1.0 INTRODUCTION

Virtual or simulated environments exist in a variety of application, such as CAD tools, computer game, virtual reality application and physics simulators. For example, a physical simulation that needs to simulate a bowling game is discussed. The physics of pins are well established under the rigid body motions and elastic collisions. Given are the physical initial locations and characteristics of the bowling lane, pins and the bowling ball. After a swing of the bowling ball towards the pins, the application should be able to calculate the trajectories, precise motion and final resting positions of all involved entities. This application can be decomposed to a few subsystems with their own functions. One subsystem can deal with the precise motion, and another can deal with the intensity of sound generated. However, there must always be the inclusion of a subsystem that determines which pins the bowling ball strikes. This seemingly simple function is often termed “collision detection” or “intersection testing”.

Compared to this physical simulation application, a computer or video game also employs the same collision detection routine but it needs not simulate the real-world physics as accurate as in the bowling application. Computer or video game simulates the real world in an acceptable way, in real time and robustly.

Collision detection or intersection testing maintains and heightens realism in virtual or simulation applications. In another simple example, a player in a computer game will not fall through the floor or walk through walls due to the technique of collision detection.

1.1 Problem Statement

However, any single object in simulated environment may often be composed of tens or hundreds of polygon combination. In such environment, there may be hundreds or thousands of these complex objects existing at one time. It is simply not feasible or in other words, entirely impossible to perform collision detection on every single objects.

This scenario, if exist, requires an unbelievably large amount of hardware performance and memory requirement. The cost of this method of collision detection for every single object and complex in nature is too great and may not justify the current technological advances.

1.2 Motivation

The idea of Bounding Volume (BV) has been introduced to encapsulate these complex objects in simpler volumes. These simpler volumes provide a more memory-efficient abstraction of the objects they are bounding. Faster and earlier rejection test can be performed to eliminate
object pairs that are far apart with lower probability of collision. Detailed intersection test are performed only on the complex objects where their corresponding BVs collide.

This project aims to bound 3D objects in BV and is useful for anyone who wishes to test their own collision detection algorithms without bothering about representing objects with simpler volumes. The BV chosen for this project is Oriented Bounding Box (OBB). Users should be able to import several 3D objects and see the resulting BV computed for each objects.

1.3 Objectives

The points below will be the main objectives for this tool that have been identified throughout the development of this project:

- Flexible in importing several different 3D objects at one time
- Draw the actual 3D objects and its corresponding OBBs
- Able to move and also orient the 3D objects
- Test and highlight the intersection between OBBs of different 3D objects
- Export the output of the visual frontend as text-based data to a text file

Several secondary objectives have been identified throughout the development of this project:

- Lower processing resources requirement
- More memory efficient (minimize the complexity of operation involved in collision detection)
- Compromise between performance and accuracy

1.4 Project Scope

This tool is designed for users who wish to generate OBB from their intended 3D objects. These users are generally involved in creating or implementing collision detection algorithms.

1.5 Project Limitations

This tool does not implement a complete collision detection algorithm. If two 3D objects are imported into the tool while executing, they will only be checked for intersection only when user manually moves these objects toward each other. Realism in a typical collision detection scenario (e.g. objects bouncing off each other) will be at a minimum level.
### 1.6 Project Schedule

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gather Project Requirements</td>
<td>29/12/2008</td>
<td>9/1/2009</td>
<td>2w</td>
</tr>
<tr>
<td>6</td>
<td>System Development</td>
<td>1/6/2009</td>
<td>7/9/2009</td>
<td>14w 1d</td>
</tr>
<tr>
<td>7</td>
<td>System Implementation</td>
<td>7/9/2009</td>
<td>21/9/2009</td>
<td>2w 1d</td>
</tr>
<tr>
<td>8</td>
<td>System Testing</td>
<td>21/9/2009</td>
<td>2/10/2009</td>
<td>2w</td>
</tr>
<tr>
<td>9</td>
<td>Documentation</td>
<td>2/10/2009</td>
<td>16/10/2009</td>
<td>40w</td>
</tr>
</tbody>
</table>