Convexity-Based Collision Detection Algorithms for a 2D Platform Game

COMP110 - Computer Architecture Essay

1507866

December 6, 2015

This essay will analyse three convexity based collision detection algorithms focusing on both their accuracy and run times. Then a recommendation will be made based on which algorithm balances the two in the best way for use in a 2D platform game.

1 Introduction

The purpose of collision detection is to check if objects are currently or going to intersect. In the application of video games this is vital for ensuring that objects interact correctly so the game can be played. This essay will analyse three convexity-based collision detection algorithms for use in a 2D platform game. The first algorithm is the Gilbert-Johnson-Keerth algorithm first proposed in 1988. Guo proposed the second algorithm in 2010 and the third is by Sulaiman et al in 2013. One algorithm will be recommended based on its balance between accuracy and speed.

2 The three algorithms

2.1 Algorithm One: Gilbert-Johnson-Keerth (GJK) algorithm

Gilbert et al proposed their collision detection algorithm, the Gilbert-Johnson-Keerth (GJK) algorithm in 1988 [1]. The algorithm is designed to work efficiently for 3D collision detection however it also works for 2D. Therefore it could be implemented in a 2D platform game. The intention of this algorithm is to determine whether objects intersect. This is done by calculating the Eucildean distance between a set of convex objects. According to Gilbert et al the distance between two convex objects is the same as the distance between those object's Minkowski difference and the origin. If that difference is zero then the objects are intersecting each other or have collided.

The original paper gives run times for the algorithm on a selection of 3D shapes. The minimum time being 0.13 seconds and the maximum being 19.81. However these are being run in 3D so it is likely it would take less time on 2D objects. Furthermore this paper was published in 1988, so the results are restricted by the technology of that time. The results from the paper are from tests run on a Harris 800 which was a computer available from 1979 [2]. If the tests were repeated on modern hardware it would most likely run considerably faster.

GJK is a very accurate algorithm due to its use of convex hulls. However the cost of this accuracy is speed. In a 2D platform game speed is vital, a few seconds lag for each collision would make the game unplayable. While accuracy is important it should be balanced with speed for games.

An issue with the algorithm could also be its age. There are a number of papers that recommend and detail improvements that could be made to GJK to make it faster or more efficient. Gilbert later detailed improvements for GJK in another paper in 1990 [3]. Lindemann says that there is a sub-algorithm in GJK that is ”somewhat outdated” but that particular algorithm is often replaced or adapted when used to fit the given situation [4, p. 1].

2.2 Algorithm Two: Guo's algorithm for 2D grapple games

The second algorithm is designed for collision detection in 2D grapple games. This paper was published in 2010 making it more recent than the GJK algorithm however it is limited to 2D. In contrast GJK can work in m-dimensions, the paper detailed its use for three dimensions but it is also viable in 2D.

Guo proposes a method which bounds the game objects in axis aligned rectangles and circles as can be seen in figure 1. A collision has occurred if there is an overlap region between the shape's detection area and a collision object. This gives a Boolean that states that either a collision is happening or not [5].

The evidence shows that this is a fast algorithm compared to GJK. The fastest results were for the collisions of two circles. The minimum and maximum time for that collision was 6.20531 milliseconds. The longest this algorithm took was 160.105 milliseconds for the collision of two rectangles. In contrast the GJK algorithm took a maximum of 19.91 seconds. The results suggest it is faster than GJK but the results for GJK were from 3D testing.

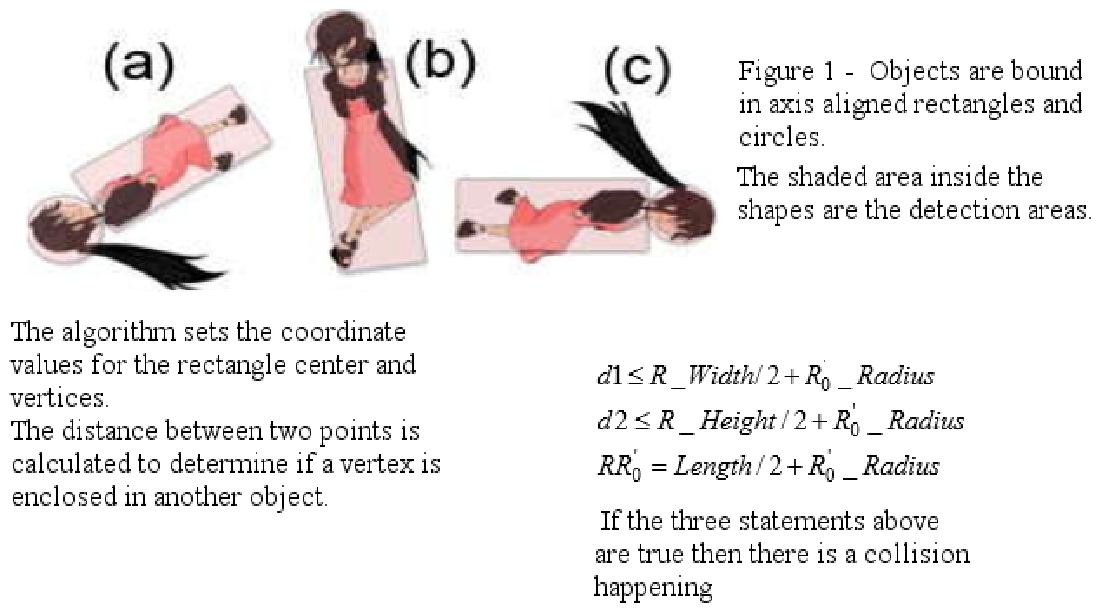

Figure 1 - Objects are bound in axis aligned rectangles and circles.
The shaded area inside the shapes are the detection areas.

The algorithm sets the coordinate values for the rectangle center and vertices.
The distance between two points is calculated to determine if a vertex is enclosed in another object.

$$d1 \leq R\_Width/2 + R_0^{'}\_Radius$$
$$d2 \leq R\_Height/2 + R_0^{'}\_Radius$$
$$RR_0^{'} = Length/2 + R_0^{'}\_Radius$$

If the three statements above are true then there is a collision happening

Figure 1: Guo's algorithm and use of bounding boxes [5].

An issue with Guo's algorithm could be that it is specific to 2D grapple games. It may be unsuitable for use in other games or require some adaptation. In contrast GJK is more open to general use as it can be used in 2D or 3D and is not just limited to use in games.

A further issue is the use of bounding boxes. Sulaiman et al [6] suggested that the use of bounding boxes in collision detection can be inaccurate. This is because the bounding boxes may intersect and report that a collision is happening. However in actual fact the objects themselves are not colliding making the results inaccurate.

2.3 Algorithm Three: Collision Detection using a dynamic origin point

The third technique has been proposed by Sulaiman et al [6]. The algorithm is for distance computation in the narrow phase of two phase collision detection. The basis of this algorithm was to find a solution that was more accurate than bounding boxes. Their method involves using a dynamic origin point, or DyOp. Figure 2 shows the first stages of DyOp.

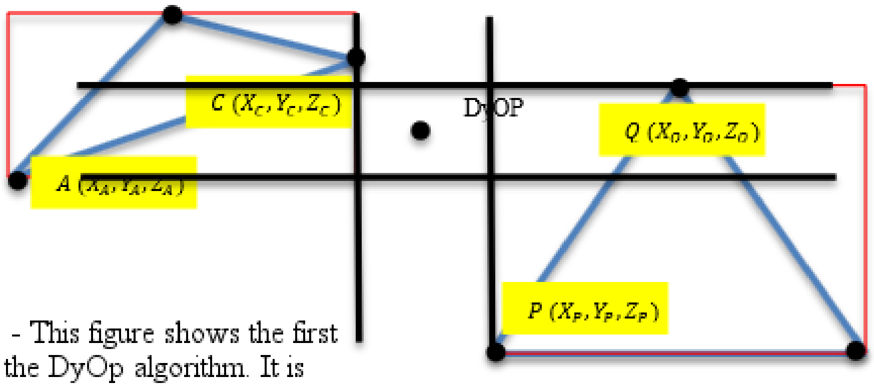

Figure 2 - This figure shows the first stage of the DyOp algorithm. It is checking to see which vertex or side of the shape is cloest to the Dynamic Origin Point. In the digram above the sides AC and QP are closest.

$$DiVectA\_P = Vertex_A - Vertex_P \qquad eq\ (10)$$

$$DiVectA\_Q = Vertex_A - Vertex_Q \qquad eq\ (11)$$

$$DiVectC\_P = Vertex_C - Vertex_P \qquad eq\ (12)$$

$$DiVectC\_Q = Vertex_C - Vertex_Q \qquad eq\ (13)$$

Then direction vectors are calucated between the vertices. The cross product is then calculated giving a distance.

Figure 2: DyOp being used on two triangles [6]

Sulaiman et al do not give run times for their algorithm. They instead compared it to two existing algorithms. One of which was the GJK algorithm. They found that their algorithm gave a speed increase between 78.1% to 110.9%. The DyOp algorithm does not provide data on how fast it runs in milliseconds making it hard to compare to the second algorithm. However Guo's algorithm used bounding boxes. Sulaiman et al were aiming to implement an algorithm more accurate than bounding boxes.

Sulaiman et al's method is more accurate than using bounding boxes and is significantly faster than the GJK algorithm therefore based on the evidence given Sulaiman et al's DyOp algorithm is most suited for use in the 2D platform game. The lack of time data means it is not possible to tell whether it is faster than Guo's or not. However Guo uses bounding boxes which while being fast leads to inaccuracy due to the way objects are bound.

3 Conclusion

In conclusion the DyOp algorithm proposed by Sulaiman et al appears to be the best choice for the 2D platform game. DyOp performed significantly faster in tests than GJK and is more accurate than Guo's bounding box based algorithm. This balance between accuracy and speed makes it the ideal choice for use in a 2D platform game.

References

[1] J. D. W. Gilbert, Elmer. G and S. S. Keerth, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193 – 203, 1988.

[2] Harris Corporation, “Reference manual harris 800 general purpose digital computer,” 1979.

[3] E. Gilbert and C. Foo, “Computing the distance between general convex objects in three-dimensional space,” *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 53 – 61, 1990.

[4] P. Lindemann, “The gilbert-johnson-keerthi distance algorithm,” *Algorithms in Media Informatics*, 2009.

[5] K. Guo and J. Xia, “An improved algorithm of collision detection in 2d grapple games,” in *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pp. 328–331, IEEE, 2010.

[6] H. A. Sulaiman, A. Bade, and M. H. Abdullah, “Distance computation using axis aligned bounding box (aabb) parallel distribution of dynamic origin point,” in *Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy (AICERA/ICMiCR), 2013 Annual International Conference on*, pp. 1–6, 2013.