Simulating the Rigid World

# Resolving Collisions

With a functioning positional correction system, you can now begin implementing collision resolution and support behaviors that resemble real-world situations. In order to focus on the core functionality of a collision resolution system, including understanding and implementing the Impulse Method and ensuring system stability, you will continue to work with axis-aligned rigid rectangles. The complications associated with angular impulse resolutions will be examined in the next section, after the mechanics behind linear impulse resolution are fully understood and implemented.

In the following discussion, the rectangles and circles will not rotate as a response to collisions. However, the concepts and implementation described generalize to support rotational collision responses. This project is designed to help you understand the basic concepts of impulse based collision resolution with axis-aligned shapes.

## Formulating the Impulse Method

You will formulate the solution for the Impulse Method by first reviewing how a circle can bounce off of a wall and other circles in a perfect world. This will subsequently be used to derive an approximation for an appropriate collision response. Note that the following discussion focuses on deriving the formulation for the Impulse Method and does not attempt to present a review on pure Newtonian Mechanics. Here is a brief review of some of the relevant terms.

* Mass: is the amount of matter in an object, or how dense an object is.
* Force: is any interaction or energy imparted on an object that will change the motion of that object.
* Relative Velocity: is the difference in velocity between two travel shapes.
* Coefficient of Restitution: the ratio of relative velocity after and before a collision. This is a measure of how much of the kinetic energy remains for the object to rebound from one another, or, bounciness.
* Coefficient of Friction: a number that describes the ratio of the force of friction between two bodies. In your very simplistic implementation, friction is applied directly to slow down linear motion or rotation.
* Impulse: accumulated force over time that can cause a change in the velocity. For example, resulting from a collision.

### Decomposing the Velocity in a Collision

Figure 4-5 illustrates a circle A in three different stages. At stage 1 the circle is traveling at velocity towards the wall on its right. At stage 2 the circle is colliding with the wall. At stage 3 the circle has been reflected and is traveling away from the wall with velocity .

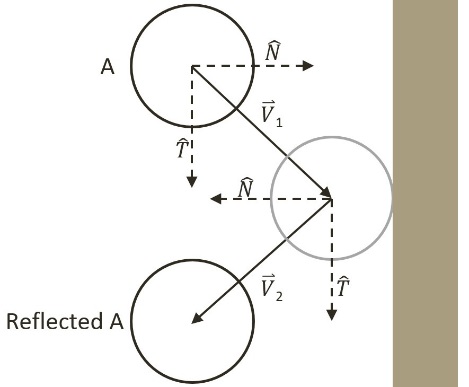


Figure 4-5 Collision Between a Circle and a Wall in a Perfect World.

Mathematically, this collision and its response can be described by decomposing the initial velocity, , into the components that are parallel, or tangent , and perpendicular, or normal , to the colliding wall. As seen in the following equation.

In a perfect world with no friction and no loss of kinetic energy, after the collision, the component along the tangent direction will not be affect while the normal component will be simply reversed. In this way, the reflected vector can be expressed as a linear combination of normal and tangent components of as followed.

Notice the negative sign in front of the component. You can see in Figure 4-5, that the component for vector points in the opposite direction of that of as a result of the collision. Notice also that the tangent component, , is still pointing in the same direction since it is parallel to the of the wall and is unaffected by the collision. This demonstrates a vector reflection.

### Relative Velocity of Colliding Shapes

This decomposition of vectors into the normal and tangent directions of the collision also applies in the general cases when the colliding shapes are both in motion. For example Figure 4-6 illustrates two traveling circle shapes, A and B, colliding.



Figure 4-6 Collision Between Two Circles

In the case of Figure 4-6, before the collision, shape A is traveling with velocity while shape B with velocity . The normal direction of the collision, , is defined to be the vector between the two circle centers and the tangent direction of the collision, , is the vector that is tangential to both of the circles at the point of collision. To resolve this collision, the velocities for shape A and B after the collision, and , must be computed.

The relative velocity between shapes A and B is defined as follows.

The collision vector decomposition can now be applied to the normal direction of the relative velocity where the relative velocity after the collision is .

* ***(1)***

The coefficient of restitution, , models the real-world situation where some kinetic energy is changed to some other form of energy during the collision. Notice that all variables on the right-hand-side of Equation (1) are defined, as they are known at the time of collision, and that the normal component of the relative velocity after the collision of shapes A and B, , is also defined. It is important to remember that,

* .

You are now ready to approximate and , the velocities of the colliding shapes after the collision.

### Approximating the Impulse Response

Accurately describing a collision involves complex considerations including factors like energy changing form, or frictions resulting from different material properties, etc. Without considering these advanced issues, a simplistic description of a collision that occurs on a shape is, a constant mass object changing its velocity from to after contact with another object. Conveniently, this is the definition of an impulse, as can be seen in the following.

Or, when solving for ,

Take a step back from the math and think about what this formula states. It makes intuitive sense. It states that the change in velocity is inversely proportional to the mass of a shape. In other words, the more mass a shape has, the less its velocity will change after a collision. The Impulse Method implements this observation, and for the normal component, it defines the velocities after a collision for shapes A and B, and , to be as followed. In this case, , and are the masses of Shapes A and B.

Subtracting the above two equations computes the normal component of relative velocity.

Recall that, is simply , and that, is , this equation simplifies to the following.

Substituting Equation (1) to the left-hand-side and remembering that and are perpendicular so that the following equation can be derived.

Collecting terms, and solving the formula for , the impulse in the normal direction gives you the following.

* ***(2)***

Finally, the impulse in the tangent direction, , can be derived in a similar manner the results of which follow.

* ***(3)***

The coefficient of friction, , is a simplistic approximation of friction.

## The Steps for Resolving Collisions

You are now ready to modify the resolveCollision function in the Physics.js file to implement the collision resolution between two colliding shapes. The resolution procedure requires access to the two RigidShape objects and the corresponding collision information. The following are the detailed steps involved:

* **Step A**: make sure at least one of the colliding shapes is not static (an inverse mass that is not equal to 0).
* **Step B**: invoke the positional correction function to snap the shapes apart by a percentage of the interpenetration depth. Recall that in your implementation, the colliding shapes will be pushed apart by a default of 80% of the interpenetration depth.
* **Step C**: compute the relative velocity between the two shapes. As presented in the derivation, the relative velocity is essential for computing the impulse in the normal and tangent directions.
* **Step D**: compute the component of the relative velocity that is in the collision normal direction. This component indicates how rapidly the two shapes are moving toward or away from each other. A positive value indicates that the shapes are moving away from each other and impulse response will not be necessary.
* **Step E**: compute the impulse in the normal direction based on results from previous step, restitution (bounciness), and the masses of the colliding shapes.
* **Step F**: compute the impulse in the tangent direction.
* **Step G**: apply impulses to modify the normal and tangent components of the shapes’ velocities to simulate the reflection of both shapes after the collision as well as friction.

The normal and tangent components of the impulse accomplish distinct purposes in simulating the results of a collision. The normal component simulates the bounciness of shapes, while the tangent component handles the friction. As illustrated in Figure 4-7, when a ball is tossed from the left towards the right, its initial spinning direction will determine the motion after the collision with the floor. On the left of Figure 4-7 the ball has an initial counter-clockwise spin while the ball on the right of the figure has an initial clockwise spin. At the point of collision with the floor, the tangent impulse component, modified by the respective friction force, will either reduce or increase the right-ward linear velocity of the ball depending on its initial spinning direction. This particular functionality will be implemented in the following section on rotational collision response. However, take note that regardless of the objects rotation upon collision the heights of the ball, after the collision, are equal to each other. This is a result of friction only affecting the tangent impulse component while the restitution affects the normal impulse component.

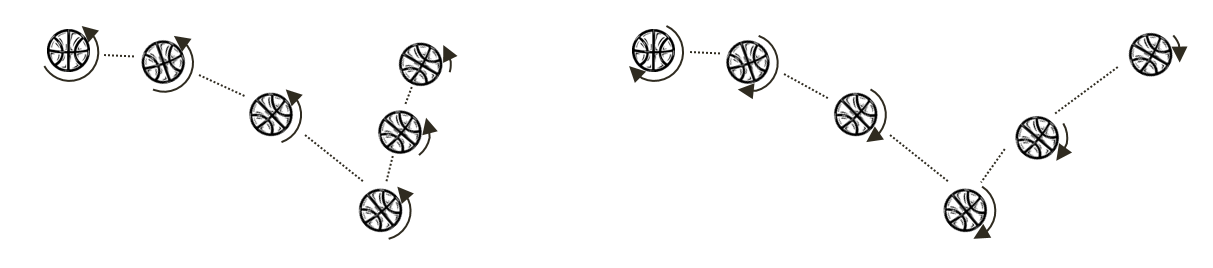


Figure 4-7. Tangent Component Impulse and Friction.

## The Collision Resolution Project

This project will guide you through implementing the outlined steps to create a function that resolves the collision between axis-aligned shapes using the Impulse Method. You can see an example of this project running in Figure 9-X2. The source code to this project is defined in chapter9/9.7.collision\_resolution.

Figure 9-X1. Running the Collision Position Correction project

The controls of the project are as follows, for both scenes:

* **This and that**
* **This and that**.

The goals of the project are as follows:

* To To understand the details of Impulse Method computations
* To build a system that resolves the collision between colliding shapes

### Modify the Physics Engine

To properly support collision resolution, you only need to modify the physics.js file to implement the previously outlined steps.

1. Open the Physics.js file and go to the resolveCollision function.
2. After positional correction, you will begin the implementation by computing the collision normal, the relative velocity, coefficient of restitution and friction of the colliding shapes.

var resolveCollision = function (s1, s2, collisionInfo) {

if ((s1.mInvMass === 0) && (s2.mInvMass === 0))

return;

// correct positions

if (gEngine.Physics.mPositionalCorrectionFlag)

positionalCorrection(s1, s2, collisionInfo);

var n = collisionInfo.getNormal();

var v1 = s1.mVelocity;

var v2 = s2.mVelocity;

var relativeVelocity = v2.subtract(v1);

// Relative velocity in normal direction

var rVelocityInNormal = relativeVelocity.dot(n);

// if objects moving apart ignore

if (rVelocityInNormal > 0)

return;

// compute and apply response impulses for each object

var newRestituion = Math.min(s1.mRestitution, s2.mRestitution);

var newFriction = Math.min(s1.mFriction, s2.mFriction);

//… details in the following steps

};

1. Compute the impulse in the direction of the collision normal based on Equation (2).

//...continue from the previous step

// Calc impulse scalar

var jN = -(1 + newRestituion) \* rVelocityInNormal;

jN = jN / (s1.mInvMass + s2.mInvMass);

//… details in the next step

1. Apply the impulse to the velocities of the colliding shapes.

//...continue from the previous step

//impulse is in direction of normal ( from s1 to s2)

var impulse = n.scale(jN);

// impulse = F dt = m \* △v

// △v = impulse / m

s1.mVelocity = s1.mVelocity.subtract(impulse.scale(s1.mInvMass));

s2.mVelocity = s2.mVelocity.add(impulse.scale(s2.mInvMass));

//… details in the next step

1. Compute the direction that is tangent to the collision normal.

//... continue from the previous step

var tangent = relativeVelocity.subtract(n.scale(relativeVelocity.dot(n)));

// relativeVelocity.dot(tangent) should less than 0

tangent = tangent.normalize().scale(-1);

//… details in the next step

1. Compute the impulse, jT, in the direction that is tangent to the collision normal based on Equation (3), and apply the impulse to the velocities of the colliding shapes.

//...continue from the previous step

var jT = -(1 + newRestituion) \* relativeVelocity.dot(tangent) \* newFriction;

jT = jT / (s1.mInvMass + s2.mInvMass);

// friction should less than force in normal direction

if (jT > jN) jT = jN;

//impulse is from s1 to s2 (in opposite direction of velocity)

impulse = tangent.scale(jT);

s1.mVelocity = s1.mVelocity.subtract(impulse.scale(s1.mInvMass));

s2.mVelocity = s2.mVelocity.add(impulse.scale(s2.mInvMass));

### Defining an Initial Rectangle in Mygame.js

You need to modify Mygame.js file to define an initial rectangular RigidShape object for testing purposes. Edit Mygame.js and add the following code to define a stationary rectangle with infinite mass.

function MyGame() {

//...identical to previous project

var r2 = new Rectangle(new Vec2(200, 400), 400, 20, 0, 1, 0);

//...identical to previous project

}

### Observations

Run You should test your implementation in two ways. First, ensure that moving shapes collide and behave naturally. Second, ensure the collision resolution system is stable when there are many shapes that are in close proximity. You also can test the collision resolution between regular shapes and shapes with infinite mass.

Notice that the scene now has a platform like shape. This is a shape with infinite mass that can be tested for collision resolution with other regular moving shapes. Now make sure movement is switched on with the comma ( , ) key and create several rectangle and circle shapes with the ‘F’ and ‘G’ keys. Notice that the shapes fall gradually to the floor and their motions stop with a slight rebound. This is a clear indication that the base case for Euler Integration, collision detection, and resolution all are operating as expected. Press the ‘H’ key to excite all shapes. Notice the wandering shapes interact properly with the platforms and the walls of the game world with soft bounces and no apparent interpenetrations. In addition, pay attention to the apparent transfer of energy during collisions. Try adjusting the shape attributes, for example, the mass, and observe what happens when two shapes with very different masses collide. Notice that the shape with more mass does not change its trajectory much after the collision. Lastly, notice that the shapes do not rotate as a result of collision. That is because your current implementation only considers the linear velocity of the shapes. In the next project you will improve the resolution function to consider angular velocity changes as a result of collisions.

The stability of the system can be tested by increasing the number of shapes in the scene. The relaxation loop count of 15, continuously pushes interpenetrating shapes apart by 80% of the interpenetration depth during each iteration in addition to the impulse correction that is calculated. For example, you can switch off movement and positional corrections with the “,” and ‘M’ keys and create multiple, e.g., 10 to 20, overlapping shapes at the exact same position. Now enable position correction with the M key and notice that after a short pause the shapes will appear again with no interpenetrations.