6CC529 Game Behaviour
Collision Response

Dr Minsi Chen

1 Introduction

In the notes on collision response, we discussed the methods for handling the collision of two rigid bodies. Both conservation of momentum and energy provide a useful foundation for resolving the motion of two colliding bodies. In this tutorial, you will be tasked to implement a "collision responder" using the impulse based method.

Before you proceed, download the source code from Brightspace. You can find the skeleton of the class representing an impulse based collision responder in `ImpulseCollisoinResponder.cs`. This is where you implement the collision response. Additionally, the source file `WorldPart1.cs` contains a method for detecting collision based on bounding circles. You can add the method to `class World` for testing your implementation. The executable inside the source package demonstrates the effect of impulse based collision response.

Perfectly Elastic Collision A perfectly elastic collision is governed by the following rule which states the conservation of both momentum and kinetic energy.

$$\begin{aligned} m_1\vec{v_1} + m_2\vec{v_2} &= m_1\vec{v_1'} + m_2\vec{v_2'} \\ \frac{1}{2}m_1|\vec{v_1}|^2 + \frac{1}{2}m_2|\vec{v_2}|^2 &= \frac{1}{2}m_1|\vec{v_1'}|^2 + \frac{1}{2}m_2|\vec{v_2'}|^2 \end{aligned} \tag{1}$$

Perfectly Inelastic Collision In contrast to a perfectly elastic collision, a perfectly inelastic collision results in two colliding bodies travelling in the same direction at the same velocity. For example, a bullet lodged into a body will carry the body into motion. Some car crash scenarios can also be simplified to this type of collision.

Perfectly inelastic collision requires only the conservation of momentum, thus it is relatively simple to resolve by direct application of the following equation.

$$\begin{aligned} m_1\vec{v_1} + m_2\vec{v_2} &= (m_1 + m_2)\vec{v'} \\ \vec{v'} &= \frac{m_1\vec{v_1} + m_2\vec{v_2}}{m_1 + m_2} \end{aligned} \tag{2}$$

See Listing 1 for an example implementation of inelastic collision.

Listing 1: Routine for resolving a perfectly inelastic collision.

```
void perfect_inelastic_collision_response ( RigidBody& body1, RigidBody& body2 )
{
    Vector3 velocity;
    float inv_mass_sum;

    inv_mass_sum = 1.0f/(body1.m_mass + body2.m_mass);

    velocity = ( body1.m_velocity*body1.m_mass + body2.m_velocity*body2.m_mass )*inv_mass_sum;

    //Update rigid bodies
    body1.m_velocity = velocity;
    body2.m_velocity = velocity;
}
```

This remainder of this notes will discuss some practical aspects of implementing a collision response system in a physics engine.

2 Collision Response

2.1 A Brief Word on Collision Detection

Before we detail the resolutions of collision response, it is worthwhile describing some values retuned by a collision detection routine. In general, the following quantities are required in order for us to resolve the motion after collision:

Contact Points This is a list of points where two objects intersect. We also need this contact points to work out torques exerted on the body if angular motion is to be modelled.

Contact Normals This is a list normals of collided surfaces. For a 3D polygonal mesh, we can retrieve these from each collided polygon. In 2D, this is the vector perpendicular to the edge where collision is detected. Normals are needed to determine the direction of motion after collision.

Penetration Because the physical motion in games in calculated at a discrete time interval, object penetration is inevitable. In order to address this problem, we need to find out the depth penetration which is measured along the contact normals.

You can find the class `CollisionData` which encapsulates the above mentioned quantities in `CollisionData.cs`.

Once we obtain these quantities, the remaining part is to determine the motion of the colliding bodies. We will be discussing the following methods for performing this task.

Projection Method This is a non-physically based method that simply prevents two objects from overlapping.

Impulse Method This is a method based on the momentum of colliding objects, therefore it resolves the velocity of each body after collision.

2.2 Projection Method

The projection method does not alter the motion of colliding bodies. The position of the colliding objects are calculated based on the depth of penetration. A common approach is to compute a displacement along the colliding normal. Each involving body is then displaced in opposite direction along the vector.

Because no physical quantities are resolved, (e.g. no change to velocity), the colliding bodies may resume their motion after the separation. This may cause oscillation and eventually the objects become stuck to each others. See Listing 2 for an example implementation of the projection method.

Listing 2: Routine for resolving a collision using the projection method.

```
void projection_collision_response ( RigidBody& body1, RigidBody& body2, const vector3& intersection_point,
        const Vector3& normal, const Vector3& query_point )
{
    //Note: use normal and intersection_point to calculate
    //the equation of the colliding plane
    //i.e. ax+by+cz+d=0
    //
    float d;
    float penetration_depth;
    Vector3 displacement;

    d = -(normal.dot(intersection_point));

    penetration_depth = query_point.dot(normal) + d;

    //Work out the displacement required to resolve the penetration
    displacement = -(normal * penetration_depth);

    //Update the body1
    body1.m_centre_of_mass += displacement;
    body2.m_centre_of_mass -= displacement;
}
```

2.3 Impulse Method

In both elastic and inelastic collision, their formulations deal directly with the velocity of two colliding bodies. Although it is relatively simpler to work out the velocity for the perfectly inelastic case, the same cannot be said for the elastic one. The impulse method introduced in this section aims to solve this type

of response by using a physical quantity known as *impulses* J. Its unit is Newton-seconds, therefore it is defined as:

$$J = F\Delta t \tag{3}$$

Since $F = ma$ and $a = \frac{\Delta v}{\Delta t}$, J can also be defined in terms of velocity and mass, i.e.

$$J = m\Delta v \tag{4}$$

Impulses are applied to each body in such way that the momentum can be transferred between two colliding bodies. Since the impulse can affect both linear and angular momentum, we will therefore discuss them separately.

2.3.1 Linear Impulses

For linear motion, the impulse method can be formulated using the following equation:

$$\begin{aligned} m_1\vec{v_1'} &= m_1\vec{v_1} + J\vec{n} \\ m_2\vec{v_2'} &= m_2\vec{v_2} - J\vec{n} \end{aligned} \tag{5}$$

J is scalar value denoting the impulse and $\vec{n}$ is the collision normal.

Equation 5 effectively adds two additional momenta of equal magnitude and opposite direction to each body. Doing so does not invalidate the rule of momentum conservation. We can verify this by adding the before and after momenta of two bodies. The newly injected impact terms simply get cancelled out.

The velocity of each body after collision can be calculated by:

$$\begin{aligned} \vec{v_1'} &= \vec{v_1} + \frac{J}{m_1}\vec{n} \\ \vec{v_2'} &= \vec{v_2} - \frac{J}{m_2}\vec{n} \end{aligned} \tag{6}$$

This is a much simpler solution as we do not have to consider the second constraint stipulated by the conservation of kinetic energy.

Before we calculate J, we should be aware that the impulse generated here is a result of two colliding bodies travelling at velocity $\vec{v_1}$ and $\vec{v_2}$. The impulse we are interested in is due to the relative velocity $\vec{v_{12}}$ of this two objects, i.e.

$$\vec{v_{12}} = \vec{v_1} - \vec{v_2} \tag{7}$$

The relative velocity is also the same direction as the collision normal $\vec{n}$ and its magnitude is

$$|\vec{v_{12}}| = \vec{v_{12}} \cdot \vec{n} \tag{8}$$

The relative velocity will be in the opposite direct after the collision, as described by:

$$\vec{v_{12}'} = -\epsilon|\vec{v_{12}}|\vec{n} \tag{9}$$

In the above equation, we introduced a scalar term $\epsilon \in [0, 1]$ to emulate the elasticity of the collision. When $\epsilon = 1$, the collision is equivalent to a perfectly elastic collision.

Now by combing all the equation defined in this section, the value of J can be solved by:

$$J = \frac{-(1+\epsilon)\vec{v_{12}} \cdot \vec{n}}{|\vec{n}|^2(\frac{1}{m_1} + \frac{1}{m_2})} \tag{10}$$

Once you have implemented the impulse method, make sure you experiment with the value of ϵ. It is an effective way of adding some interesting variations to the dynamics of your environment.

2.3.2 Angular Impulses

Angular impulses need to be considered when two colliding bodies also undergo angular motions (rotation). Please note, the rigid bodies in `SimpleDynamics` currently do not spin since torques are not calculated. However, adding torques to rigid bodies will certainly result in more believable physical behaviour. For implementing torques on rigid bodies, you should refer to the pendulum exercise from the last tutorial.

The impulses for the angular component are largely similar to the linear case. The angular momenta of two colliding bodies are defined by:

$$\begin{aligned} \vec{L_1} &= \mathbf{I_1}\vec{\omega} \\ \vec{L_2} &= \mathbf{I_2}\vec{\omega} \end{aligned} \tag{11}$$

Recall angular momentum is related to the inertia tensor and the angular velocity of an object.

Additionally, when two rotating objects colliding each other, the impulse is in the direction tangential to the colliding plane. Therefore, we need to find out the tangential velocity $\vec{v_t}$ from a rotating body and the point $\vec{p}$ at which the collision occurs. $\vec{v_t}$ can be solved by:

$$\vec{v_t} = \vec{\omega} \times (\vec{p} - \vec{c}) \tag{12}$$

where $\vec{c}$ is the centre of mass of the object.

If two objects travelling at velocity $\vec{v_1}$ and $\vec{v_2}$ and rotating at $\vec{\omega_1}$ and $\vec{\omega_2}$, their respective velocity at collision point $\vec{p}$ is given by

$$\begin{aligned} \vec{v_{p1}} &= \vec{v_1} + \vec{\omega_1} \times (\vec{p} - \vec{c_1}) \\ \vec{v_{p2}} &= \vec{v_2} + \vec{\omega_2} \times (\vec{p} - \vec{c_2}) \end{aligned} \tag{13}$$

Where $\vec{c_1}$ and $\vec{c_2}$ are their respective center of mass. The relative velocity $\vec{v_{12}}$ and $\vec{v'_{12}}$ used in the linear impulse case becomes:

$$\begin{aligned} \vec{v_{12}} &= \vec{v_{p1}} - \vec{v_{p2}} \\ \vec{v'_{12}} &= -\epsilon|\vec{v_{12}}|\vec{n} \end{aligned} \tag{14}$$

The impulse J is then added to the angular momentum analogously to the linear case.

$$\begin{aligned} \mathbf{I_1}\vec{\omega'_1} &= \mathbf{I_1}\vec{\omega_1} + (\vec{p} - \vec{c_1}) \times J\vec{n} \\ \mathbf{I_2}\vec{\omega'_2} &= \mathbf{I_2}\vec{\omega_2} - (\vec{p} - \vec{c_2}) \times J\vec{n} \end{aligned} \tag{15}$$

Angular momentum is also conserved after collision.

The impulse J combining both linear and angular motion is given by

$$\begin{aligned} J &= \frac{-(1+\epsilon)\vec{v_{12}} \cdot \vec{n}}{|\vec{n}|^2(\frac{1}{m_1} + \frac{1}{m_2}) + J_\alpha} \\ J_\alpha &= ((I_1^{-1}(\vec{r_1} \times \vec{n})) \times \vec{r_1} + (I_2^{-1}(\vec{r_2} \times \vec{n})) \times \vec{r_2}) \cdot \vec{n} \\ \vec{r_1} &= \vec{p} - \vec{c_1} \\ \vec{r_2} &= \vec{p} - \vec{c_2} \end{aligned} \tag{16}$$

The angular velocity after collision ω'_1 and ω'_2 can be obtained by plugging the result of Equation 16 into Equation 15. You may have noticed the similarity to the linear impulse in the previous section.

3 Implementation

The following listing shows an indicative implementation of the impulse response method in C++. Please note directly copying the code to Visual Studio will not work. You should first try to understand how the statements are related to the theory of the impulse response method.

Listing 3: Routine for resolving a collision using the impluse method.

```
void impulse_collision_response( RigidBody& body1, RigidBody& body2,
        const Vector3& intersection_point, const Vector3& normal )
{
    Vector3 r1, r2, v1, v2, dv;
    Vector3 r1xn, r2xn;
    float normal_impulse, tangent_impulse;
    float normal_div, tagent_div;
    float normal_length_squared;
    float epsilon = 0.0f;

    r1 = intersection_point - body1.m_centre_of_mass;
    r2 = intersection_point - body2.m_centre_of_mass;
```

```
    v1 = body1.m_velocity + body1.m_angular_velocity.cross(r1);
    v2 = body2.m_velocity + body2.m_angular_velocity.cross(r2);

    normal_length_squared = normal.dot(normal);

    dv = v1 − v2;

    if ( dv ∗ normal < 0.0f ) return;

    //From this onward we need to resolve both normal and tangential impulses
    //
    //1. Normal Impulse
    r1xn = r1.cross( normal );
    r2xn = r2.cross( normal );

    normal_div = normal_length_squared∗(body1.m_inverted_mass + body2.m_inverted_mass) +
            normal.dot(
                (body1.m_inertia_tensor_inverted ∗ r1xn).cross(r1) +
                (body2.m_inertia_tensor_inverted ∗ r2xn).cross(r2)
                );
    normal_impulse = −(1.0+epsilon)∗(dv.dot(normal))/normal_div;

    body1.m_velocity += normal ∗ normal_impulse ∗ body1.m_inverted_mass;
    body2.m_velocity −= normal ∗ normal_impulse ∗ body1.m_inverted_mass;

    body1.m_angular_velocity += body1.m_inertia_tensor_inverted ∗ (r1.cross(normal ∗ normal_impulse));
    body2.m_angular_velocity −= body2.m_inertia_tensor_inverted ∗ (r2.cross(normal ∗ normal_impulse));

    //2. Tangential Impulse
    //NOTE: To complete the computation we also need to calculate the impulse in the tangent direction
    //The formula is exactly same as the normal direction
    //This will be left as an exercise
}
```