Collision Detection Using Bounding-Volume Hierarchies in Urban Simulation

Hamzah Asyrani Sulaiman
Department of Computer Graphics and Multimedia,
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia
81310 UTM Skudai, Johor Darul Takzim, Malaysia
+60137771180
themazor@yahoo.com

Abdullah Bade
Department of Computer Graphics and Multimedia,
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia
81310 UTM Skudai, Johor Darul Takzim, Malaysia
+60197596038
abade@utm.my

Daut Daman
Department of Computer Graphics and Multimedia,
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia
81310 UTM Skudai, Johor Darul Takzim, Malaysia
+60137702975
daut@utm.my

Norhaida Mohd Suaib
Department of Computer Graphics and Multimedia,
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia
81310 UTM Skudai, Johor Darul Takzim, Malaysia
+607-5560044
haida@utm.my

ABSTRACT
Performing fast and accurate collision detection between geometric models is still most open problems in modeling, robotics, manufacturing and computer-simulated environments. Most of previous method seems to have their own specification and restricted to a specific geometric models. For example, convex polytopes tend to solve accuracy problems but left behind the speed of collision detection. In this paper, we present a Bounding-Volume Hierarchies (BVH) technique for collision detection between general polygonal models in urban simulation. By using hierarchical approach, we believe that collision detection between static and dynamic object can be preferred in real-time and suitable to overcome the equipment of urban simulation. Our preliminary result shows that bounding-volume hierarchies achieve favourable frame-rates in real times simulation using binary tree. In practice, we can construct bounding-volume hierarchies for accurate and fast collision detection method in urban simulation.

Categories and Subject Descriptors

General Terms

Keywords
Collision detection, Bounding-volume hierarchies, Urban simulation.

1. INTRODUCTION
Computer graphics (CG) technology has evolve quite captivatingly especially for three-dimensional (3D) games, animation and simulation. Since the birth of 3D technology, enormous techniques and tools have been developed to design more powerful and realistic 3D applications. Technology keep evolves and almost everything that airing and seeing in television and computer games for example, related to the 3D CG. Films such as Transformers, Dragon Wars, and even latest Indiana Jones film for instance, have embedded with superior 3D CG animation that bring 3D CG into a new way of entertainment instead of music and theatre. That is why researchers throughout the world are paying full attention to explore delving researches and contribute significant achievement just to bring worthy and advance effects to the audience in term of realism and futuristic. However, there are some criteria and important issues need to be considered when designing a 3D application. For instance, in 3D games, animation and simulation, there are needs to maintain the application runs with suitable frame rates such as 30-60 fps [7, 22, and 28]. To achieve that goal, the applications should put high attention during construction phase through applying some appropriate techniques with some optimization. Instead of that, there is a need to balance between the realism and the complexity of the applications [12, 31].

In urban simulation, simulating virtual models such as buildings, trees, roads that consist of thousand polygons must be rendered correctly. For this high consumption memory environment, every issue and aspect for rendering for example collision detection, textures, culling, and others must be taken into account [31]. However, implementing all those things is not necessarily critical as it depends on what the purposes of the application. On one hand, lecturers may want to use it as learning tools for teaching civil student. On the other hand; urban planner used it to give their client better perspective of the proposed architectural and city design. Hence, it is quite useful for researchers to broaden the previous researches and the limitation of current technologies when designing urban simulation.

Countless numbers of researches have been undergoing regarding collision detection algorithms in virtual environments. For urban simulation, it always utilizes very
large number of polygons in order to produce realistic environment setting. The need to maintain real-time frame rates during running time for urban simulation is very demanding. Looking into collision detection perspective, detecting interference between objects in urban simulation is important and not an easy task. Any proposed techniques should be efficient and robust to handle objects interference during running time. Bounding-volume hierarchies’ technique has proven to be effective when detecting object interference in urban simulation. Our contribution in this paper is to introduce bounding-volume hierarchies for detection object interference in urban simulation. Our result shows that by using bounding-volume hierarchies, it provides more accurate and fast collision detection method.

2. RELATED WORK

In this section, we will briefly review related work on collision detection and urban simulation. The primary type of collision detection algorithms deals with static rigid bodies in static positions. But many recent studies are concentrated at more difficult situations such as deformable models rather that rigid models. The problems of collision detection in virtual environment have been extensively studied and recent surveys are available in Kockara et al [1] and Lin and Manocha [2]. However, we will limit our discussion to collision detection between rigid bodies in large environment, such as one in urban simulation.

2.1 Hierarchical Approaches

Spatial partitioning or bounding-volume hierarchies are common method for detecting object interference in large-scaled simulation. Both have shown remarkable result when applying them in virtual environment as they create faster and precise collision detection. Common bounding-volume for instance axis-aligned bounding boxes (AABBs) [10, 11], spheres [3, 4, 5, 6, 7, 14], oriented bounding boxes (OBBs) [13], k-DOPs [8, 15], ellipsoid [9] and oriented convex polyhedra [26] have been used to perform collision detection in virtual environment. In order to lower the computational cost of updating the hierarchy trees, it is necessary to use simple bounding-volume such as spheres or AABBs for fast computations.

2.2 Urban Simulation Collision Detection

Collision detection can be distinguished through their design method (how they employ and based). Earlier studies by Cohen et al in 1994 had used exact collision detection for detecting collision between objects in interactive environments [12]. Musse et al. in 1997 developed two methods to detect collision between human crowds [16]. The first method can be achieved by using simple mathematical equations while the second one is using earlier prediction of collision and then change the path through a simple geometric computing. Other researchers such as Loscos et al [17] and Feurtey [18] studied collision detection to determine path motion for human crowd in urban simulation.

Aliaga et al in 1998 had used bounding-volume hierarchy for detecting object interference in massive models [19]. Farenc et al (2002) had developed an Informed Environment city that provided details city information of recognition places and using bounding box for collision detection [20]. Tecchia et al in 2002 used a simple and fast method to detect collision in complex city cramps with human and other 3D models [21]. They used two different approaches to achieve real time collision detection. On one hand, it involves predicting the future or final position of the object. On the other hand, they reallocate the task for detecting collision further from current object position.

Later in 2003, Hamill et al [22] had presented Virtual Dublin urban simulation that can be used as a prototype to test various projects such as crowd and traffic simulation, land design, and disseminated graphics studies. They implemented a Moller Ray-Triangle Intersection Algorithm for detecting collision in urban simulation [23]. Nurminen et al. had suggested a 3D mobile map, called m-LOMA city [24]. The city was designed with special mobile features that can keep the location-based information of the city. In m-LOMA, collision detection had been used to detect intersection between user moving line and the objects of the city.

2.3 Project Background

This research intends to propose a framework of hierarchical bounding-volume technique that can be used in urban simulation for a faster intersection test. Using UTM Walkthrough in Figure 1 with simple simulation consisting of a few buildings in virtual environment, we believe that our hierarchical bounding-volume technique can create fast and accurate collision detection in urban simulation.

Figure 1: Universiti Teknologi Malaysia (UTM) simulation

The simulation of UTM is a project that has been developed by GMM research group using OpenGL since 2002 and will become our platform that representing urban simulation to test our propose framework of collision detection. Below are the steps for creating urban simulation (Figure 2).
we will perform a collision detection test to our proposed approach using collision detection module.

The collision detection modules that will be detect any collision pairs and proceed to collision response. Each detected collision is then responded to in sequential order, and effects are passed back to the application which is our urban simulation.

Figure 3. Collision detection framework

3. BOUNDING VOLUME HIERARCHIES

This section will describe the overview of hierarchical method that we propose into our urban simulation. Bounding-volume hierarchies (BVH) have proved to be most efficient method for collision detection [11, 13]. For this research, we will use hierarchical cost function that been used by previous researchers. Basic cost function was first formulated by [25] for hierarchical method in ray tracing and later been used by [13]. Then been refined by [15]. The calculation of execution time is formulated as follows:

\[ T = N_v X C_v + N_p X C_p + N_u X C_u + C_o \]  

(1)

Where

- \( T \): total execution time for detecting interference
- \( N_v \): number of bounding-volume 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
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 $N_c$ 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 $C_v$), it is resulting an increase of the intersection tests between bounding-volumes. Minimizing one value will causes an increase to another value. This is the challenge in all collision detection to find which one is the most important.

### 3.1 BVH building strategies

Bounding Volume Hierarchy (BVH) is a tree structure that represents geometric models with specific bounding volume. It works like a tree that has a root (upper division), a group of leaves (middle division) and a leaf (last division). The main idea of the bounding volume hierarchies is to build a tree that has a primary and secondary root which each of the secondary nodes stores as a leaf (Figure 3).

![Binary Tree construction](image)

**Figure 3. Binary Tree construction [30]**

There are also three types how we can construct a tree: top down, bottom up, and 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 become root node. Insertion methods can be implemented by inserting one object at one time into the tree. Each type has its own unique characteristic.

### 4. BUILDING AABB HIERARCHIES

Traditionally, a naïve approach to check for any collision is by checking each triangle for intersection with another triangle for each object. However, by checking all the polygons for intersection is quite expensive even in the simulation that has very few objects because collision detection must always checking for intersection, whether it is collided or not. That is why Hubbard et al [4], came out with a concept called broad phase and narrow phase. The main idea of broad phase collision detection is to reduce the computational load by performing only necessary test and prune an unnecessary pair test. Narrow phase intends to perform actual pair test after broad phase confirmed that there is collision. Hence, by using hierarchical method along with bounding-volume, we can localize the areas where the actual collision will be occurred.

AABB provides fast and almost accurate collision detection method comparing to tighter bounding-volumes such as convex hull, k-DOP, Oriented-DOPs, and few others. By doing comparison between those bounding-volumes and AABB bounding-volume, AABB performance is quite good as their computation is inexpensive and makes it relatively faster than any of those bounding-volumes. Interestingly, AABB tree can be used to generate an approximate response. As we never perform collision detection between primitive, AABBs can be used to approximate the collisions. Although spheres are the most preferable bounding-volumes as they are faster than AABB, we chose AABB based on their accuracy is more relevant in order to perform fast and accurate collision detection method. Another important reason is the fact that AABB used only six variables which are min and max of three axes and also can be easily implemented. In our case, we compute AABB bounding-volume using brute force computation where we find min and max value by checking all coordination of object in local coordinate space. Then, once we found the suitable min and max value, we used them to create our AABB by using OpenGL function called GL QUADS in order to combine all min max value into box.

However, the most disadvantages thing is AABB need to be realigned each time the object is rotated. Thus, updating the AABB for object in rotation is quite expensive and the best solution for this problem is by computed a looser box so that the AABB can fit properly even though the object is rotated [30].

Updating bounding-volume hierarchies becomes more complicated when object undergoing rotation or deformation. Hence, any bounding-volume trees need to be recalculating back again even after a small movement to ensure their effectiveness of detecting object interference. The high cost of computing newly bounding-volume data structures makes researchers to use only simple bounding volume such as spheres and AABB. Below is the excerpt of C++ codes from our tree construction to update AABB each time the object undergoing rotation.

```cpp
void transform(float m[16], vectorf *v)
{
    vectorf tmpv;
    tmpv.x = v->x;
    tmpv.y = v->y;
    tmpv.z = v->z;
    //homogenous transformation
    v->x = m[0]*tmpv.x + m[4]*tmpv.y + m[8]*tmpv.z + m[12]*1;
    v->y = m[1]*tmpv.x + m[5]*tmpv.y + m[9]*tmpv.z + m[13]*1;
    v->z = m[2]*tmpv.x + m[6]*tmpv.y + m[10]*tmpv.z + m[14]*1;
}
```

**Figure 4. Transformation update of AABB**

There are many algorithms that can be used to construct AABB-trees. According to Hubbard et al [4], any algorithms that construct bounding-volume hierarchies must meet the following basic requirements:-

i. It must be an automatic construction without any user intervention.

ii. Hierarchy must be constructed in efficient way in order to make searching for nodes easier, with
each level eliminating the need to search a significant subset of the next level.

iii. It should fit the object as tightly as possible (depends on algorithm or type of bounding-volume)

4.1 Building the Binary Tree
In order to build binary tree, we first bound the object with large AABB which is become root of our tree. Then, by choosing a splitting point for the AABB tree, the root tree that covered with AABB is cut into two tight fitted AABB which is either one of them become left tree and the other one become assigned to the right tree. The process of assignment into two sub tree is shown in figure 5 based on our tree construction method. First we assigned a new pointer and declare both right and left as a NULL. Then we tested for its flag, whether we really have an empty tree or not. But if we have a tree, then we assigned the AABB to left and right. The algorithm computes the separating axis of parent level in order to become child nodes. The AABB trees can be seen in Figure 7. For this tree construction algorithm, we used top-down approach as it is the most preferable approach by most researchers.

```c
insertAABBTree(tree,minP,maxP,loc)
{
    //Empty Tree
    if(tree == NULL){
        //Declare new tree node
        tree=new treeNode;
        tree->right=NULL;
        tree->left=NULL;
        //Assign minPobj and maxPobj variables with minP and max
        tree->minPobj=&minP;
        tree->maxPobj=&maxP;
        //testing for flag
        tree->AABBnode.flagInt=loc;
        //Create a AABB BV,given minP and maxP value for each node
        AABBInitVertices(&tree->AABBnode,minP,maxP);
    }

    //If minP is bigger than tree minPobj and maxP is lower than tree maxPobj
    //Insert this object to the left
    else if(((minP.x > tree->minPobj->x)&&(maxP.x < tree->maxPobj->x))
    &&((minP.y > tree->minPobj->y)&&(maxP.y < tree->maxPobj->y))
    &&((minP.z > tree->minPobj->z)&&(maxP.z < tree->maxPobj->z))
    {
        insertAABBTree(tree->left,minP,maxP,loc);
    }
    else
    {
        //Insert aabb to the right
        insertAABBTree(tree->right,minP,maxP,loc);
    }
}
```

Figure 5. AABB sub tree assignment to left and right tree

5. MAIN ALGORITHM
The general idea behind this main algorithm is to take advantage of the AABB tree hierarchy to computer the collision between two objects using their own trees and find any branches that can be pruned. In this way, collision might occur at the certain place in the object hierarchy and collision checking can be reduced to the specific section of each tree. Therefore, further testing will be conducted at the area that probably has potential to collide and thus reducing time consumption to check all polygons for intersection.

5.1 AABB-Tree Intersection Algorithm
In our implementation, we test intersection between two objects that completed bounded with AABB trees. We pointed it as TreeA and TreeB, which approximate two objects whose bounding boxes overlap in all three dimensions. Then, we created a pointer to the roots of both trees that can be assessed using function BVOverlap. Figure 6 depicts the intersection algorithm for our tree.

```c
BVOverlap(m1[16], TreeA, obj1, m2[16],TreeB, obj2)
{
    //When there is a movement, the corresponding AABB
    //will be updated
    //Find the new vertex after some movement
    UpdateAABB(TreeA, m1, obj1);
    UpdateAABB(TreeB, m2, obj2);

    //Possible Colliding Pairs
    TreeA minPoint = TreeB AABB min value
    TreeB maxPoint = TreeA AABB max value
    TreeA maxPoint = TreeB AABB max value
    TreeB minPoint = TreeA AABB min value

    //Testing the collision between two objects distance.
    //Make sure all axes is overlap in order to confirm there is collision between them.
    if( (TreeB AABB min value < TreeA AABB max value) && (TreeB AABB max value && TreeA AABB min value) )
    {
        return true;//collide
    }
    else return false;//no collision
}
```

Figure 6. Algorithm to verify if two objects are in collision

Figure 7. Object covered with one AABB is separated into two smaller AABB

Currently, we show how the intersection at each level occur by performing direct collision detection between each level AABB by selecting any level of our tree hierarchies. For example when we select level 4 binary trees, our system will report back to us that intersection has occurred.
### 6. EXPERIMENTAL RESULTS

We have implemented a collision detection algorithm based on the binary AABB-tree structure. The code was well written in C++ language, using OpenGL library and runs on most Windows XP and Windows Vista operating systems and PC Intel Core 2 Duo, the CPU was a 2.2 Ghz with 2 GB of RAM with frequency of 677 Mhz. Our algorithm uses recursive method on the tree structure to solve the problem.

In the following, we analyze the performance of BVH for 8 level binary trees by computing it processing time. There are four objects involved which are Bunny, KDop, Teapot and Venus which taken from Stanford library for our research. At first we load only single object and then we load two same objects to compare its loading time for BVH construction. Notice that the AABB that we used has transformation update when object is in rotation thus it is quite challenging to bring 8th level of our AABB binary trees into favorable frame-rates. Figure 8 shows the construction times for those objects in 8 level of BVH.

#### 6.1 Evaluation

Independent experiments have been conducted on several geometric models. In this paper, we highlight the BVH construction times for four geometric models from Stanford Library which consist hundreds of triangles. Each of these models can move in any direction and rotation. In urban simulation, this object can be replaced with multiple buildings that has it owns AABB trees. Then, by using a camera that completely bounded with AABB trees, we will conduct some test to evaluate our BVH construction and traversal algorithm. In this experiment, we just conducted simple collision between two same objects and their tree construction times. Later all those objects will be replaced with buildings and urban models to conduct real testing in urban simulation environment. UTM walkthrough will be further developing into real urban environment. Figure 9 shows some AABB trees construction at certain level or depth.

### 7. CONCLUSIONS AND FUTURE WORK

We have shown our proposed method that will be useful for faster traversal between nodes when detecting object interference in urban simulation. Currently we are in the phase of implementing bounding-volume hierarchies' technique in urban simulation. Sample urban simulation will be built with various buildings and entities to make urban simulation more realistic. Bounding-volume hierarchies with suitable bounding-volume enclosed each object in urban environment can increase the efficiency and speed of object intersection. According to Somchaipeng et. al.[27], heterogeneous bounding-volume hierarchies is still in limited study thus we believe that it can provides good solution for us to speed up the intersection test using multiple bounding-volume although it may increases the computational cost.

The work is still ongoing status with further implementation and testing of our propose method. Bounding-volume hierarchies' algorithm using top-down approach will be modified to optimize the tree construction along with tree traversal update. Furthermore, the construction of BVH must be automatically without user intervention and without manual construction. Next, we will also do some comparison of its performance between other bounding-volume hierarchies' algorithms. Finally, the BVH will be tested in urban simulation environment where many object must be involved in order to perform collision detection in urban simulation. Linking the gap between fundamental theory of bounding-volume hierarchies and coming up with good algorithms is still an open challenge for researchers and thus creates good opportunities for further research project.

### 8. ACKNOWLEDGEMENT

We would like express our appreciation to Malaysian Ministry of Science, Technology and Innovation under eScienceFund grant (VOT No. 79237) for financial support of this research.
9. REFERENCES


