Chapter 10
Collision Detection in Video Games

Benjamin Rodrigue
University of Louisiana at Lafayette, United States of America

ABSTRACT

This chapter will describe several methods of detecting collision events within a 3D environment. It will also discuss some of the bounding volumes, and their intersection tests that can be used to contain the graphical representation of objects in a video game. The first part of the chapter will cover the use of Axially Aligned Bounding Boxes (AABBs) and Radial Collision Volumes. The use of hierarchies with bounding volumes will be discussed. The next section of the chapter will focus on Object Oriented Bounding Boxes (OOBs). The third section is concerned with the Gilbert-Johnson-Keerthi distance algorithm (GJK). The last three sections will focus on ways of optimizing the collision detection process by culling unnecessary intersection tests through the use of type lists, sorted lists and spatial partitioning.

INTRODUCTION

Collision detection is a difficult aspect of a game engine's architecture. Knowing when two or more objects are colliding is crucial to ensure that a proper response occurs within the game's space and mechanics. The response can be a loss of health of a player character or the toppling of a lamp off of a table. The response that helps immerse the player in the game's world first starts with the detection of the collision. This chapter will cover some of the methods in use by the game industry for detecting a collision event within a game. This chapter will not discuss how to respond to a collision event as that will vary depending on the game at hand just as the choice in collision detection will vary depending on the needs of the game.

A collision detection system should be designed early in a game's development. It should not be

DOI: 10.4018/978-1-4666-1634-9.ch010

put off to be added at a later time; this can result in inefficient changes to the engine architecture, and worst case scenario, become an execution bottleneck. There is no generic algorithm that works best for all collision detection scenarios because games have different demands for detection. A game with relatively few collidable objects will not need the same detection system as a game with thousands of objects.

All bounding volumes described in this chapter are pre-computed before execution time. Predictive collision checks are assumed in the sense that an object's next position should be pre-calculated one loop before it is applied. The collision detection system will then be doing checks one frame ahead of where the object will be in order to stop an object from entering into a collision. The alternative method of moving the object out of the collision can result in more complicated and inefficient situations. Moving an object outside of a collision could result in it moving into a new collision with another object.

All pseudo code examples within this chapter are based off of C++.

BACKGROUND

This chapter is organized into three parts. The first section is a math primer on vectors and matrices. The math primer is not meant to be comprehensive, but it will present a couple of the rules for matrices and vectors. The second topic will focus on collision detection volumes that improve detection performance over pixel by pixel overlapping tests. The collision volumes are Axially Aligned Bounding Boxes (AABB), Radial Volumes, Binary Volume Hierarchies (BVH), Object Oriented Boxes (OOB) and the Gilbert-Johnson-Keerthi distance algorithm (GJK). These bounding volumes are arranged in order of increasing complexity and precision.

The last topic will describe ways to reduce the number of necessary collision tests. This part is split up into three sections. These sections describe the ways of reducing the number of detection tests based on object type and object location. The first part of this section describes how to use dynamic and static object lists. The next section is concerned with the sort and sweep algorithm. The last section describes spatial partitioning using Quadtrees and Octrees.

Math Primer

Vectors

This chapter will make use of two mathematical structures: vectors and matrices. A vector is an ordering of two or more real numbers in a set. Throughout this chapter, two and three dimensional vectors will be used to explain the different ways to check for collisions. In the pseudo code examples throughout this chapter, Vector3 will indicate a three dimensional vector while Vector2 will indicate a two dimensional vector. This data structure is useful in collision detection because it can be used to represent a position, direction, and velocity among other things in two or three dimensions. The vector, **v** is a positional vector in two dimensional space. The normalized form of **v,** represents the direction from the origin through **v** and is represented with two vertical bars, ||**v**||. The normal of **v** is calculated using the magnitude of **v**, |**v**|.

```
v = (2, 2)
//position vector
```

The magnitude is calculated by taking the square root of the addition of each element squared. An example of calculating the magnitude:

```
|v| = SquareRoot(v[0]^2 + v[1]^2)
//magnitude
|v| = 2.8284
```