Collision Detection in Two Dimensions

Project 3 - P03
CS6491 Computer Graphics
Georgia Institute of Technology
Course Instructor: Jarek Rossignac
30 September 2008

Authors:

Dana Forsthoefel
gtg771w@mail.gatech.edu

www.prism.gatech.edu/~gtg771w/CS6491/

Adam Fitzgerald
gtg485v@mail.gatech.edu

www.ampheck.com/6491/

Abstract

In this paper, we give a detailed explanation of how to animate a 2D scene with colliding disks that move in a constant (possibly null) velocity and are subject to elastic shocks. First, we explain the derivations of a few formulas that are used in implementations of collision detection techniques, including formulas for predicting the time to first collision, for checking for static interfaces, and for computing the new velocities after elastic shock. Using these formulas, we then describe the implementation of two different collision detection techniques, PIT (Periodic Interference Test) and PIC (Predicted Instant of Collision). The PIT technique checks which pairs of disks interfere at the end of each frame, computes new velocities for these pairs, sets the disks' velocities to the new velocities, and repeats for every frame. On the other hand, the PIC technique computes the time t of the first collision (if one occurs before the next frame), advances the animation to t, computes new velocities after the shock, and then repeats for every frame. After comparing these techniques, we present four clear cases where the PIT collision detection technique produces incorrect trajectories including delay, wrong direction, missed collisions, and incorrect handling of nearly simultaneous collisions.

Table of Contents

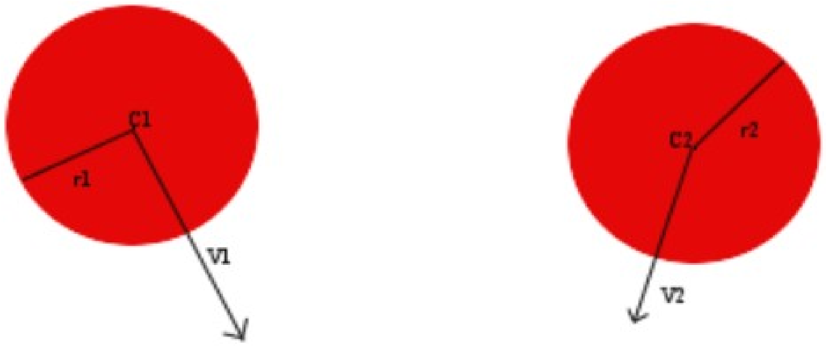
I. Derivations and Implementations of Formulas

The following section provides and explains the derivations and implementations of the formulas for predicting the time to first collision, for checking static interfaces, and for computing the new velocities after elastic shock.

I.1 Predicting Time to First Collision

I.1.1 Derivation:

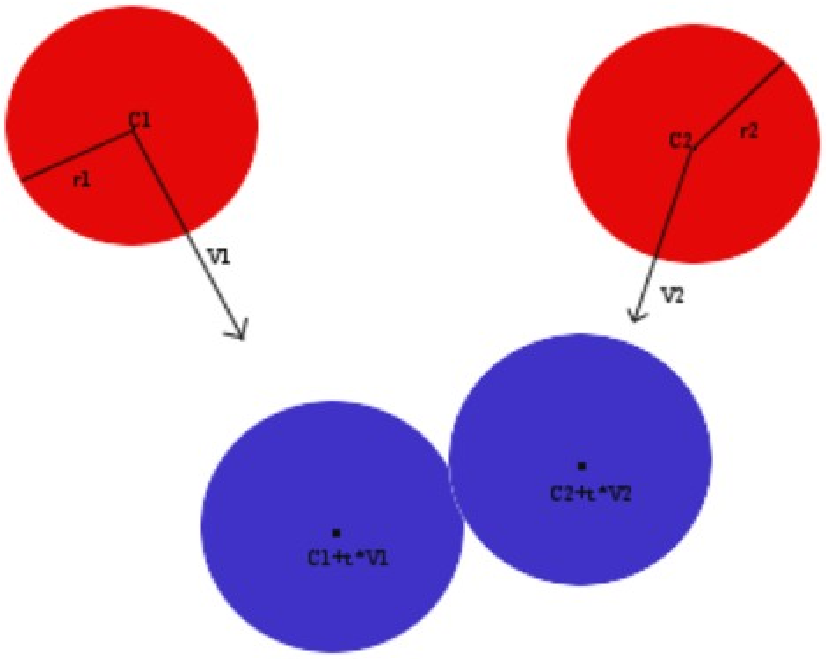
Here we will derive the formula for predicting the time to first collision. First, we set up the scenario for our derivation. We have two disks, the first with center C_1, radius r_1, and velocity vector V_1, the second with center C_2, radius r_2, and velocity vector V_2. The scenario can be seen below with the initial positions of the disks shown in red.



There are eight steps in this derivation.

Step 1: Find out where disks are at time t

The velocity vectors shown in the figure above indicate that the two disks are moving toward each other. We can see in the figure below, that the disks collide at time t in the positions shown in blue. The centers for the two disks at time t are at C_1+t*V_1 and C_2+t*V_2 for disks 1 and 2 respectively.



Step 2: Write condition in English that says this is what happens in collision

The distance between the two centers at the time of collision will be the sum of the radii of the two disks, r_1 and r_2. So in English, the condition that says what happens in the collision is:

distance between centers = sum of the radii

Step 3: Eliminate the square root

The distance formula is:

$$d = \text{sqrt}((x_2-x_1)^2 + (y_2-y_1)^2)$$

Square root operations are expensive, so we want to eliminate this use of this operation here. To do this, we square both sides of the equation from Step 2. The result is:

distance between centers^2 = sum of the radii^2

or

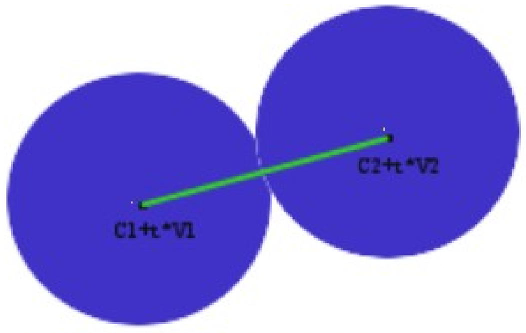
$$d^2= (x_2-x_1)^2 + (y_2-y_1)^2$$

Step 4: Substitute new centers into the equation

We found the new centers before as C_1+ t*V_1 (for disk 1) and C_2+t*V_2 (for disk 2) at time t of collision. Now the centers of the disks need to be substituted into the equation because the disk centers have moved:

$$((C_2+t*V_2) - (C_1+t*V_1))^2 = (r_1+r_2)^2$$

This gives the vector shown in the figure below in green.



Step 5: Use vectors

We want to use vectors to write our equation in a different way. C_1C_2 is a vector because it is two points together and V_2-V_1 are two subtracted vectors which results in a vector. Our resulting equation incorporating the use of vectors is:

$$(C_1C_2 + t*(V_2-V_1))^2 = (r_1 + r_2)^2$$

Step 6: A vector squared equals the dot product

Using the property for vectors that says a vector squared is equal to the dot product of the two vectors :

$$V^2 = V\cdot V = n(V)^2 = V_x^2 + V_y^2$$

our equation becomes:

$$(C_1C_2 + t*(V_2\text{-}V_1)) \cdot (C_1C_2 + t*(V_2\text{-}V_1)) = (r_1 + r_2)^2$$

Step 7: Distribute dot product over vector addition

Now we distribute the dot product in our equation over vector addition:

$$C_1C_2 \cdot C_1C_2 \ + \ 2 * (C_1C_2 \cdot t*(V_2\text{-}V_1)) + (t*(V_2\text{-}V_1))^2 = (r_1 + r_2)^2$$

We can use the property of vectors again that says a vector squared is equal to the dot product of two vectors to obtain the formula:

$$[C_1C_2 \cdot C_1C_2] \ + \ [2*(C_1C_2 \cdot (V_2\text{-}V_1))]*t + [(V_2\text{-}V_1) \cdot (V_2\text{-}V_1)]*t^2 = (r_1 + r_2)^2$$

Step 8: Solve for t

Now you can solve for t, you will get scalars for the dot products which will be coefficients for a quadratic equation over t.

For the equation $Ax^2 + Bx + C = 0$, The quadratic formula is:

$$(-B \text{ +- } \mathrm{sqrt}(B^2 - 4AC))/\ (2A)$$

We rearrange our formula to fit it into the quadratic formula:

$$[(V_2\text{-}V_1) \cdot (V_2\text{-}V_1)]*t^2 + [2*(C_1C_2 \cdot (V_2\text{-}V_1))]*t + [C_1C_2 \cdot C_1C_2 - (r_1 + r_2)^2] = 0$$

For our equation, the coefficients for the quadratic formula are:

$$\begin{array}{ccccccc} [(V_2\text{-}V_1) \cdot (V_2\text{-}V_1)]*t^2 & + & [2*(C_1C_2 \cdot (V_2\text{-}V_1))]*t & + & [C_1C_2 \cdot C_1C_2 - (r_1 + r_2)^2] & = 0 \\ A*t^2 & + & B*t & + & C & = 0 \end{array}$$

$$A = (V_2\text{-}V_1) \cdot (V_2\text{-}V_1)$$
$$B = 2*(C_1C_2 \cdot (V_2\text{-}V_1))$$
$$C = C_1C_2 \cdot C_1C_2 - (r_1 + r_2)$$

We can then solve for t by finding the roots of the quadratic equation using the quadratic formula above. The least non-negative value of t that results from this formula is the time t of the first collision (because we want to smallest possible time that is non-negative).

I.1.2 <u>Implementation</u>:

Now that we have derived the formula for predicting the time to first collision we can now discuss its implementation. Below is the processing code function called "collision" that computes the collision time, t and returns it if t is between 0 and 1. If t is not between 0 and 1, the function returns the number 2 to indicate that there is no collision.

The function takes in BALL A and BALL B which are two ball objects that include the information of the center, the velocity, the radius, and the mass of a ball.

```
1    float collision(BALL A, BALL B) {
2        vec W=M(-1,A.V,1,B.V);
3        vec D=V(A.C,B.C);
4        float a=dot(W,W);
5        float b=2*dot(D,W);
6        float c=dot(D,D)-sq(A.r+B.r);
7        float d=sq(b)-4*a*c;
8        if (d>=0) {
9            float t1=(-b-sqrt(d))/2/a;
10           if(t1<0) {
11               t1=2;
12           }
13           float t2=(-b+sqrt(d))/2/a;
14           if(t2<0) {
15               t2=2;
16           }
17           float m=min(t1,t2);
18           if ((-0.02<=m)&&(m<=1.02)){
19               return m;
20           }
21       }
22       return 2;
23   }
```

First, line 2 sets W = (V_2-V_1) from above and line 3 sets D = C_1C_2. Then, lines 4, 5, and 6 set A, B, and C respectively according to the coefficient equations derived above and shown again below:

$$A = (V_2-V_1) \cdot (V_2-V_1)$$
$$B = 2*(C_1C_2 \cdot (V_2-V_1))$$
$$C = C_1C_2 \cdot C_1C_2 - (r_1 + r_2)$$

Then line 7 calculates the discriminant of the quadratic equation, or $B^2 – 4AC$. Then line 8 checks if the discriminant is legal (greater than 0) and then in lines 9 through 20 the quadratic formula is implemented. First, in lines 9 through 12, the case of subtracting the discriminant is tried to obtain the first root of the quadratic equation:

$$(-B - \text{sqrt}(B^2 – 4AC))/ (2A)$$

Then, in lines 13 through 16, the case of adding the discriminant is tried to obtain the second root of the quadratic equation:

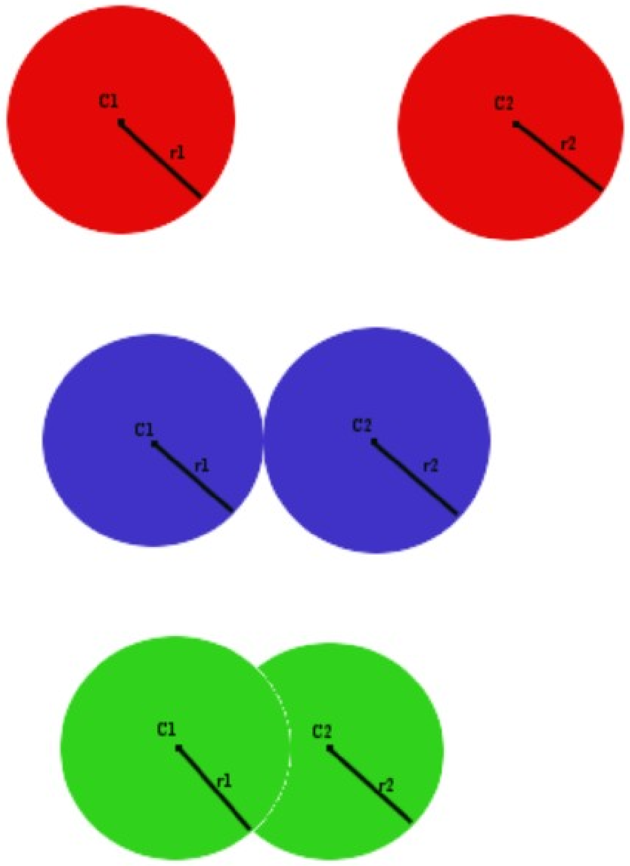
$$(-B + \text{sqrt}(B^2 – 4AC))/ (2A)$$

Finally, in remaining lines, we test a root between 0 and 1 has been found. If such a root or roots exist, the minimum non-negative root between 0 and 1 is returned. If not, the number 2 is returned by the function.

I.2 Checking Static Interferences

I.2.1 Derivation:

Here we will derive the formula for checking static interferences. We will derive a formula that can detect if two objects are intersecting or interfering at any given instant of the animation. First, we set up the three scenarios for our derivation. In all three scenarios we have two disks, the first with center C_1, radius r_1, the second with center C_2, radius r_2. The scenarios can be seen below in the figure with the positions of the disks shown in red for scenario 1, shown in blue in scenario 2, and shown in green for scenario 3.



There are two steps to this derivation.

Step 1: Write condition in English that says if there is an interference

We know that the disks are interfering if the distance between the two centers of the disks is less than or equal to the sum of the radii of the two disks, r_1 and r_2. So in English, the condition that says if there is an interference is:

distance between centers < = sum of the radii

We can verify this solution by examining the three scenarios presented in the figure above. In scenario 1 (the red disks), the distance between the centers is greater than the sum of the radii, therefore this formula would say this scenario contains no interference which is correct. In scenario 2 (the blue disks), the distance between the centers is equal to the sum of the radii, therefore this formula would say this scenario does contain an interference which is also correct. Lastly, in scenario 3 (the green disks), the distance between the centers is less than the sum of the radii, therefore this formula would say this scenario does contain an interference which is correct. We have verified our condition of interference and we can now move on.

Step 2: Give the equality a threshold

The equality we have formed needs some threshold for interference between the two disk. We can choose a threshold of 0.01 so that:

distance between centers < = sum of the radii + 0.01

Testing this inequality with information about the two disks at a given instant in the animation will tell us whether the two disks are interfering so this function therefore provides a way of checking for static interferences.

I.2.2 Implementation:

Now that we have derived the formula for checking static interferences we can now discuss its implementation. Below is the processing code function called "interfere" that computes whether two balls are interfering. If the two balls are interfering, the function returns the boolean 1, or true. Otherwise, if the two balls are not interfering, the function returns the boolean 0, or false.

The function takes in BALL A and BALL B which are two ball objects that include the information of the center, the velocity, the radius, and the mass of a ball.

```
1    boolean interfere(BALL A, BALL B) {
2        if(A.r>0) {
3            return d(A.C,B.C)<=A.r+B.r+0.01 ;
4        }
5        else {
6            return d(A.C,B.C)>=-A.r-B.r-0.01 ;
7        }
8    }
```

First, line 2 tests if the radius in the ball object has been specified as positive or negative. If the radius is specified as positive, line 3 performs the interference check for positive radii. First, the distance between the two centers of the balls is calculated, then it is compared to the sum of the radii of the two ball plus a threshold of 0.01. This line implements the formula we derived above:

distance between centers < = sum of the radii + 0.01

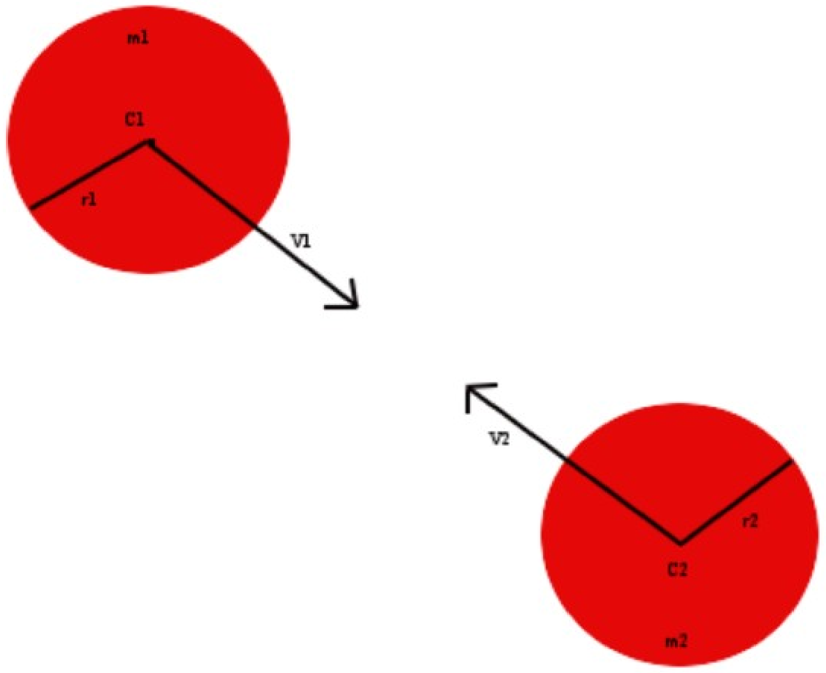
If this inequality is true, the function returns 1 to indicate that the two balls interfere. Otherwise, the function returns 0 to indicate the two balls do not interfere.

If the radius of a ball object has been specified as negative, line 6 performs the interference check for negative radii.

I.3 Computing the New Velocities After Elastic Shock

I.3.1 Derivation:

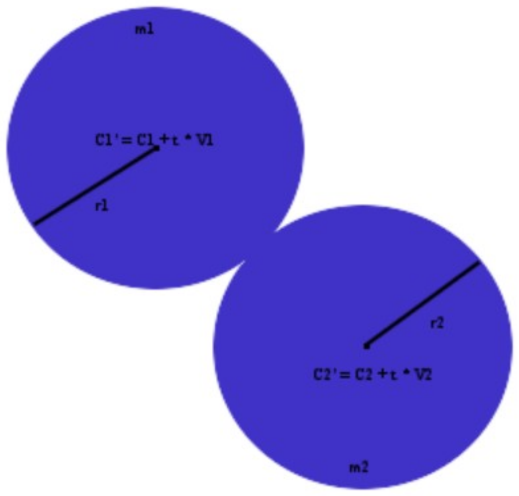
Here we will derive the formula for computing the new velocities after elastic shock where kinetic energy and moment of inertia are preserved. First, we set up the scenario for our derivation. We have two disks, the first with center C_1, radius r_1, velocity vector V_1, and mass of m_1 the second with center C_2, radius r_2, velocity vector $V_{2,}$, and mass m_2. The scenario can be seen below with the initial positions of the disks shown in red.



There are 5 steps to this derivation.

Step 1: Find out where the disks are at time t

The velocity vectors shown in the figure above indicate that the two disks are moving toward each other. We can see in the figure below, that the disks collide at time t in the positions shown in blue.



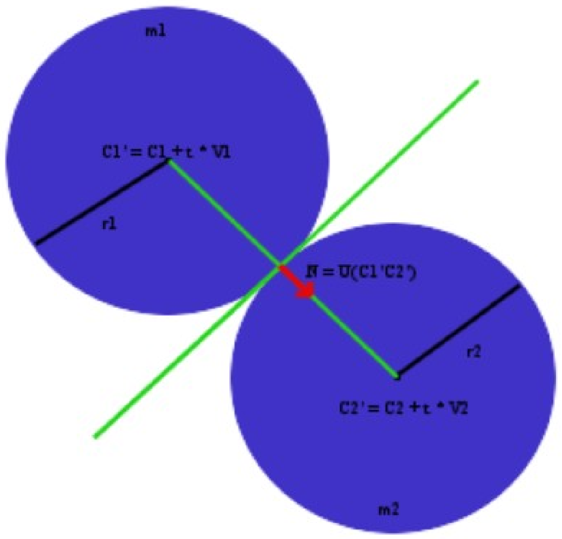
The centers for the two disks at time t are at C_1+t*V_1 and C_2+t*V_2 for disks 1 and 2 respectively. We set these values equal to the variables:

$$C_1' = C_1+t*V_1$$
$$C_2' = C_2+t*V_2$$

Step 2: Compute normal vector

We compute the vector N, orthogonal to the plane of contact. This vector is the normal of the tangential contact plane of collision and can been seen in the figure below in red. N is equal to the unit vector of a vector created between the centers of the two disks at time t:

$$N = U(C_1'C_2')$$



Step 3: Compute Δ

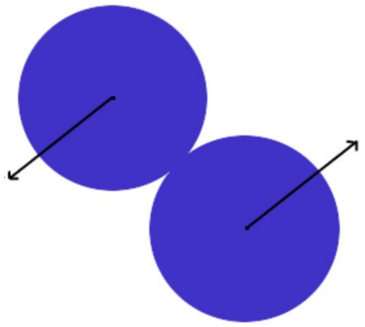
Now we want to create a vector in the normal direction with its magnitude related to the normal components of the initial velocities. First we compute the normal component of the relative velocities:

$$(V_2-V_1) \cdot N$$

Then we use this to compute vector Δ:

$$\Delta = ((V_2-V_1) \cdot N)\, N$$

We only look at the normal component of the velocities because there is no change in velocity for velocity components not in the direction of the normal. This can be seen in the figure below:



In the figure above, we can see that the final velocities of the two disks would not change as a result of this collision because there are no velocity components in the direction of the the normal of the tangential contact plane of collision.

Step 4: Compute fraction of Δ using ratio of masses

Next we find the ratio of masses that will be multiplied by Δ to get the fraction of Δ that contributes to the final velocity. The heaviness of the balls colliding will affect the final velocities of both balls if one takes into account the conservation of energy and momentum. The fraction of masses can be seen below:

$$\text{disk 1:} \qquad m_2/[(m_2+m_1)/2] * \Delta$$
$$\text{disk 2:} \qquad m_1/[(m_2+m_1)/2] * \Delta$$

Step 5: Write equations for final velocities

Finally, we can write the equations for the new velocities after elastic shock, V_1' and V_2', for disks 1 and 2 respectively:

$$V_1' = V_1 + m_2/[(m_2+m_1)/2] * \Delta$$
$$V_2' = V_2 + m_1/[(m_2+m_1)/2] * \Delta$$

If disk 2 is much heavier than disk 1, the Δ term will dominate the final velocity of disk 1. If disk 2 is much lighter than disk 2, the Δ term will be eliminated in the final velocity of disk 1 because it doesn't matter how the other disk is moving, its mass is too small to affect disk 1's velocity.

I.3.2 Implementation:

Now that we have derived the formula for computing the new velocities after elastic shock where kinetic energy and moment of inertia are preserved, we can now discuss its implementation. Below is the processing code function called "shock" that computes new velocities for elastic shock. The function does not return the velocities, but instead modifies the velocities of the balls passed to it to equal the new, calculated velocities.

The function takes in BALL A and BALL B which are two ball objects that include the information of the center, the velocity, the radius, and the mass of a ball.

```
1    void shock(BALL A, BALL B) {
2            vec N = U(V(A.C,B.C));
3            vec D=S(dot(M(-1,A.V,1,B.V),N),N);
4            A.V.setTo(M(1,A.V,2*B.m/(A.m+B.m),D));
5            B.V.setTo(M(1,B.V,-2*A.m/(A.m+B.m),D));
6    }
```

First, line 2 sets N, the normal of the tangential contact plane of collision, using the equation we derived above in Step 2:

$$N = U(C_1'C_2')$$

Then, line 3 sets Δ, a vector in the normal direction with its magnitude related to the normal components of the initial velocities, using the equation we derived above in Step 3:

$$\Delta = ((V_2 - V_1) \cdot N)\, N$$

The final velocities are computed and set in lines 4 and 5 of the function. The final velocities are computed using the final equations we derived above:

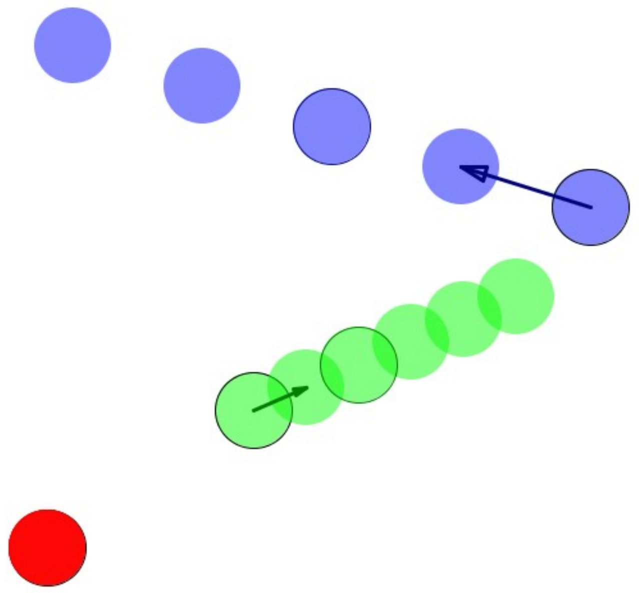
$$V_1' = V_1 + m_2/[(m_2+m_1)/2] * \Delta$$
$$V_2' = V_2 + m_1/[(m_2+m_1)/2] * \Delta$$

The function does not return the velocities, but instead modifies the velocities of the balls passed to it to equal the new, calculated velocities.

II. PIT and PIC

The following section explains two animation techniques for collision detection, PIT and PIC. Then, these two techniques are compared so we may obtain a full understanding of their differing implementations.

An example setup of the scenario we will use to describe this technique is shown in the figure below. The scene holds three disks. All disks have the same radius. The blue and green disks can be moved to simulate collisions and each has a velocity vector indicating its direction and speed of movement. For the blue and green disks, the path of each disk's movement is shown using multiple copies of the disk spaced apart from each other in time. The closer the disks in a disk path are together, the faster the disk itself is moving.

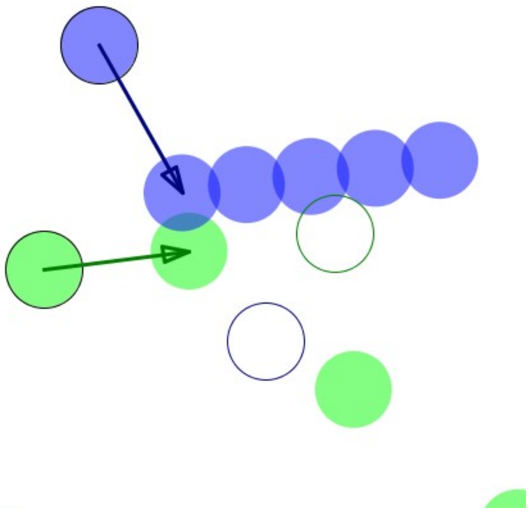


We simulate movement of either the blue or green disk by first selecting an initial position for the that disk. This is done by placing the first path disk on the scene where desired. Then, we can place the third disk in the disk path to select the direction and speed of that disk in the animation. The closer the third disk is place to the first disk in the disk path, the faster the disk itself is moving.

The red disk is modeled as having a very high mass, therefore it can be used as a barrier for the other two disks to run into. The red disk can be moved around to different places in the scene but we cannot control its velocity. The red disk does not move when the other disks collide with it because its mass is so high. The red disk will not be used in this section to explain PIT and PIC, however, it will be used later in some further simulations of the two techniques.

II.1 PIT

The first animation technique for collision detection that we will approach is the Periodic Interference Test, or PIT, technique. PIT detects collisions at the end of each frame, and computes new velocities for all pairs of interfering disks. Shown in the figure below is the collision detection of two disks using the PIT method. Note that this figure has the same initial configuration as the that of the figure that will be used when describing the PIC technique below in section II.2. Also note that the collision detection figures from the two techniques show different resulting disk velocities and configurations.



There are five steps in this technique.

Step 1: Move every disk one frame

Each disk in the scene is moved forward with its initial velocity for one full frame. The position of the disk at the end of this frame is then drawn in the scene. This can be seen in the figure above, the second disks in the paths of both the green and blue disks are drawn according to where they ends up after one frame.

Step 2: Check for static interference

At the end of the frame, each pair of disks is compared to identify if there is any static interference between the disks. This is done using the “interfere” function described previously in section I.2.2 which computes whether two disks are interfering.

Step 3: Calculate resulting velocities

If an interference is detected in Step 2, the “shock” function described previously in section 1.3.2 is used to calculate the new velocities that result from the collision. The shock function is called only at the end of the frame so it interprets the collision as happening at that moment in time at the end of the frame (the collision could have happened anytime within that time frame but the PIT method only looks for collisions at the end of frames).

Step 4: Change the velocities of the disks

The “shock” function then sets the velocities of the disks to the new velocities calculated within the shock function that resulted from the collision.

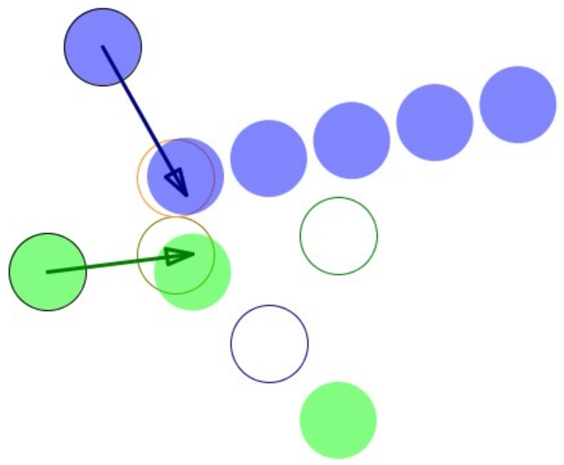
Step 5: Repeat for every frame

This process is repeated every frame using the disk velocities from the previous frame until all movement in the scene is completed.

In summary, PIT checks which pairs of disks interfere at the end of each frame, computes new velocities for these pairs, sets the disks' velocities to the new velocities, and repeats for every frame until all movement is finished.

II.2 PIC

The second animation technique for collision detection that we will approach is the Predicted Instant of Collision, or PIC, technique. PIC simulates the animation between frames by computing the time t to the first collision (if any) between the frames, rolling the animation until time t, computing new velocities for the colliding pair, and iterating until the time for the next frame is reached. Shown in the figure below is the collision detection of two disks using the PIC method. Note that this figure has the same initial configuration as the that of the figure used when describing the PIT technique above in section II.1. Also note that the collision detection figures from the two techniques show different resulting disk velocities and configurations.



There are 6 steps in this technique.

Step 1: Check for collisions

At beginning of each frame, before moving anything, we compare all disks to all other disks once to predict the earliest collision times of disk pairs in the scene. To do this, we use the method described in section I.1, “predicting the time to first collision,” that uses the “collision” function described in detail in section I.1.2 which takes in the initial velocities and positions of the disks to predict the earliest collision time between the two disks.

Step 2: Advance all the disks

The earliest collision prediction time was chosen in Step 1. Now, the disks are moved to the positions they would occupy at that collision time. This can be seen in the figure above. The initial positions of the disks are shown as the first disks in the two paths. The disks are then advanced to their positions at the collision time. This is shown in the figure as two unfilled disks colliding just before the disks for the second time frame are drawn.

Step 3: Calculate the new velocities

The "shock" function described previously in section 1.3.2 is used to calculate the new velocities that result from the collision at the collision time calculated in Steps 1 and 2. In this, the PIC, technique, the shock function can be called at any point within a time frame. We call the shock function on the exact , calculated time when the collision occurs so PIC interprets collisions as happening exactly at the actual time of collision, not just at the end of the time frame like the PIT technique.

Step 4: Repeat collision calculations for all collisions in time frame

In the PIC technique, unlike the PIT technique, a collision can occur at any time within a time frame. In Step 2, the disks were advanced to the time of the collision. This advanced time may be less than the time step, in which case the collision calculations need to be repeated to make sure there are no other collisions in the time left in the time frame.

Step 5: Advance disks to end of time frame

If there are no more collisions in the time step, we advance the disks to the end of the time frame using the current velocities that have been calculated. In the figure above, this is when we draw the next disks in the disk paths.

Step 6: Repeat for every frame

This process is repeated every frame using the velocities from the previous frame until all movement in the scene is completed.

In summary, PIC computes the time t of first collision (if one occurs before the next frame), advances the animation to t, computes new velocities after the shock, and then does it again. If no collision occurs before the next frame, it advances the balls to the next frame.

II.3 PIT vs PIC

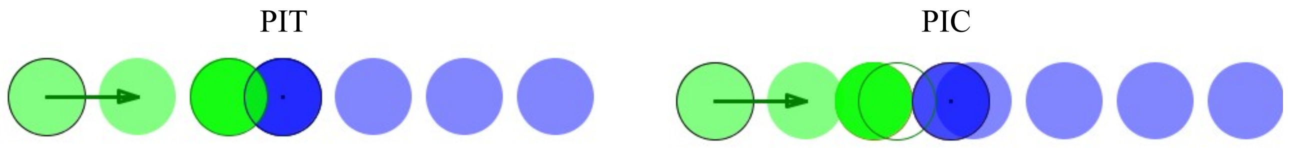
The fundamental difference between PIT and PIC is when they perform their sampling of collisions in the animation. PIT only looks for collisions at the end of each time frame by searching for interference. So collisions can only be detected at the end of a time frame with PIT. If collisions occur during the time frame, they can only be detected if they result in interference that can be detected by PIT at the end of the frame. So PIT essentially takes all collisions occurring during a given time frame and gives them the collision time t of the end of the time frame (if it detects the collision that occurred at all since there may be no interference at the end of the time frame). PIC, on the other hand, starts by computing the time t of the first collision within a time frame (if a collision exists within that frame) and advancing the disks to that collision time. So PIC can identify exact collision times within a time frame. Because PIC advances the disks to the detected collision time within the time frame and then continues to search for collisions within that frame, PIC can identify more than one collision per frame. PIT cannot identify more than one collision per frame, it only sees the collision that occurs by interference at the end of a particular frame.

III. Illustration and Explanation of Four Cases Using PIT

The following section presents four clear cases where the PIT collision detection animation technique described in section II is wrong. For each case, images are provided that show the trajectory computed by PIT (the wrong trajectory) and then the expected trajectory for the situation computed by PIC (the correct trajectory). The cases presented here include: delay, wrong direction, missed collisions, and incorrect handling of nearly simultaneous collisions.

III.1 Case 1 : Delay

The first case presented here in which the PIT collision detection technique does not perform correctly and provides an incorrect output is “delay” An example of this case can be viewed in the figure below. Both simulations (PIT and PIC) have the same initial configuration. The wrong, PIT simulation output can be seen on the left side of the figure, the correct, PIC simulation output is shown on the right.



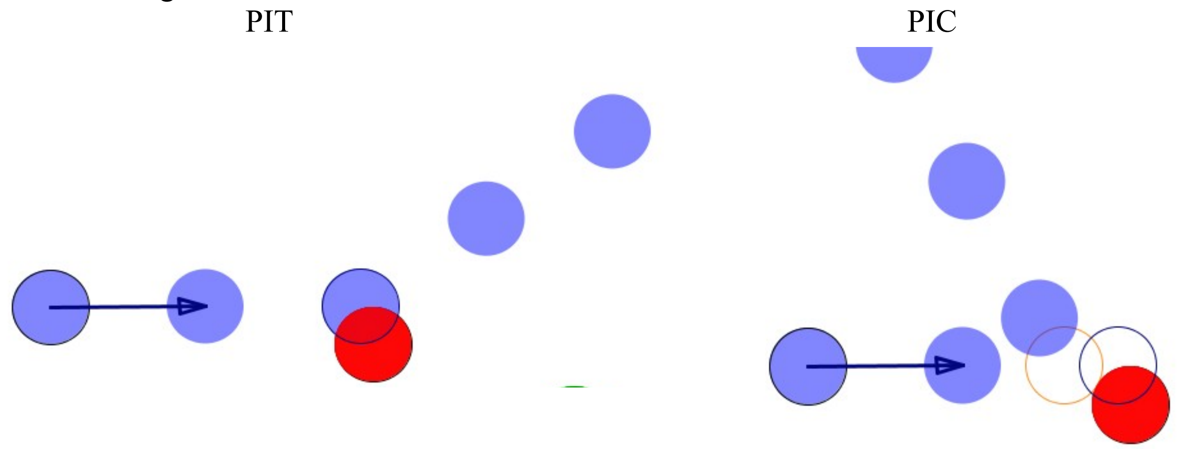
The PIT technique, shown on the left in the figure above, resulted in the blue disk moving to the right in the animation with a small delay on its movement when compared to the movement of the blue disk in the correct implementation shown on the right. To understand why this delay on the blue disk movement occurred in the PIT animation, we can study the two figures above.

In the PIT output on the left, we can see that the collision between the stationary blue ball and the moving green ball was detected later than it was in the PIC collision on the right. Because the PIT technique waited until the end of the frame to check for collisions, it saw the collision between the two balls later than the actual collision time as can be seen by comparing the two outputs in the figure above. Because the stationary blue ball was not hit by the green ball until a later time, the blue disk did not begin to move until a later time. Because of this, the blue disk was delayed in its motion in the PIT animation.

The correct timing for this scenario can be seen on the right in the figure above. Here, the collision was detected right when the green disk first touched the blue disk so the blue disk began its movement precisely when the two disks collided and its motion was not delayed in the PIC animation.

III.2 Case 2 : Wrong Direction

The second case presented here in which the PIT collision detection technique does not perform correctly and provides an incorrect output is "wrong direction." An example of this case can be viewed in the figure below. Both simulations (PIT and PIC) have the same initial configuration. The wrong, PIT simulation output can be seen on the left side of the figure, the correct, PIC simulation output is shown on the right.



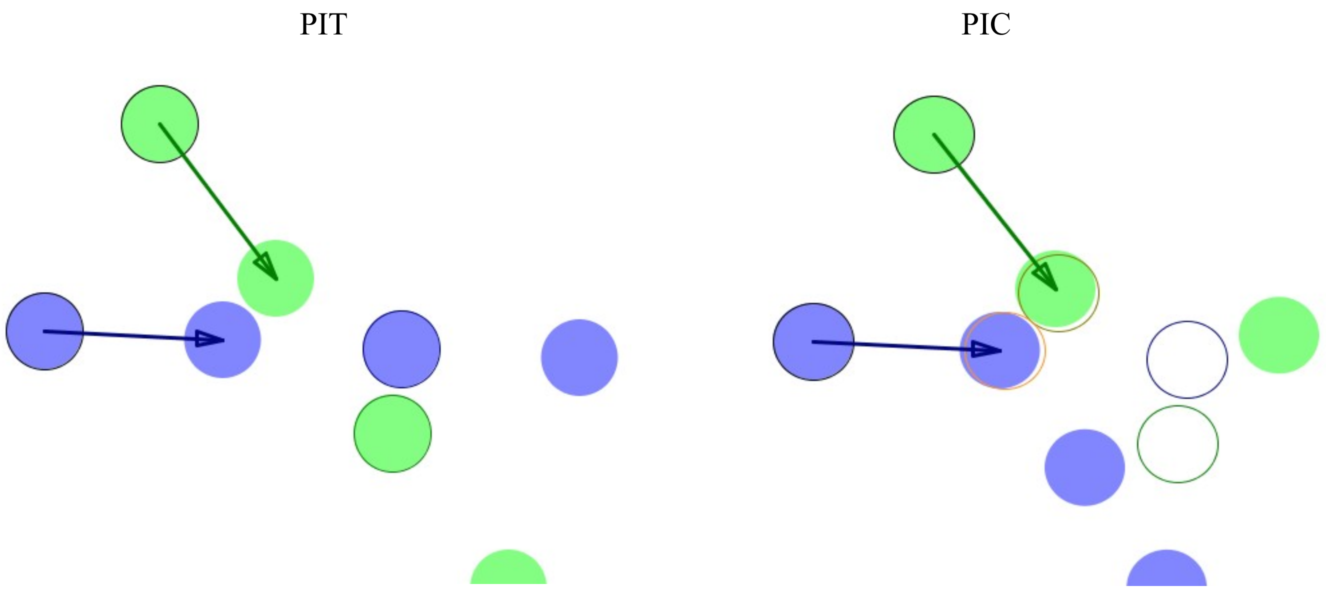
The PIT technique resulted in the blue disk moving in the wrong direction after the collision with the red disk. This indicates that the new velocity value calculated by the shock function was incorrect. To understand why this happened, we can study the two figures above.

In the PIT output on the left, we can see that the collision was detected later than it was in the PIC collision on the right. Because the PIT technique waited until the end of the frame to check for collisions, it saw the collision with the red ball at the time shown on the left in the figure above. PIT sent this time to the shock function and it calculated the final velocity for the collision as if it had occurred right at that point. So, this resulted in a final velocity in the wrong direction.

The correct final velocity for this scenario can be seen on the right in the figure above. Here, the collision was detected right when the blue disk first touched the red disk. When this time of collision was sent to the shock function, the final velocity was correctly calculated.

III.3 Case 3 : Missed Collisions

The third case presented here in which the PIT collision detection technique does not perform correctly and provides an incorrect output is “missed collisions.” An example of this case can be viewed in the figure below. Both simulations (PIT and PIC) have the same initial configuration. The wrong, PIT simulation output can be seen on the left side of the figure, the correct, PIC simulation output is shown on the right.



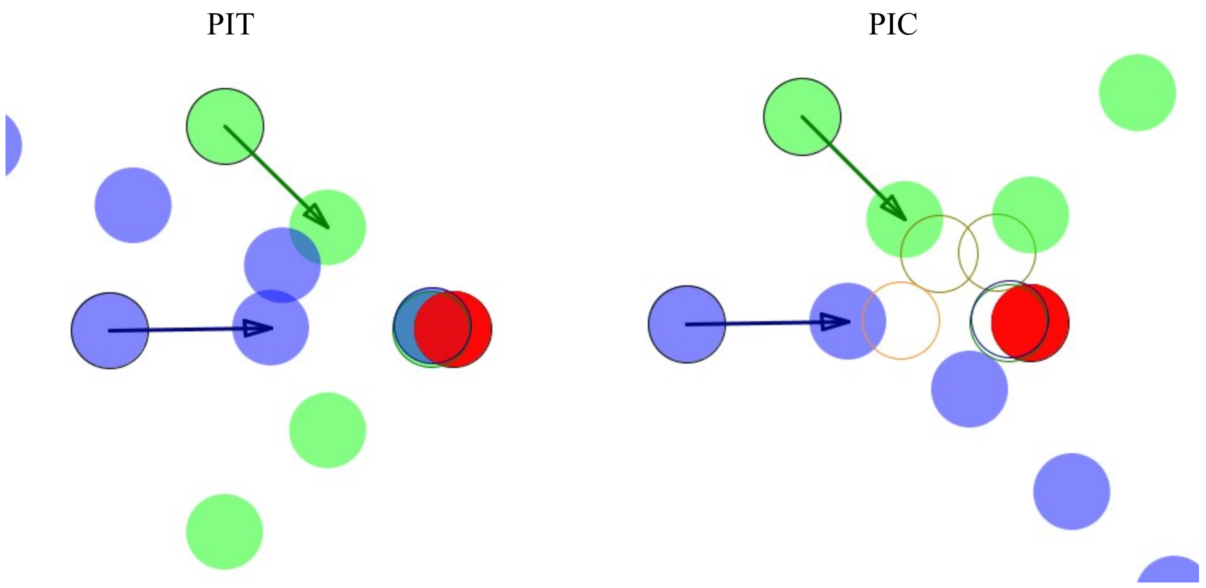
The PIT technique resulted in the blue and green disks completely missing each other in the animation. This indicates that no collision was found by the interfere function. To understand why this happened, we can study the two figures above.

In the PIC output on the right, we can see that the collision between the two disks occurred between the second and third time frames. We can tell this because the empty circles indicating the collision are present between the second and third disks on the disk paths for both the blue and green disks. In the PIT output on the left, we can see that the collision was completely missed because the PIT technique waited until the end of the frame to check for collisions. When PIT called the interfere function at the end of the second time frame (just before drawing the third disks in the blue and green disk paths), the function detected no interference between the green and blue disks because, as we can see on the left in the figure above, the two disks do not interfere at the end of the second frame.

The correct collision detection for this scenario can be seen on the right in the figure above. Here, the collision was correctly detected right when the blue disk touched the green disk at a time between the second and third frames.

III.4 Case 4 : Incorrect Handling of Nearly Simultaneous Collisions

The last case presented here in which the PIT collision detection technique does not perform correctly and provides an incorrect output is “incorrect handling of nearly simultaneous collisions.” An example of this case can be viewed in the figure below. Both simulations (PIT and PIC) have the same initial configuration. The wrong, PIT simulation output can be seen on the left side of the figure, the correct, PIC simulation output is shown on the right.



In the figure above on the right, we can see that in the correct implementation: first the blue disk should collide with the green disk, then the green disk should immediately collide with the red disk. These two collisions should both occur between the second and third time frame. This is an example of nearly simultaneous collisions (multiple collisions between two frames).

The PIT technique, shown on the left in the figure, did not properly handle the multiple collisions that happened between two frames. To understand why this happened, we can study the two figures above. Instead of first detecting the collision between the green and blue disks and then subsequently detecting the green disk's collision with the red ball, as is correct, the PIT technique waited until the end of the second frame to check for collisions and therefore saw the all collisions at the same time, including an incorrect collision between the blue and red disks because the trajectory of the blue disk had not been altered from the first collision.

So, PIT calculated the final velocities of the blue and green disks based on the assumption that all the collisions occurred at the end of the second time frame. This results in the wrong output shown on the left in the figure above.

IV. Conclusion

The two methods PIT and PIC are very different, and while it evident that for two-dimensional disks PIC is clearly superior, there are circumstances where it is not. Further, even for simulations of colliding two-dimensional disks, there are certain circumstances where the resulting motion from the PIT method is reasonably close to the PIC method for all collisions.

PIT is acceptable to use when it is reasonable to assume that it will miss or mishandle very few of the collisions that should occur in a "perfect" system. If this assumption can be made, then PIT can be chosen as the method of detecting collisions. It remains to find the conditions under which such an assumption is reasonable, however. We saw earlier four cases which can cause the PIT method to fail to accurately detect and resolve a collision. In all cases this is caused by the static interference being detected "too late." If we can ensure that all the interferences are detected before the discs have passed each other, we can reasonably assume that PIT will behave reasonably in all situations.

This quality of the scenario can be calculated from the relationship of three parameters:

- -The radius of the smallest disk in the simulation
- -The maximum velocity of the simulation.
- -The time step of the simulation.

If the discs are unable to move by a distance greater than the radius of the smallest disc in between static interference tests, then it is impossible for the centers of two discs to pass during that time.

Let t = a fixed time step
Let r = radius of the smallest disc in the simulation
Let v = maximum expected velocity in the simulation
IF $t*|v| < r/2$ THEN PIT will accurately find nearly all collisions in the simulation.

If the velocities are low enough that even when the discs are both headed straight towards each other the will not pass each others' centers in a single step, then PIT will function reasonably well. Furthermore, given a scenario to simulate, we can adjust the time step t such that $t*|v| < r/2$ remains true, and do multiple sub-steps if we want to advance more quickly. This only becomes infeasible when the discrepancy necessitates calculating very many sub-steps per frame.

Clearly, PIT is generally acceptable when the difference between the smallest object and fastest speed is not very large. There are some situations in which PIT is better suited than PIC, despite PIC being able to function well regardless of the relationship between object size, velocity, and time step. The disc is a shape that is extremely simple to give an implicit formulation of, and this is crucial in the derivation of the formula used to predict the collision time. Changing the shape would necessitate the re-derivation of an equation to solve, and a reimplementation of the collision functionality. On the other hand, the PIT method would not be especially more complex for different shapes. Because it is calculated by finding static interference, the collisions between various shapes would be much simpler to implement.

If the velocity were non-constant, we would need to model the motion using forces applied over the motion. This would further complicate solving for collision times, we need to predict at what time the force has influenced the velocity which as influenced the position to cause a collision. Compared to the changes to PIT to implement non-constant velocity, this is much more complicated. Again, because

PIT is a sampling method, we simply update the positions by newton's equations and then check for static interference.

The third case in which it is difficult to use PIC, when compared to PIT, is if the objects are rotating. The trend in the first two instances continues here: now instead of just worry about position and velocity, we need to account for rotational position and velocity. For example two boxes centered at some points x and y, may or may not be colliding at some time – it depends on their rotational position. And again, following the trend, this is an area where PIT is much simpler to implement. We again only need to update the simulation without worrying about predicting, and then we check for static interference.

V. References

Bobic, Nick. “Advanced Collision Detection Techniques”. Gamasutra. <http://www.gamasutra.com/features/20000330/bobic_01.htm>. 20 March 2000.

This paper walked through the building of collision detection formulas and techniques. It was a very helpful source when writing the derivations and implementations of the formulas in this paper.

Fauer, Kasper. “Improved Collision Detection and Response.” <http://www.peroxide.dk/papers/collision/collision.pdf> 25 July 2003.

This paper described laid out in detail a particular design of an algorithm for collision detection. This was a helpful source on how to successfully explain collision detection algorithms.

Hadap, Sunil and David Eberle. “Collision Detection and Proximity Queries”. SIGGRAPH Course. <http://www.gvu.gatech.edu/~jarek/graphics/papers/06CollisionCourseSIG04.pdf> 2004.

This paper described a half-day course on animation, focusing on collision detection. It provided a compilation of papers, slides, and other information on collision detection techniques that covered a wide range of issues.

Miller, Kurt. “Basic Collision Detection”. Flipcode.com. <http://www.flipcode.com/archives/Basic_Collision _Detection.shtml> 21 January 2000.

This paper provided a helpful tutorial on basic collision detection and provided some small improvements that can be made to the algorithms used. This was helpful in researching the derivations and implementations of the collision detection formulas.