Simulating the Rigid World

# Collision Resolution

With a proper 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 begin by examining collision responses without rotations. The complications associated with angular impulse resolutions will be examined in the next section, after the mechanics behind simple 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 can be generalized in a straightforward manner to support rotational collision responses. This project is designed to help you understand the basic concepts of impulse based collision resolutions.

## 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 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 measurement of how much of kinetic energy remains after an the object bounces off 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.

### Components of Velocity in a Collision

Figure 9-25 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 and at stage 3 the circle has been reflected and is traveling away from the wall with velocity .



Figure 9-25 Collision Between a Circle and a Wall in a Perfect World

Mathematically, this collision and the 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 simply be 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 9-25, that the component for vector points in the opposite direction of that of as a result of the collision. Additionally, notice 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 analysis applies to a vector reflecting off a stationary surface.

### Relative Velocity of Colliding Shapes

The decomposition of vectors into the normal and tangent directions of the collision can also be applied to the general case of when both of the colliding shapes are in motion. For example, Figure 9-26 illustrates two traveling circle shapes, A and B, coming into a collision.



Figure 9-26 Collision Between Two Traveling Circles

In the case of Figure 9-26, 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 post-collision velocities are determined based on the relative velocity between the two shapes. The relative velocity between shapes A and B is defined as follows.

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

* ***(1)***
* ***(2)***

The coefficients of restitution, , and friction, , model the real-world situation where some kinetic energy is changed to some other form of energy during the collision. The negative sign of Equation (1) signifies that after the collision objects will travel in the direction that is opposite to the initial collision normal direction. Equation (2) says, after the collision, friction will scale back the tangent direction where objects will continue to travel in the same tangent direction, only with a lower velocity. Notice that all variables on the right-hand-side of Equations (1) and (2) are defined, as they are known at the time of collision. It is important to remember that,

* .

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

**Note** The restitution coefficient, e, describes bounciness or, the proportion of the velocity that is retained after a collision. A restitution value of 1.0 would mean that velocities will be the same from before and after a collision. In contrast, intuitively, friction is typically associated with the proportion lost, or the slow down after a collision. For example, a friction coefficient of 1.0 would mean perfectly frictional where a velocity of zero will result from a collision. For consistency of the formulae, the coefficient f in Equation (2) is actually 1 minus the intuitive friction coefficient.

### The Impulse

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 contacting with another object. Conveniently, this is the definition of an impulse, as can be seen in the following.

Or, when solving for ,

* ***(3)***

Remember that the same impulse also causes the velocity change in object B, only in the opposite direction,

Or, when solving for ,

* ***(4)***

Take a step back from the math and think about what this formula states. It makes intuitive sense. The equation 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.

Recall that Equations (1) and (2) describe the relative velocity after collision according to the collision normal and tangent directions independently. Now, let the components of the impulse vector in the collision normal and tangent directions be and ,

* =

#### Normal Component of an Impulse

The normal component of the impulse can be analyzed by performing a dot product with the vector on both side of Equations (3) and (4),

Subtracting the above two equations,

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

Substituting Equation (1) to the left-hand-side to derive the following equation,

Collecting terms and solving for , the impulse in the normal direction, resulting in the following,

* ***(5)***

#### Tangent Component of an Impulse

The tangent component of the impulse can be analyzed by performing a dot product with the vector on both side of Equations (3) and (4),

Following the similar steps as in the case for the normal component, subtracting the equations, and recognizing is and is , to derive the following equation,

Now, substituting Equation (2) to the left-hand-side,

Finally, collect terms and solve for , impulse in the tangent direction,

* ***(6)***

## The Collision Resolution Project

This project will guide you through resolving a collision by calculating the impulse and updating the velocities of the colliding objects. You can see an example of this project running in Figure 9-27. The source code to this project is defined in chapter9/9.7.collision\_resolution.



Figure 9-27. Running the Collision Resolution project

The controls of the project are identical to the previous project with additional controls for restitution and friction coefficients:

* **Behavior control:**

P key: Toggle penetration resolution for all objects

**V key**: Toggle motion of all objects

**H key**: Inject random velocity to all objects

G key: Randomly create a new rigid circle or rectangle

* **Draw control**

C key: Toggle the drawing of all CollisionInfo

T key: Toggle textures on all objects

R key: Toggle the drawing of RigidShape

B key: Toggle the drawing of the bound on each RigidShape

* **Object control:**

Left/right-arrow key: Sequence through and select an object

WASD keys: Move the selected object

Z/X key: Rotate the selected object

Y/U key: Increase/decrease RigidShape size of the selected object, this does not change the size of corresponding Renderable object

**Up/down-arrow key + M/N/F**: Increase/decrease the mass/restitution/friction of the selected object

The goals of the project are as follows:

* To understand the details of the Impulse Method
* To implement the Impulse Method in resolving collisions

### Updating the Physics Component

To properly support collision resolution, you only need to focus on the physics component and modify the physics.js file in the src/engine/components folder.

1. Edit physics.js and define the resolveCollision() function to resolve the collision between RigidShape objects, a and b, with collision information recorded in the collisionInfo object.

function resolveCollision(b, a, collisionInfo) {

let n = collisionInfo.getNormal();

// Step A: Compute relative velocity

let va = a.getVelocity();

let vb = b.getVelocity();

let relativeVelocity = [0, 0];

vec2.subtract(relativeVelocity, va, vb);

// Step B: Determine relative velocity in normal direction

let rVelocityInNormal = vec2.dot(relativeVelocity, n);

//if objects moving apart ignore

if (rVelocityInNormal > 0) {

return;

}

// Step C: Compute collision tangent direction

let tangent = [0, 0];

vec2.scale(tangent, n, rVelocityInNormal);

vec2.subtract(tangent, tangent, relativeVelocity);

vec2.normalize(tangent, tangent);

// Relative velocity in tangent direction

let rVelocityInTangent = vec2.dot(relativeVelocity, tangent);

// Step D: Compute and apply response impulses for each object

let newRestituion = (a.getRestitution() + b.getRestitution()) \* 0.5;

let newFriction = 1 - ((a.getFriction() + b.getFriction()) \* 0.5);

// Step E: Impulse in the normal and tangent directions

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

jN = jN / (a.getInvMass() + b.getInvMass());

let jT = (newFriction - 1) \* rVelocityInTangent;

jT = jT / (a.getInvMass() + b.getInvMass());

// STEP F: Update velocity in both normal and tangent directions

vec2.scaleAndAdd(va, va, n, (jN \* a.getInvMass()));

vec2.scaleAndAdd(va, va, tangent, (jT \* a.getInvMass()));

vec2.scaleAndAdd(vb, vb, n, -(jN \* b.getInvMass()));

vec2.scaleAndAdd(vb, vb, tangent, -(jT \* b.getInvMass()));

}

The listed code follows the solution derivation closely.

1. Steps A and B: compute the relative velocity and its normal component. When this normal component is positive, it signifies the two objects are moving away from each other and thus collision resolution is not necessary.
2. Step C: computes the collision tangent direction and the tangent component of the relative velocity.
3. Step D: use the averages of the coefficients for impulse derivation. Notice the subtraction by one when computing the newFriction for maintaining consistency with Equation (2).
4. Step E: follows the listed Equations (5) and (6) to compute the normal and tangent components of the impulse.
5. Step F: solves for the resulting velocities by following Equations (3) and (4).
6. Edit collideShape() to invoke the resolveCollision() function when a collision is detected and position corrected.

function collideShape(s1, s2, infoSet = null) {

let hasCollision = false;

… identical to previous code …

positionalCorrection(s1, s2, mCInfo);

resolveCollision(s1, s2, mCInfo);

… identical to previous code …

};

### Updating MyGame for Testing Collision Resolution

The modifications to the MyGame class are trivial, mainly to toggle both motion and positional correction to be active by default. Additionally, initial random rotations of the created RigidShape objects are disabled because at this point collision response does not support rotation. As always, you can refer to the source code files in the src/my\_game folder for implementation details.

### Observations

You should test your implementation in three ways. First, ensure that moving shapes collide and behave naturally. Second, try changing the physical properties of the objects. Third, observe the collision resolution between shapes that are in motion and shapes that are stationary with infinite mass (the surrounding walls and stationary platforms). Remember that with only linear velocities considerations, rotations will not result from collisions.

Now, run the project and notice that the shapes fall gradually to the platforms and floor with their motions coming to a halt after slight rebounds. This is a clear indication that the base case for Euler Integration, collision detection, positional correction, and resolution all are operating as expected. Press the ‘H’ key to excite all shapes and the C key to display the collision information. Notice the wandering shapes and the walls/platforms interact properly with soft bounces and no apparent interpenetrations.

Use the left/right-arrow to select an object and adjust its restitution/friction coefficients with the N/F and up/down-arrow keys. For example, adjust the restitution to 1 and friction to 0. Now inject velocity with the H key. Notice how the object seems extra bouncy and, with a friction coefficient of 0, seems to skid along platforms/floors. You can try different coefficient settings and observe corresponding bouncy and slipperiness.

The stability of the system can be tested by increasing the number of shapes in the scene with the G key. The relaxation loop count of 15, continuously and incrementally pushes interpenetrating shapes apart during each iteration. For example, you can toggle off movement and positional corrections with the V and P keys and create multiple, e.g., 10 to 20, overlapping shapes. Now toggle on motion and positional corrections and observe a properly functioning system.

In the next project you will improve the resolution function to consider angular velocity changes as a result of collisions.