This tutorial explains the mathematical methods used to implement rigid body physics during object interaction in 3D real time computer graphics. This tutorial will not discuss the math involved from a calculus or linear algebra standpoint, but will focus on the variables in the equations themselves how they operate to make a physical reaction happen. Before starting any math problem it is good to know what variables and values are known when applying the math to solve the problem. In real time computer graphics everything happening on screen should be governed by a time variable that knows the time of each update/frame of the environment. This means the changes happening on screen are synced to this time variable and cannot stray away from the values it represents. Since physics is the topic of this tutorial, the movement and orientation (rotation) changes of an object are governed by this time variable. You can initialize and know the values of the objects position and orientation when the objects are created. So far you can assume you know the time of the frame and the position and orientation the object has.

The Simple equations representing the values of the objects position and orientation at any given frame are as follows:

Current Frame Position = Previous Frame Position + Current Frame Velocity * (Current Frames Time – Previous Frames Time)

and

Current Frame Orientation = Previous Frame Orientation + Current Frame Angular Velocity * (Current Frames Time – Previous Frames Time)

These values must be calculated before every frame is displayed on the computer screen. Although you will notice that not only time is determining these changes now. You now have velocities that represent the change is position and orientation between frames when multiplied by the change in time between frames. These values can be pre determined or calculated based on what you want to do. Whether the object starts its life moving or not? You can simply initialize these velocities when the object is created. Any outside forces or interactions the object has will now decide how these velocities change.

Adding a pre determined outside force is easier then determining the changes in velocities based on object interactions and can explained in a few simple equations:

Current Frame Acceleration = Current Frame Outside Force / Mass of Frame Object

Current Frame Velocity = Previous Frame Velocity + Current Frame Acceleration * (Current Frames Time – Previous Frames Time)

and

Current Frame Angular Acceleration = Current Frame Torque / Current Frame Moment of Inertia

Current Frame Angular Velocity = Previous Frame Angular Velocity + Current Frame Angular Acceleration * (Current Frames Time – Previous Frames Time)

The first new variable you will notice is the mass of the object which you can choose to be anything but 0. The mass in laymen's terms is how compact is the material that makes up the object in a certain volume represented in a number. One example of this is 2 balloons the same size, one filled with air and the other with water. The one with water has a greater mass because the material it's made up of is closer packed together. Water is denser then air. Although if the balloon with air was 1000 times the size of the one with water it might have more mass because there's more material inside the air balloon. Respectively a smaller balloon filled with lead might have more mass than the huge air balloon. Since you choose the size and the density of the object and these values will only determine the mass of our rigid body object. You can skip computation and just give the object a mass.

Basically what the first equation does is it scales the Force by the inverse of the mass. So if the same force was applied to two objects, one with a larger mass than the other. The one with the larger mass will receive less acceleration than the one with smaller mass.

Then the new velocity is calculated the same way as the position was except in respect to the acceleration. In fact instead, unless you really needed to know the force you were applying to the object you can just give the object an acceleration to represent these forces.

The angular velocity is calculated almost the same way except for the variables involved, the objects torque is the angular force and the objects moment of inertia is the angular mass. The inertia is derived from the mass that we already have. Depending on the shape of the object rotating, the distribution of mass in the object and the axis that it is rotating by this value can sometimes be complicated to compute which is bad in a real time environment. The inertia for symmetric 3D shapes with an even distribution of mass simplifies this and if you can fit these types of shapes to almost fit strangely shaped objects. The resulting inertia value will be a good enough approximation.

The solid sphere is the easiest shape of all for computing inertia. Neither what axis of rotation nor the current orientation of the sphere will cause this value to change. So after the mass of the sphere is calculated. As long as the mass and the size stay the same you can calculate the inertia once and never have to calculate it again.

The equation for a sphere's inertia:

2/5 * Sphere Mass * Sphere Radius * Sphere Radius

If multiple forces are causing changes they can be added together to get a resulting net force or acceleration that will still get you the single change in velocity on the object during that frame.

Now that I have discussed how to apply outside forces on an object down to its change in position and orientation I will next describe how to determine responses to

interactions with other objects in the environment. In Short we now know how to computer and know the following values on an object during a frame of our real time 3D environment. These knowns will help us in determining the response of an interaction between objects.

Knowns
Change in time between frames
Objects Mass
Objects Inertia
Objects Current Position
Objects Current Velocity
Objects Current Orientation
Objects Current Angular Velocity

The following values will also need to be determined to describe how the object responds to an interaction.

Unknowns for now
Coefficient of Static Friction
Coefficient of Dynamic Friction
Coefficient of Restitution
Point of Collision

Notice all the coefficients. These are easy because all the values will be between 0 and 1 and you get to determine them. I will explain later the consequences of choosing and changing their values. First you need to know the point or points at which the collision occurs. Sticking to the unique geometry of a sphere will make this easier. Notice that if a sphere is colliding with any other convex shape that there can only be one point of collision. This allows you to skip having to compute or measure a net response based multiple to a possible infinite collision points and also allows ease of finding the collision point.

If a sphere is to be detected colliding, this means that the sphere is overlapping into another objects space. To compute this differs depending on the other's shape and must yield to a response to get the sphere to a non overlapping state just touching the other object. How to get this response will not be discussed, but you need this to get the collision point.

The following math will calculate the point of collision:

Collision Point = The Negated Overlap Response Direction * Sphere Radius + Sphere's Center(Position)

The sphere's response to the overlap will push it in a direction directly away from the shape it collided with towards the spheres center. Now that it is just touching the other object the collision point is located in the opposite direction of the response at the distance equal to the spheres radius from the sphere's center position. Hopefully the

math above explains that better. The remainder of this tutorial will go through the math involved to compute the frame in which the sphere collides with another object.

Now that you have the collision point the relative point velocity at the collision point must be calculated:

Radvec1 = collision point – sphere position;
Radvec2 = collision point – object position;

Velocity at collision point From Angular Velocity of sphere =
Direction of [sphere angular velocity (cross) Radvec1] * Sphere Radius * Sphere's Angular Speed

Velocity at collision point From Angular Velocity of object =
Direction of [object angular velocity (cross) Radvec2] * Radvec2 length * Object's Angular Speed

Velocity at Collision Point from Sphere = Sphere Velocity – Velocity at collision point from angular velocity of sphere

Velocity at Collision Point from Object = Object Velocity – Velocity at collision point from angular velocity of object

Relative Velocity at the collision Point = Velocity at Collision Point from Sphere + Velocity at Collision Point from Object

You now have to split the relative velocity into two components. One is the velocity in the response direction of the sphere that we already have and two is the remaining velocity along a tangent or perpendicular that the object is sliding against.

Velocity in sphere response direction = [Relative Velocity at the collision Point (dot) sphere response direction] * sphere response direction

Velocity perpendicular to response direction = Relative Velocity at the collision Point - Velocity in sphere response direction

The reason for splitting the velocity into two parts is because different parts of the response are going to act only in a specific direction. This is where the coefficients mentioned above come in. The friction part of the response is going to act along the tangent of the sphere and object, while the restitution will act is the response direction or normal of the collision.

The Coefficient of Restitution controls how much of the relative velocity's magnitude in the response direction will be lost after the collision. The Coefficients of Static and Dynamic Friction control how the velocity in the direction of the surface perpendicular to the response direction will change, but will also cause angular velocity changes in two different situations. Static Friction is an immediate barrier that the collision point's movement must overcome before the point can slide and have Dynamic friction act upon it. Once there is enough force to overcome the static friction the dynamic friction controls how easily the objects slide against each other. Any opposition in an object sliding will cause rolling behavior especially in a sphere. All these values

range from 0 – 1 and are used to find fractions of the parts of the relative velocity to base the collision response on.

$$Jn(Impulse) = \frac{-(1+CoR) * RelVel(dot)Normal*(Normal)}{(MaInv1 + MaInv2) + (InInv1* (Rad1(cross)Normal)^2) + (InInv2*(Rad2(cross)Normal)^2)}$$

$$Jt(Impulse) = \frac{CoDF * RelVel(dot)Normal*(Tangent)}{(MaInv1 + MaInv2) + (InInv1* (Rad1(cross)Normal)^2) + (InInv2*(Rad2(cross)Normal)^2)}$$

$$Js(Impulse) = \frac{-(1+CoSF) * RelVel(dot)Normal*(-Tangent)}{(MaInv1 + MaInv2) + (InInv1* (Rad1(cross)Normal)^2) + (InInv2*(Rad2(cross)Normal)^2)}$$

Total Impulse = Jn + Jt + Js

Rad1 and Rad2 represent the distance from the center of mass(position) of the objects to the collision point.

This seems like too much math, but you can see the denominator in the equations is the same in all 3 and the dot product in the numerator is always the same. So most of this can be computed just once however we may want to compute the 3rd equation for static friction separately because it won't always be applied when the object slide against each other.

To calculate the changes in velocity and angular velocity

DeltaVel = Total Impulse / Mass
DeltaAngVel = Total Impulse(cross)Rad / Inertia

The new velocities will be tuned into a change in position and orientation when multiplied by the change in time between frames.

Applying this math will cause rigid bodies to perform much like objects perform here on earth and by changing the Coefficients you may achieve proper reactions depending on the results you want. Setting the friction to 0 will cause the objects to never change rotation but to slide easily against each other. Doing the same for restitution will cause no bounce as you would expect when two object bump each other such like pool balls. While this is a good approximation it isn't a precise representation of real physical phenomena. When it comes to simulating real physical things, the more accurate a result you need requires a much more detailed computational approach. This however is not ideal for all real time graphics applications as it will prove costly for hardware to accomplish if needed for many different objects.

The demo prepared allows you to fiddle around with the all three of these constants. The controls for doing so are:

Hold the 1 key and press up and down keys to range Restitution
Hold the 2 key and press up and down keys to range Dynamic Friction
Hold the 3 key and press up and down keys to range Static Friction

Pressing Space Bar will reset the ball
W, A, S, D and Mouse Movement allow camera movement
While Holding Left Mouse Button, Mouse movement will tilt the table up down left right

Works Cited

1) Chris Hecker “Rigid Body Dynamics” http://chrishecker.com/Rigid_Body_Dynamics 2007.