Mathematical Approaches for Collision Detection in Fundamental Game Objects

Weihu Hong\textsuperscript{1}, Junfeng Qu\textsuperscript{2}, Mingshen Wu\textsuperscript{3}

\textsuperscript{1} Department of Mathematics, Clayton State University, Morrow, GA, 30260
\textsuperscript{2} Department of Information Technology, Clayton State University, Morrow, GA, 30260
\textsuperscript{3} Department of Mathematics, Statistics, and Computer Science, University of Wisconsin-Stout, Menomonie, WI 54751

Abstract – This paper presents mathematical solutions for computing whether or not fundamental objects in game development collide with each other. In game development, detection of collision of two or more objects is often brought up. By categorizing most fundamental boundaries in game object, this paper will provide some mathematical fundamental methods for detection of collisions between objects identified. The approached methods provide more precise and efficient solutions to detect collisions between most game objects with mathematical formula proposed.

Keywords: Collision detection, algorithm, sprite, game object, game development.

1 Introduction

The goal of collision detection is to automatically report a geometric contact when it is about to occur or has actually occurred. It is very common in game development that objects in the game science controlled by game player might collide each other. Collision detection is an essential component in video game implementation because it delivers events in the game world and drives game moving through game paths designed.

In most game developing environment, game developers relies on written APIs to detect collisions in the game, for example, XNA Game Studio from Microsoft, Cocoa from Apple, and some other software packages developed by other parties. Most open source or proprietary game engines supports collision detection, such Unreal, C4, Havok, Unity etc. However, a primary limitation of game development kits or game engines is the precision is too low, and the collision detection approaches are limited in the package.

Since the advent of computer games, programmers have continually devised ways to simulate the world more precisely. It’s very often that programmers need to develop their own collision detection algorithms for a higher precision and performance.

The most basic approach is collision detection of the sprite or boundary class which represents an object in the game scenes and is often rectangle, sphere, or cube. This approach works well if the object that is represented by is a simple shape and there is almost no blank space between object and the image. Otherwise, a lot of false alarm will be introduced in collision detection as show in Figure 1, where two objects, one circle and one pentagon, are not collided at all even the represented sprites collide each other.

Figure 1. Collision detection based on Boundary

Figure 2. Per-Pixel based collision detection

Per-pixel collision detection is a relic from the past. It uses bit-masks to determine whether two objects collide. The biggest advantages of using this detection method are that it also checks the empty space within the objects boundaries, and collision is detected with high precision as pixel-perfect and fair, there are no false positives nor false negatives as shown in Figure 2, where (a) shows the improved per-pixel detection, and (b) shows correct collision happened when two object contacting each other. The main disadvantage is that it’s expensive to compute, and is extremely slow compared to bounding boxes and it won't work if you do any transformations on the target objects such as rotating, or resizing. For example, if viewable sprites are 32x32 pixels. In order to check collision, the program needs to check 32X32x32x32 pixels with each pass to find out if a single pixel of a 32x32 frame of the sprite sheet has collided with
another image. Therefore per-pixel collision checks are rarely used, especially because object visible shape and collision shape are usually different.

In computational geometry, there is a point-in-polygon (PIP) problem that asks whether a given point in the plane lies inside, outside, or on the boundary of a polygon. It is a special case of point location problems and finds applications in areas that deal with processing geometrical data, such as computer graphics, computer vision, geographical information systems (GIS), motion planning, and CAD. The Ray Casting method is one simple way of finding whether the point is inside or outside a simple polygon by testing how many times a ray, starting from the point and going any fixed direction, intersects the edges of the polygon. If the point in question is not on the boundary of the polygon, the number of intersections is an even number; if the point is outside, and it is odd if inside (see [1] to [12]).

However, the detection of collision problem is different from the PIP problem. We are not interested in whether a point inside or outside of a polygon. We are just interested in a collision occurs or not. Therefore, it is not good idea to use the Ray Casting method in our collision detection problem.

There are a lot of work have been done to improve collision detection For example to solve the performance issues with per-pixel algorithm, space partition algorithms has been used, such as Quadtree or Octree. Reference [13] to [23] has listed a spectrum of researches in collision detection.

In this paper, we proposed mathematic solutions for collision of each fundamental game object shape identified. The approach is not based on the bounding of the sprite, but the collision detection is based on the bounding shape which represents the game object with minimum empty space between the object and bounding shape. The approach works as follows. For each object that in the game scene, define a fundamental geometric shape or combination of fundamental geometric shaped that encloses its texture. The fundamental shapes that are identified in the research including: point, segment, squares, rectangles, triangles or circles. The following figure shows some example of bounding shapes for some sprites for example.

![Figure 3. Bounding Shape of Circle, Triangle, and Rectangle in Game.](image)

By defining the bounding shape for a game object, once we have mathematical solution for collision detection of bounding shape, it should be easily accommodate sizing and rotating, it also provides better accuracy of collision detection without performance sacrificing of check each pixel. We believe that our method for detection of collision between two triangles is easier to implement than some of existing methods. To represent a complicated visual object, a different combination of fundamental bounding shapes can be used together.

2 Collision detection of bounding shapes

Here upon, we will refer our defined bounding shape of a visible object as an object. Let us consider collision between an object in motion and another object in still.

2.1 Between two points

Let P be fixed with coordinates \((x, y, z)\) and Q be point in motion. If the motion of Q is given in \(u = u(t), v = v(t),\) and \(w = w(t)\), then P and Q will be collided if and only if there exists a real number \(t\) such that

\[
u(t) = x, v(t) = y, \text{and } w(t) = z.
\]

2.2 Between a line segment and a point in a plane

Let L be a fixed line segment with end points A and B and Q be the point in motion as shown in Figure 4. Let \((x_0, y_0)\) and \((x_1, y_1)\) be coordinates of A and B, respectively. If the motion of Q is given by \(u = u(t), v = v(t),\) and the line segment L is given by \(px + qy + c = 0\), then Q will collide with L if and only if there exists a real number \(t\) such that

\[
pu(t) + qv(t) + c = 0
\]

And

\[
= \sum_{k=0}^{1} \sqrt{(x_1 - x_k)^2 + (y_1 - y_k)^2}.
\]

![Figure 4.](image)

2.3 Between a circle and a point in a plane

As indicated in Figure 5, let’s consider a circle \(\mathcal{C}\) with center \(C(a, b)\) and radius of \(r\). Let Q be the point in motion. Then Q will collide with the circle if and only if there exists a real number \(t\), such that

\[
(u(t) - a)^2 + (v(t) - b)^2 = r^2
\]

![Figure 5.](image)
We can generalize it to three dimensional case: if \( Q \) is the point in motion in space, let \( Q \) be given by \((u(t), v(t), w(t))\) and a sphere with center at \((a, b, c)\) and radius of \( r \) as shown in Figure 6. Then \( Q \) will collide with the sphere if and only if there exists a real number \( t \) such that

\[
(u(t) - a)^2 + (v(t) - b)^2 + (w(t) - c)^2 = r^2
\]

**Figure 6**

2.4 Between a triangle and a point

Let us consider a triangle with the set of vertices \((x_k, y_k): k = 1, 2, 3\) as shown in Figure 7.

Assume \( Q \) is the point in motion with coordinates \((u(t), v(t))\). Then point \( Q \) will collide with the triangle if and only if

\[
\sum_{k=1}^{3} \left| (x_{h(k)} - x_k)(v(t) - y_k) - (y_{h(k)} - y_k)(u(t) - x_k) \right|
\]

Where \( h(k) = k + 1 \) if \( k = 1, 2, 3 \) and \( h(3) = 1 \). Where \( x_1 = 1 \) and \( y_2 = 2 \).

**Figure 7**

2.5 Between a polygon and a point

Let us consider a rectangle \(((x, y): a \leq x \leq b, c \leq y \leq d)\) and \( Q \) as point in motion. Then point \( Q(u, v) \) will collide with the rectangle if and only if there exists a real number \( t \) such that

\[
a \leq u(t) \leq b \text{ and } c \leq v(t) \leq d.
\]

If the rectangle does not have sides that are parallel to the axes, then the above detection will not work. In this case, we can use the following criteria. Assume the rectangle has area \( T \) with consecutive vertices \((x_k, y_k), k = 1, 2, 3, 4\). Then \( Q \) will collide with the rectangle if and only if there exists a real number \( t \) such that

\[
\sum_{k=1}^{4} \left| \begin{array}{ccc} u(t) & v(t) & 1 \\ x_k & y_k & 1 \\ x_{h(k)} & y_{h(k)} & 1 \end{array} \right| = 2T
\]

This is based on the fact that the total area formed by \( Q \) with any two consecutive vertices of the rectangle equals to the area of the rectangle. Where \( h(k) = k + 1 \) if \( k = 1, 2, 3 \) and \( h(4) = 1 \).

As we can see that the problem of detection of collision is different from the PIP that is to determine whether a point is inside a polygon (PIP). Therefore, it is not a good idea to use the method of Ray Casting (see [1] and [2], [12]) because it needs to find number of intersections with the boundary of the polygon, which will be time-consuming.

2.6 Between a set of polygon and a point

If an object is consisting of a set of non-overlapping polygons as shown in Figure 8, then we can use the method given in previous section to check each of polygons involved. If an object is consisting of a large polygon and some small polygons that are contained in the large polygon as shown in Figure 9, then we can use the above method to check the large polygon if we just need to determine whether the point \( Q \) will collide the object. If we need determine whether the point will collide with small polygons, then we can apply the method to check the small polygons.

**Figure 8**

2.7 Between two line segments

Let us consider two line segments \( L_k, k = 1, 2 \) as shown in Figure 10. It is easy to verify that these two segments intersect if and only if \( d = \left| \begin{array}{ccc} x_1 - x_0 & y_0 - y_1 \\ x_1' - x_0' & y_0' - y_1' \end{array} \right| \neq 0 \) and \( \min\{|x_0, x_1|, \max\{|x_0, x_1|\} \leq \bar{x} \leq \min\{|x_0, x_1|, \max\{|x_0, x_1|\} \} \) and \( \min\{|y_0, y_1|, \max\{|y_0, y_1|\} \leq \bar{y} \leq \min\{|y_0, y_1|, \max\{|y_0, y_1|\} \} \). Where the intersection point \((\bar{x}, \bar{y})\) can be found by Cramer’s rule, in fact, \( \bar{x} = \frac{x_1 - x_0}{x_1' - x_0'} \) and \( \bar{y} = \frac{y_0 - y_1}{y_0' - y_1'} \). If \( d = 0 \), it means these two line segments are parallel. If any three end points of them form a triangle with non-zero area, then they don’t collide. Otherwise, they collide. That is, if \( d = 0 \) and

\[
\left| \begin{array}{ccc} x_0 & y_0 & 1 \\ x_0' & y_0' & 1 \\ x_1 & y_1 & 1 \end{array} \right| = 0
\]

Then they collide.
2.8 **Between one triangle and a line segment**

Consider a triangle with the set of vertices \( \{(x_k, y_k); 1 \leq k \leq 3\} \) and a line segment \( L \) with end points \( A \) and \( B \) as shown in Figure 11. To check for any collision between the line segment and the triangle, we can perform a loop for each side of the triangle and the segment \( L \) by applying the method given in section 2.7. If the line segment does not intersect any side of the triangle, we just need check whether it lies entirely inside the triangle by checking one of the end points of \( L \) by using the method given in section 2.4 above.

![Figure 11](image)

2.9 **Between two triangles both in motion**

Consider triangles \( T_k \) \( (k = 1, 2) \) with the sets of vertices \( S_k = \{(x^k_j, y^k_j); 1 \leq j \leq 3\} \), \( (k = 1, 2) \) as shown in Figure 12. To check for any collision between these two triangles, we can perform a loop for each side of \( T_1 \) and consider that side and the triangle \( T_2 \) by applying the method given in section 2.8.

![Figure 12](image)

2.10 **Between two line segments in space**

Consider two line segments in space as shown in Figure 10. \( L_k(t) = (x_k, y_k, z_k) + t < p_k, q_k, r_k >, t \in [0, 1]; k = 1, 2 \). It is well known that

\[
L_1 \parallel L_2 \text{ if and only if } \frac{p_1}{p_2} = \frac{q_1}{q_2} = \frac{r_1}{r_2}.
\]

And

\[
L_1 \cap L_2 \text{ if and only if } \frac{p_1}{p_2} = \frac{q_1}{q_2} = \frac{r_1}{r_2}.
\]

2.11 **Between a triangle and a line segment in space**

Consider a triangle and a line segment \( L \). To check for any collision between the line segment and the triangle, we can perform a loop for each side of the triangle and the segment \( L \) by applying the method given in section 2.8. If \( L \) does not collide with any side of the triangle, we need check whether it lies entirely inside the triangle by checking the signs of the normal vectors of the triangle surface, \( \vec{n}_k \) \( (k = 1, 2) \) are vectors with one of the end points of \( L \) as an initial point and the point \( P \) as the terminal point, where the point \( P \) is defined as

\[
P = \left( \frac{1}{2} \sum_{k=1}^{3} x_k \vec{x}_k + \frac{1}{2} \sum_{k=1}^{3} y_k \vec{y}_k + \frac{1}{2} \sum_{k=1}^{3} z_k \vec{z}_k \right),
\]

which lies on the surface of the triangle with vertices \( V_k \) \( (k = 2, 3) \). If the signs of \( \vec{n} \cdot \vec{u}_k \) and \( \vec{n} \cdot \vec{u}_2 \) are different, then we need check to see if the distance from the point \( P \) to \( L \) is less than the min\( d(P, V_k); k = 1, 2, 3 \). If it is true, then \( L \) penetrates the surface of the triangle, otherwise, it does not penetrate.

2.12 **Between two triangles in space**

Consider triangles \( T_k \) \( (k = 1, 2) \). To check for any collision between these two triangles, we can perform a loop for each side of \( T_1 \) and consider that side and the triangle \( T_2 \) by applying the method given in section 2.11.
3 Conclusions and Future Work

This paper has presented some mathematical solution for some fundamental bounding shape in game collision detection in 2D and 3D. Our method is easy to implement without a lot of performance issues in per-pixel based algorithm, but the accuracy and precision improved greatly compared with pixel based collision detection mechanism.

In the future, collision between combination of different fundamental bounding shape and their represented visual object will be studied.

4 References

[3] C code to determine if a point is in a polygon http://www.cs.unc.edu/~geom/pnpoly.html
[9] SQL code to determine if a point is in a polygon (http://www.sql-statements.com/point-in-polygon.html)
[10] Pascal code to verify if a point is inside a polygon (http://wiki.lazarus.freepascal.org/Geometry_in_Pascal)

#Checking_if_a_point_is_inside_a_poly