), traverse down from the root to leaf nodes or leaf nodes to the root, and report any intersected area between two intersected BVH that possible to collide. The construction of BVH is then implemented using C++ language by loading 3D object into the environment. Then, the object vertices is calculated and stored into a temporary location, as it is required for the BVH tree construction.

Next step is to choose a suitable BV for the BVH tree**.** The purpose of using BV is to reduce the computation cost of detecting object interference. If the object performs primitive-primitive testing without BV, it could consume time, as it needs to check each object triangle with other object triangle set. However, the time to check for each collision can be reduced through enveloping highly complex object with BV. Instead of using single BV, BVH is used to achieve improved collision detection. BVH provides a hierarchical representation that could split the single BV into certain level before performing primitive-primitive testing. It can also be used for fast collision detection method by stopping at certain level using stopping function or criteria and approximately response to the collision as the object has been collided.

At present, there are several well-known BVs such as spheres (Liu et al., 2007), Axis Aligned Bounding Box (AABB) (Zhang and Kim, 2007, Weller et al., 2006, Tu and Yu, 2009), Oriented Bounding Box (OBB) (Chang et al., 2009, Gottschalk et al., 1996, Tu and Yu, 2009), Discrete Oriented Polytope (k-DOP) (Klosowski et al., 1998), Oriented Convex Polyhedra (Bade et al., 2006), and hybrid combination BV (Kockara, 2007).

3.2 BVH cost function

This section will describe the overview of hierarchical method that is used in the proposed urban simulation. BVH is proven to be the most efficient method for collision detection (Sulaiman et al., 2009, Chang et al., 2008, Bergen, 2004, Bergen, 1999). Thus, in this research, the hierarchical cost function that has been used by previous researchers will be used. Basic cost function was first formulated by (Weghorst et al., 1984) for hierarchical method in ray tracing and later was applied by (Gottschalk et al., 1996) and enhanced by (Klosowski et al., 1998). The calculation of execution time is formulated as follows:

$$T = N_v \text{ X } C_v + N_p \text{ X } C_p + N_u \text{ X } C_u + C_o \tag{1}$$

Where

T: total execution time for detecting interference
N_v: number of BV pairs overlap tests
C_v: time require for testing a pair of bounding-volumes
N_p: numbers of primitive pairs overlap tests
C_p: time require for testing a pair of primitives for interference
N_u: numbers of nodes that need to be updated
C_u: cost of updating each node
C_o: Cost of one-time processing

From the formula 1, N_v shows the number of the BV that is currently overlapped when the objects has come into contact while N_p shows number of the primitive itself when overlapped. Given of example of two rigid bodies that enclosed with their BVHs. When both objects come into contact, the system will calculated how much BV between these two rigid bodies has been overlapped and we also measure its C_v. Next, the system also will store the information of the number of primitive inside the BVs that need to perform intersection test in order to determine the exact number of primitives that currently overlapping and with the C_p to measure time require to test each primitive-primitive testing. Meanwhile, N_u takes into account the numbers of nodes that need to be updated once the intersection has occurred (where each node has a BV and few primitives inside). C_u is the time taken to update each node while C_o is for any additional time taken for transformation update or coordinate update of each objects.

The formula shows that the most important factors that determine the performance of collision detection between rigid bodies are the tightness of bounding-volumes and the simplicity of bounding-volumes. When we have lower number of overlap tests (lower N_v and N_p) per intersection between two rigid bodies for example, the object must be bounded with tight bounding-volumes and will eventually decrease the potential of object interference hence increase performance. However when we enclosed the objects with simple bounding-volumes (lower C_v and C_p), it is resulting significant increment of the intersection tests between bounding-volumes. Minimizing one value will cause another value to increase. This is the challenge in all collision detection to find which one is the most important.

4. References

Baciu, G. Recode: An Image-based Collision Detection Algorithm. *In:* WONG, W. & SUN, H., eds., 1998. 125-125.

Bade, A., Suaib, N., A, M. Z. & M, T. S. T. 2006. Oriented convex polyhedra for collision detection in 3D computer animation. *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia.* Kuala Lumpur, Malaysia: ACM.

Baraff, D. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Proceedings of the 16th annual conference on Computer graphics and interactive techniques.* ACM.

Baraff, D. 1990. Curved surfaces and coherence for non-penetrating rigid body simulation. *SIGGRAPH Comput. Graph.,* 24**,** 19-28.

Benitez, A., Ramirez, M. D. C. & Vallejo, D. 2005. Collision Detection Using Sphere-Tree Construction. *Proceedings of the 15th International Conference on Electronics, Communications and Computers.* IEEE Computer Society.

Bergen, G. V. D. 1999. A fast and robust GJK implementation for collision detection of convex objects. *J. Graph. Tools,* 4**,** 7-25.

Bergen, G. V. D. 2004. *Collision Detection in Interactive 3D Environments,* United States of America, Elsevier, Inc.

Bradshaw, G. & O'Sullivan, C. 2002. Sphere-tree construction using dynamic medial axis approximation. *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.* San Antonio, Texas: ACM.

Chang, J.-W., Wang, W. & Kim, M.-S. 2008. Efficient Collision Detection Using a Dual Bounding Volume Hierarchy *Geometric Modeling and Processing.* Berlin Heidelberg.

Chang, J.-W., Wang, W. & Kim, M.-S. 2009. Efficient collision detection using a dual OBB-sphere bounding volume hierarchy. *Computer-Aided Design,* In Press, Corrected Proof.

Cohen, J. D., Lin, M. C., Manocha, D. & Ponamgi, M. 1995. I-Collide: an interactive and exact collision detection system for large-scale environments. *Proceedings of the 1995 symposium on Interactive 3D graphics.* Monterey, California, United States: ACM.

Cohen, J. D., Manocha, D., Lin, M. C. & K.Ponamgi, M. 1994. Interactive and Exact Collision Detection for Large-Scale Environments. *Technical Report TR94-005.*

Ericson, C. 2004. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*, Morgan Kaufmann Publishers Inc.

Erleben, K., Sporring, J., Henriksen, K. & Dohlman, K. 2005. *Physics-based Animation (Graphics Series)*, Charles River Media, Inc.

Garcia-Alonso, A., Nicol, Serrano, S. & Flaquer, J. 1994. Solving the Collision Detection Problem. *IEEE Comput. Graph. Appl.,* 14**,** 36-43.

Gareth, B. & Carol, O. S. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph.,* 23**,** 1-26.

Gilbert, E. G. & Foo, C. P. 1990. Computing the distance between general convex objects in three-dimensional space. *Robotics and Automation, IEEE Transactions on,* 6**,** 53-61.

Goldsmith, J. & Salmon, J. 1987. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Comput. Graph. Appl.,* 7**,** 14-20.

Gottschalk, S., Lin, M. C. & Manocha, D. 1996. OBBTree: a hierarchical structure for rapid interference detection. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* ACM.

Gottschalk, S. A. 2000. *Collision queries using oriented bounding boxes.* The University of North Carolina at Chapel Hill.

Haverkort, H. J. 2004. *Results on Geometric Networks and Data Structures.* PhD, Technische Universiteit Eindhoven.

Kamat, V. R. & Martinez, J. C. 2007. Interactive collision detection in three-dimensional visualizations of simulated construction operations. *Engineering with Computers,* 23**,** 79-91.

Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H. & Zikan, K. 1998. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics,* 4**,** 21-36.

Kockara, S. H., T.; Iqbal, K.; Bayrak, C.; Rowe, Richard; 2007. Collision Detection - A Survey. *IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC.*

Larsson, T. 2009. *Adaptive Bounding-Volume Hierarchies for Efficient Collision Queries.* PhD, Malardalen University.

Larsson, T. & Akenine-Moller, T. 2008. Bounding Volume Hierarchies of Slab Cut Balls. Malardalen University.

Larsson, T., Akenine-Möller, T. & Lengyel, E. 2007. On Faster Sphere-Box Overlap Testing. *Journal of Graphics Tools,* 12**,** 3-8.

Lin, M. C. 1994. EFFICIENT COLLISION DETECTION FOR ANIMATION AND ROBOTICS. University of California at Berkeley.

Lin, M. C. & Manocha, D. 2004. Collision and Proximity Queries. *In Handbook of Discrete and Computational Geometry, 2nd Ed.* Boca Raton, FL: CRC Press LLC.

Liu, L., Wang, Z.-Q. & Xia, S.-H. A Volumetric Bounding Volume Hierarchy for Collision Detection. 10th IEEE International Conference on Computer-Aided Design and Computer Graphics, 2007 2007. 485-488.

Madera, F. A., Day, A. M. & Laycock, S. D. A Hybrid Bounding Volume Algorithm to Detect Collisions between Deformable Objects. Second International Conferences on Advances in Computer-Human Interactions, 2009. ACHI '09. , 2009. 136-141.

Müller, G., Schäfer, S. & Fellner, D. W. 1999. Automatic Creation of Object Hierarchies for Radiosity Clustering. *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications.* IEEE Computer Society.

Nguyen, A. 2006. *IMPLICIT BOUNDING VOLUMES AND BOUNDING VOLUME HIERARCHIES.* Doctor of Philosophy, Stanford University.

Omohundro, S. 1989. Five Ball tree Construction Algorithm. *Technical Report TR-89-063.* International Computer Science Institute, Berkeley, CA.

Redon, S., Kheddar, A. & Coquillart, S. 2002. Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum.*

Rocha, R. D. S. & Maria Andre'ia Formico, R. 2008. An evaluation of a collision handling system using sphere-trees for plausible rigid body animation. *Proceedings of the 2008 ACM symposium on Applied computing*. Fortaleza, Ceara, Brazil: ACM.

Sobottka, G. & Weber, A. Efficient Bounding Volume Hierarchies for Hair Simulation. Proc. Second Workshop Virtual Reality Interactions and Physical Simulations (VRIPHYS '05), 2005. 101-110.

Spillmann, J., Becker, M. & Teschner, M. 2007. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *J. Vis. Comun. Image Represent.,* 18**,** 101-108.

Suaib, N. M., Bade, A. & Mohamad, D. Collision Detection Using Bounding-Volume for avatars in Virtual Environment applications. The 4th International Conference on Information & Communication Technology and Systems, August 2008 2008 Institut Teknologi Sepuluh Nopember (ITS), Surabaya, Indonesia. 486 - 491.

Sulaiman, H. A., Bade, A., Daman, D. & Suaib, N. M. 2009. Collision Detection using Bounding-Volume Hierarchies in Urban Simulation. *The 5th Postgraduate Annual Research Seminar.* Faculty of Computer Science & Information System, UTM.

Tu, C. & Yu, L. Research on Collision Detection Algorithm Based on AABB-OBB Bounding Volume. First International Workshop on Education Technology and Computer Science, 2009. ETCS '09. , 2009. 331-333.

Weghorst, H., Hooper, G. & Greenberg, D. P. 1984. Improved Computational Methods for Ray Tracing. *ACM Trans. Graph.,* 3**,** 52-69.

Weller, R. & Zachmann, G. 2009. Inner Sphere Trees. *In:* DIX, J. (ed.). Clausthal-Zellerfeld, Germany: Clausthal University of Technology.

Weller, R. E., Klein, J. & Zachmann, G. A Model for the Expected Running Time of Collision Detection using AABB Trees. *In:* HUBBOLD, R. & LIN, M., eds. Eurographics Symposium on Virtual Environments (EGVE), 8--10 May 2006 Lisbon, Portugal.

Zhang, X. & Kim, Y. J. 2007. Interactive Collision Detection for Deformable Models Using Streaming AABBs. *IEEE Transactions on Visualization and Computer Graphics,* 13**,** 318-329.