Physically Based Animation

One of the most exciting parts of computer graphics is rendering physical phenomena: realistic looking bodies of water with different character physics when swimming, explosions or bullet impacts that fracture objects in the scene, making a character rag-doll when they've died. I'm sure you've seen these effects in your favorite games or animated movies. The truth is, they are awfully hard to animate by hand or to write down a simple rule to change each matrix in the render step. All of these systems are governed by physics, some which we simulate very well in real time (rigid body dynamics) and some that we're only pretty good at (fluid simulation, deformable objects, etc.). We can exploit techniques in numerical analysis to track and simulate physics in our scene and render objects behaving under these physical constraints with a fair deal of realism. We're going to focus on frictionless non-deformable rigid-body dynamics for our assignment. This is a simplification of the physics you'd encounter in most 3D game engines like PhysX, Bullet, or Havok.

1 State Space

We wish to simulate the dynamics of several "rigid bodies" in a scene. A rigid body is intrinsically defined by its mass, center of mass, and inertial matrix. Its center of mass is the origin of the rigid body's coordinate system, and in world space we call it x_a. Its mass is M_a. Its inertial matrix I_a is a 3×3 matrix that relates torque to angular acceleration (also angular momentum to angular velocity). A rigid body A is extrinsically defined by its state vector at time t:

$$X_a(t) = \begin{bmatrix} x_a(t) \\ R_a(t) \\ P_a(t) \\ L_a(t) \end{bmatrix} \tag{1}$$

where $x_a(t)$, $R_a(t)$, $P_a(t)$, $L_a(t)$ are the position, rotation, linear momentum and angular momentum of A at time t respectively.

2 Derived Quantities

Writing down dynamics in terms of momentum is a bit easier than in terms of velocity, but we can always derive velocity terms at time t:

Linear Velocity (given M_a is the mass of A):

$$v_a(t) = \frac{P_a(t)}{M_a} \tag{2}$$

Inertial Matrix (Given I_a is the inertial matrix in body coordinates):

$$I_a(t) = R_a(t) \cdot I_a \cdot R_a(t)^\intercal \tag{3}$$

Angular velocity:

$$\omega_a(t) = I_a^{-1}(t) \cdot L(t) \tag{4}$$

3 Equations of Motion

For a point on A in body coordinates r_a, the configuration of that point in the world at time t is:

$$p_a(t) = x_a(t) + R_a(t) \cdot r_a \tag{5}$$

$F_p(t)$ is a force acting at point $p_a(t)$ on A at time t and the external torque $\tau_p(t)$ on a point $p_a(t)$ is

$$\tau_p(t) = (p_a(t) - x_a(t)) \times F_p(t) \tag{6}$$

The total external force on the body $F_a(t)$ is:

$$F_a(t) = \sum_{p \in A} F_p(t)$$

The total external torque on the body $\tau_a(t)$ is:

$$\tau_a(t) = \sum_{p \in A} \tau_p(t)$$

Then the differential equations defining the equations of motion for A are:

$$\dot{X}(X(t), t) = \begin{bmatrix} v_a(t) \\ \omega_a^*(t) \cdot R_a(t) \\ F_a(t) \\ \tau_a(t) \end{bmatrix} \tag{7}$$

Where

$$\omega_a^*(t) = \begin{bmatrix} 0 & -\omega_{az} & \omega_{ay} \\ \omega_{az} & 0 & -\omega_{ax} \\ -\omega_{ay} & \omega_{ax} & 0 \end{bmatrix}$$

and $\omega_a^*(t) \cdot v = \omega_a(t) \times v$ thus

$$\omega_a^*(t) \cdot R_a(t) = \begin{bmatrix} | & | & | \\ \omega_a(t) \times R_{ax} & \omega_a(t) \times R_{ay} & \omega_a(t) \times R_{az} \\ | & | & | \end{bmatrix}$$

3.1 Quaternions

If we are to represent the rotational configuration of the rigid body with a quaternion $q_a(t) = [s, v] = s + v_x i + v_y j + v_z k$, then the equivalent equations of motion are:

$$\dot{X}_a(t) = \begin{bmatrix} v_a(t) \\ \frac{1}{2}\omega_a(t) q_a(t) \\ F_a(t) \\ \tau_a(t) \end{bmatrix}$$

Where we use short hand for the quaternion multiplication:

$$\omega_a(t) q_a(t) = [0, \omega_a(t)] \cdot q_a(t)$$

3.2 Integrating equations of motion numerically

If we have our initial conditions of our system:

$$X_0 = \begin{bmatrix} x_0 \\ R_0 \\ P_0 \\ L_0 \end{bmatrix}$$

we can evolve the system forward with a simple first order integration of the dynamics called Euler's Method:

$$X_{n+1} = X_n + (t_{n+1} - t_n)\dot{X}(X_n, t_n)$$

For our system:

$$\begin{bmatrix} x(t_{n+1}) \\ R(t_{n+1}) \\ P(t_{n+1}) \\ L(t_{n+1}) \end{bmatrix} = \begin{bmatrix} x(t_n) \\ R(t_n) \\ P(t_n) \\ L(t_n) \end{bmatrix} + (t_{n+1} - t_n) \begin{bmatrix} v(t_n) \\ \omega^*(t_n) \cdot R(t_n) \\ F(t_n) \\ \tau(t_n) \end{bmatrix}$$

This can be seen as taking a first order Taylor Series approximation to the dynamics at a given point in time t_n. We define the step size to be $h = t_{n+1} - t_n$. We have to make a choice of step size, small enough such that our first order approximation is accurate, yet large enough so that our simulation is relatively efficient. This is a subtle choice in most cases. If we choose a fixed time step h and our simulation starts at t_0, then $t_n = t_0 + hn$

Euler's Method is the most basic method for numerical integration of ODEs, as such it is not very accurate and has some bad properties (namely is does not conserve kinetic energy). We can actually make a small change to our Euler update (that will amount to rearranging two lines in the code) that will surprisingly amount to a considerable improvement in the accuracy of our simulation. The idea is to update the momentum terms of the state vector for time $t+1$, and then use those values to update the position/rotation terms of the state vector:

$$\begin{bmatrix} P_{n+1} \\ L_{n+1} \end{bmatrix} = \begin{bmatrix} P_n \\ L_n \end{bmatrix} + h \begin{bmatrix} F(t_n) \\ \tau(t_n) \end{bmatrix}$$

Followed by:

$$\begin{bmatrix} x_{n+1} \\ R_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ R_n \end{bmatrix} + h \begin{bmatrix} v_{n+1} \\ \omega^*_{n+1} R_n \end{bmatrix}$$

Or written in momentum terms:

$$\begin{bmatrix} x_{n+1} \\ R_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ R_n \end{bmatrix} + h \begin{bmatrix} \frac{P_{n+1}}{M} \\ (I_n^{-1} \cdot L_{n+1})^* R_n \end{bmatrix}$$

This small change turns the basic explicit Euler's method into what's called Symplectic or Semi-Implicit Euler's method. This method increases both the order or the error and the order of the change in kinetic energy with respect to h.

4 Impulse Derivation

So far we can simulate the dynamics of rigid bodies experiencing external forces (wind, gravity, etc.) But we are not considering the interactions between colliding rigid bodies, nor rigid bodies colliding with the environment. In our theory, no parts of any two rigid bodies can occupy the same point in space. For rigid bodies A and B, this means that $A \cap B = 0$. Our dynamics currently will allow bodies to overlap. We need to detect the collision the moment it happens and apply a repulsive force in order to keep the bodies separated.

A and B are colliding at points $p_a(t)$ and $p_b(t)$. We designate the vector that is normal to the collision as $\hat{n}(t)$ (either face normal in vertex/face collision or edge cross product in edge/edge contact). We take the convention that $\hat{n}(t)$ points away from the surface of A. We know the velocities of the two points:

$$\dot{p}_a(t) = \dot{x}_a(t) + \omega_a(t) \times (p_a(t) - x_a(t))$$

and

$$\dot{p}_b(t) = \dot{x}_b(t) + \omega_b(t) \times (p_b(t) - x_b(t))$$

We can look at the component of the relative velocity of the two points in the direction of the contact normal.

$$v_{rel} = \hat{n}(t) \cdot (\dot{p}_a(t) - \dot{p}_b(t)) \tag{8}$$

If v_{rel} is positive, then the bodies in contact are moving away from each other and we don't have to worry about resolving the conflict. If v_{rel} is exactly 0, then the two bodies are in resting contact. This is a case that is a bit more complicated to resolve and involves computing non-penetration constraint forces for the entire scene at the same time. As such we will ignore that case for now, and as a result our objects will never be completely at rest with one another, there will be small jitters in slow moving objects in contact.

We consider the contact resolution case of $v_{rel} < 0$. To resolve the collision we need to apply a force to the two objects. In reality, any force would have to be applied over a period of time in order to enact change in the velocity, but we want a way to simulate the collision in one go, and instantaneously change the velocities. This can't be represented by continuous forces on the rigid body, so we will use what is called an impulse. An impulse J is much like a force, and is derived by integrating a force F over an increasingly small period of time Δt. Assuming mass is constant:

$$J = F\Delta t = \Delta P = M\Delta v \tag{9}$$

So we can relate the change in linear velocity to the impulse. Similarly for angular velocity:

$$\tau_{impulse} = (p(t) - x(t)) \times J = \Delta L = I(t)\Delta\omega(t) \tag{10}$$

For frictionless bodies, the impulse felt is only in the direction of the normal to the contact:

$$J = j\hat{n}(t) \tag{11}$$

We can also relate the relative velocities of the objects before and after the impulse has been applied. Let the velocity before collision be:

$$v_{rel}^- = \hat{n}(t) \cdot (\dot{p}_a^- - \dot{p}_b^-) \tag{12}$$

and after collision/impulse response be:

$$v_{rel}^+ = \hat{n}(t) \cdot (\dot{p}_a^+ - \dot{p}_b^+) \tag{13}$$

Then we use an empirical law for frictionless collision:

$$v_{rel}^{+} = -\epsilon\, v_{rel}^{-} \quad 0 \leq \epsilon \leq 1 \tag{14}$$

ϵ is called the "coefficient of restitution" and represents the elasticity of the collision. $\epsilon = 1$ represents the completely elastic collision (no net kinetic energy gained or lost). We know, even without friction, energy is lost during collision (be it through internal stress, sound, heat, etc.), so we choose an appropriate ϵ to represent the material we desire to simulate. We are now ready to compute the impulse for the collision.

Let's consider the change in linear velocity for Δv for the body A.

$$J = j\hat{n}(t) = \Delta P(t) = M_a \Delta v_a(t) = M_a(v_a^{+}(t) - v_a^{-}(t))$$

We can rearrange this to express $v_a^{+}(t)$ in terms of $j\hat{n}(t)$ and $v_a^{-}(t)$:

$$v_a^{+}(t) = v_a^{-}(t) + \frac{j\hat{n}(t)}{M_a} \tag{15}$$

Similarly we can consider the change in angular velocity for body A. We can define $r_a(t) = R_a(t) \cdot r_a = (p_a(t) - x_a(t))$ in order to simplify the notation a bit:

$$\tau_{impulse} = r_a(t) \times J = \Delta L = I_a(t)\Delta\omega_a(t) = I_a(t)(\omega_a^{+}(t) - \omega_a^{-}(t))$$

We can also rearrange this to express $\omega_a^{+}(t)$ in terms of $j\hat{n}(t)$ and $\omega_a^{-}(t)$:

$$\omega_a^{+}(t) = \omega_a^{-} + I_a^{-1}(t)(r_a(t) \times j\hat{n}(t)) \tag{16}$$

We can then combine (15) and (16) to describe the total change in velocity for $p_a(t)$:

$$\begin{aligned}
\dot{p}_a^{+}(t) &= v_a^{+}(t) + \omega_a^{+}(t) \times r_a(t) \\
&= (v_a^{-}(t) + \frac{j\hat{n}(t)}{M_a}) + (\omega_a^{-}(t) + I_a^{-1}(t)(r_a(t) \times j\hat{n}(t)) \times r_a(t)) \\
&= v_a^{-}(t) + \omega_a^{-}(t) \times r_a(t) + \left(\frac{\hat{n}j}{M_a}\right) + \left(I_a^{-1}(t)\big(r_a(t) \times j\hat{n}(t)\big)\right) \times r_a(t) \\
&= \dot{p}_a^{-} + j\left(\frac{\hat{n}(t)}{M_a} + I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\right) \times r_a(t)
\end{aligned} \tag{17}$$

Newton's third law tells us that the rigid body B will feel an equal and opposite impulse $-j\hat{n}(t)$, thus:

$$\dot{p}_b^{+} = \dot{p}_b^{-} - j\left(\frac{\hat{n}(t)}{M_b} + I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\right) \times r_b(t) \tag{18}$$

Subtracting (18) from (17) gets us closer to our original heuristic for the change in relative velocity:

$$
\begin{aligned}
\dot{p}_a^+(t) - \dot{p}_b^+(t) = (\dot{p}_a^- - \dot{p}_b^-) + j\Bigg(\frac{\hat{n}(t)}{M_b} + \frac{\hat{n}(t)}{M_a} + \big(I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\big) \times r_a(t) + \\
\big(I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\big) \times r_b(t)\Bigg)
\end{aligned} \tag{19}
$$

If we dot both sides of (19) by $\hat{n}(t)$ we get an equation in terms of v_{rel}^+ and v_{rel}^-

$$
\begin{aligned}
v_{rel}^+ &= \hat{n}(t) \cdot (\dot{p}_a^+(t) - \dot{p}_b^+(t)) \\
&= \hat{n}(t) \cdot (\dot{p}_a^-(t) - \dot{p}_b^-(t) \ + \quad \hat{n}(t) \cdot j\Bigg(\frac{\hat{n}(t)}{M_a} + \frac{\hat{n}(t)}{M_b} + \big(I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\big) \times r_a(t) + \\
&\qquad \big(I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\big) \times r_b(t)\Bigg) \\
&= \ v_{rel}^- \qquad + \qquad j\Bigg(\frac{1}{M_a} + \frac{1}{M_b} \ + \ \hat{n}(t) \cdot \Big[\big(I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\big) \times r_a(t) + \\
&\qquad \big(I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\big) \times r_b(t)\Big]\Bigg)
\end{aligned}
$$

If we then use (11), our heuristic for frictionless collision $v_{rel}^+ = -\epsilon \ v_{rel}^-$:

$$
\begin{aligned}
-\epsilon \ v_{rel}^- = v_{rel}^- + j\Bigg(\frac{1}{M_a} + \frac{1}{M_b} + \hat{n}(t) \cdot \Big[\big(I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\big) \times r_a(t) + \\
\big(I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\big) \times r_b(t)\Big]\Bigg)
\end{aligned}
$$

We can then solve for j:

$$
j = \frac{-(1+\epsilon)v_{rel}^-}{\Bigg(\frac{1}{M_a} + \frac{1}{M_b} + \hat{n}(t) \cdot \Big[\big(I_a^{-1}(t)\big(r_a(t) \times \hat{n}(t)\big)\big) \times r_a(t) + \big(I_b^{-1}(t)\big(r_b(t) \times \hat{n}(t)\big)\big) \times r_b(t)\Big]\Bigg)} \tag{20}
$$

For fixed bodies (walls/ground) or bodies that aren't simulated in our dynamics but have collision (kinematically controlled character animations), we can set $\frac{1}{M} = 0$ and $I^{-1} = \mathbf{0}$ and ignore updating their state vectors with impulses. The effect for the moving rigid bodies will be to effectively reflect the component of it's velocity normal to the surface across the surface (with some energy loss due to ϵ.

I'll admit, the derivation is a bit hairy, and that's just for two interacting bodies! Imagine getting into more complicated dynamics: adding friction, constraint forces, connecting rigid bodies at joints to create composite bodies. $F = ma$ in world space gets verbose pretty quick. In robotics we use Screw Theory and more recently Spatial Vector Algebra to simulate rigid body dynamics on entities that are recursively defined with constrained joints, etc in a more compact and algorithm friendly way.

Many of the derivations in these notes were borrowed from a well known source on simulating rigid body dynamics for graphics, Physically Based Modelling.