Basics of rigid body motion for computer animation

Steve Marschner
CS 5643 Spring 2025
Cornell University

Many objects don't deform very much in everyday settings—a hammer, a brick, a ceramic bowl, a block of wood. Of course these objects do deform, but the deformations are small and happen on short timescales so that we don't see them, and it's often a useful approximation to model such objects as *rigid*, meaning that the only motions they undergo are rotations and translations. Objects that move rigidly are known as *rigid bodies*, and computing the motion of rigid bodies is in some ways easier and in other ways harder than deformable bodies.

On the plus side, the motion of a rigid body can be described with far fewer degrees of freedom: we'll see that a 3D rigid body only needs 6 DoF to describe its motion, whereas a deformable body of course needs a whole mesh with lots of vertex DoFs. Also, deformable objects that have high stiffness require implicit integrators or small timesteps to simulate stably. On the other hand, rigid bodies communicate the effects of collisions instantaneously across the whole object, which makes it hard to get away with reasoning locally about collision and contact when rigid bodies are involved.

In these notes we'll develop the basic math for simulating rigid bodies in 2D, with occasional forays into 3D. This includes resolving instantaneous collisions, but we will leave robust methods for resting contact to later notes.

Kinematics

Everything we have simulated so far has been in terms of particle motion: even the most sophisticated deformable object is still represented as a collection of masses at its vertices, and the simulation is all about computing forces on those vertices. Rigid bodies depart from this way of doing things.

A particle has only a position; a rigid body has both a position and an orientation. Just like with deformable objects, you can think of the state of a rigid body as a mapping from a point $\mathbf{r}_b$ in a rest space to a point $\mathbf{r}$ in the scene; in this case the rest space is called "body space" and the mapping is a rigid motion

$$\mathbf{r} = \mathbf{R}(t)\mathbf{r}_b + \mathbf{x}(t) \quad \text{where } \mathbf{R} \text{ is a rotation and } \mathbf{x} \text{ is a position in the scene.}$$

The time-varying rotation and position together describe the body's motion. In 2D a rotation has only one degree of freedom—it can be described by a single angle—and in 3D there are three DoFs, for

instance three Euler angles (we'll come back to how best to actually represent the rotation in code). So in total a 2D rigid body has 3 DoF and a 3D rigid body has 6 DoF.

To describe the velocity of a particle, we wrote down its velocity, which is the time derivative of the position. To write down the velocity of a rigid body we also need the time derivative of the rotation. What does that mean, the derivative of a rotation? The rotation is a linear transformation, which can be written out as a square matrix, so one good way to think about the derivative of a rotation is as the derivative of the rotation matrix.

To expand on this a bit, think of a fixed body-space point $\mathbf{r}_b$ that is just rotating:

$$\mathbf{r}(t) = \mathbf{R}(t)\mathbf{r}_b$$

If we take the time derivative of $\mathbf{r}$ we get

$$\dot{\mathbf{r}}(t) = \dot{\mathbf{R}}(t)\mathbf{r}_b + \mathbf{R}(t)\dot{\mathbf{r}}_b = \dot{\mathbf{R}}(t)\mathbf{r}_b \quad (\mathbf{r}_b \text{ is constant since the point is not moving in body space})$$

So the rotation $\mathbf{R}$ is a linear transformation that maps a body-space point to that point's *position* in world space, and its derivative $\dot{\mathbf{R}}$ is a linear transformation that maps a body-space point to that point's *velocity* in world space.

Rotation matrices are orthogonal: $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$. This makes them quite special: this orthogonality amounts to 6 constraints on 9 entries in the 3D case or 3 constraints on 4 entries in the 2D case, leaving the 3 or 1 DoF discussed above. The derivative of the rotation is similarly constrained, and we can show how with a clever algebraic manipulation: take the time derivative of the product $\mathbf{R}\mathbf{R}^T$.

$$\begin{aligned}
\frac{d}{dt}(\mathbf{R}\mathbf{R}^T) &= \dot{\mathbf{R}}\mathbf{R}^T + \mathbf{R}\dot{\mathbf{R}}^T \\
\mathbf{0} &= \dot{\mathbf{R}}\mathbf{R}^T + (\dot{\mathbf{R}}\mathbf{R}^T)^T \\
\dot{\mathbf{R}}\mathbf{R}^T &= -(\dot{\mathbf{R}}\mathbf{R}^T)^T
\end{aligned}$$

This says that $\dot{\mathbf{R}}\mathbf{R}^T$ is an antisymmetric matrix. A similar argument shows that $\mathbf{R}^T\dot{\mathbf{R}}$ is also antisymmetric.

An antisymmetric (or skew symmetric) matrix is simply a matrix with the property that $\mathbf{A} = -\mathbf{A}^T$, which implies a few things:

- If you look at the entries of $\mathbf{A}$, $a_{ij} = -a_{ji}$ (hence the name antisymmetric).
- Considered as a linear transformation, $\mathbf{A}$ transforms a vector $\mathbf{x}$ to a vector that is perpendicular to $\mathbf{x}$. You can see this because $\mathbf{x} \cdot \mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{A}\mathbf{x}$, being a scalar, is equal to its transpose $\mathbf{x}^T\mathbf{A}^T\mathbf{x} = -\mathbf{x}^T\mathbf{A}\mathbf{x}$. Since $\mathbf{x} \cdot \mathbf{A}\mathbf{x} = -\mathbf{x} \cdot \mathbf{A}\mathbf{x}$, it must be zero, so $\mathbf{x}$ is orthogonal to $\mathbf{A}\mathbf{x}$.

A familiar operation that produces a vector orthogonal to its input is the cross product, and it turns out that antisymmetric matrices in 2D and 3D do the same thing as cross products. In particular, for any antisymmetric matrix $\mathbf{A}$ there is a vector $\boldsymbol{\omega}$ that has the property $\mathbf{A}\mathbf{x} = \boldsymbol{\omega} \times \mathbf{x}$ for all $\mathbf{x}$. In 3D this vector might point in any direction, and in 2D this vector is a z-axis vector that is orthogonal to the x-y plane. We write $\mathbf{A} = \boldsymbol{\omega}^\times$ to mean "$\mathbf{A}$ is the antisymmetric matrix that implements taking the cross product of $\boldsymbol{\omega}$ with a vector" and we call $\boldsymbol{\omega}$ the axial vector of $\mathbf{A}$.

Bringing this fact back to the topic of rotations, we can write the matrix $\dot{\mathbf{R}}\mathbf{R}^T$ using its axial vector which we name $\boldsymbol{\omega}$ and call the angular velocity. Then

$$\begin{aligned}\boldsymbol{\omega}^\times &= \dot{\mathbf{R}}\mathbf{R}^T \\ \dot{\mathbf{R}} &= \boldsymbol{\omega}^\times\mathbf{R}\end{aligned}$$

This is handy because in code we can keep track of the angular velocity vector $\boldsymbol{\omega}$, and then we can compute $\dot{\mathbf{R}}$ whenever we need it. Any vector $\boldsymbol{\omega}$ produces a valid $\dot{\mathbf{R}}$ (meaning a way that $\mathbf{R}$ can change while staying a rotation), and every valid $\dot{\mathbf{R}}$ can be produced in this way.

So to keep track of the velocity of a rigid body we need two things:

- a vector $\mathbf{v} = \dot{\mathbf{x}}$ which describes how the body's position in world space is changing with time: it is the world velocity of the origin of the body's local coordinates.
- a vector $\boldsymbol{\omega}$ for which $\dot{\mathbf{R}} = \boldsymbol{\omega}^\times\mathbf{R}$, which describes how the body's orientation in world space is changing with time: it is the angular velocity of the body, expressed in world space.

With these two pieces of information in hand we can compute the velocity of a point on the body. If we know the point's body coordinates $\mathbf{r}_b$, then its world space velocity expands to

$$\begin{aligned}\mathbf{r} &= \mathbf{x} + \mathbf{R}\mathbf{r}_b \\ \dot{\mathbf{r}} &= \dot{\mathbf{x}} + \dot{\mathbf{R}}\mathbf{r}_b \\ &= \mathbf{v} + \boldsymbol{\omega}^\times\mathbf{R}\mathbf{r}_b = \mathbf{v} + \boldsymbol{\omega} \times (\mathbf{R}\mathbf{r}_b)\end{aligned}$$

On the other hand, if we know the point's world space coordinates $\mathbf{r}$ then $\mathbf{R}\mathbf{r}_b = \mathbf{r} - \mathbf{x}$ and we can compute the velocity as

$$\dot{\mathbf{r}} = \mathbf{v} + \boldsymbol{\omega} \times (\mathbf{r} - \mathbf{x})$$

The body's linear velocity $\mathbf{v}$ contributes the same to the velocities of all points on the body, but the angular velocity adds a contribution that is proportional to the distance of a point from the body's center.

So the kinematic state of a rigid body has four parts: $\mathbf{x}$, $\mathbf{R}$, $\mathbf{v}$, and $\boldsymbol{\omega}$. These four quantities define where the body is and where it is going, and they are the basis for writing the equations of motion in the next section.

Aside on cross products in 3D and 2D

The cross product is familiar as an operation on 3D vectors: it has two vector operands $\mathbf{u}$ and $\mathbf{v}$ and returns a vector $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ that points along the line orthogonal to both of the operands. The length is $\|\mathbf{w}\| = \|\mathbf{u}\|\|\mathbf{v}\| \sin(\mathbf{u}, \mathbf{v})$, and the direction is chosen so that $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ form a right-handed triad.

In a 2D simulation, the vectors $\mathbf{x}$, $\mathbf{r}$, and $\mathbf{v}$ all lie in a plane, call it the x-y plane. The vector $\boldsymbol{\omega}$ is perpendicular to the plane, pointing along the z axis. This invites us to use 2D vectors to represent the in-plane quantities and scalars to represent out-of-plane vectors like ω. You can think of this as just an optimization where we don't store numbers we know are always zero, or as a kind of type system with two kinds of quantities. But in any case in practice it means there are two kinds of cross products to code up: one that takes two vectors and returns a scalar

$$w = \mathbf{u} \times \mathbf{v} = u_x v_y - u_y v_x \quad \text{— the signed area of the parallelogram spanned by } \mathbf{u} \text{ and } \mathbf{v}$$

and one that takes a scalar and a vector and returns a vector

$$\mathbf{w} = u \times \mathbf{v} = (-u v_y, u v_x) \quad \text{— the vector } \mathbf{v} \text{ rotated 90 degrees counterclockwise and scaled by } u$$

In both cases the formulas are just specializations of the usual 3D ones.

In these notes I'll use non-bold ω as a cue that I am talking about the 2D case and the equations may not be expected to generalize to 3D.

Kinematics in code

In a rigid body simulator we need an efficient way to write down the state. $\mathbf{x}$, $\mathbf{v}$, and $\boldsymbol{\omega}$ are all pretty straightforward: in 3D they are all 3-vectors, and in 2D $\mathbf{x}$ and $\mathbf{v}$ are 2-vectors and ω can be represented as a scalar. But $\mathbf{R}$ is a little more awkward because there is not such a neat way to write it down uniquely as a set of numbers.

In 2D we have

$$\mathbf{R} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$$

where s and c are the sine and cosine of an angle. One reasonable option is to write down the angle, but it's a little unsatisfying that the representation is not unique (angles differing by $2\pi k$ describe the same orientation), and it's practically troublesome that if we just let an object spin for a long time its angle will grow to a large number so that the accuracy of its floating point representation eventually degrades. Also, we will have to compute trigonometric functions all the time whenever we need to apply $\mathbf{R}$ to anything.

These problems all have fine workarounds, but another option is to store the pair $\mathbf{q} = (s, c)$ as a 2D unit vector and ensure that it stays normalized as it evolves. Then the matrix entries are right there, they are always in the range $[-1,1]$, and the representation is unique. The downside is redundancy: whenever we update $\mathbf{q}$ we need to normalize it. But this is cheaper arithmetic than trig functions.

Since $\mathbf{q}$ is the first column of $\mathbf{R}$, it is simple to compute its time derivative:

$$\dot{\mathbf{q}} = \omega^{\times} \mathbf{q} = \omega \times \mathbf{q}$$

So the position update in an Euler integrator looks like

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \mathbf{v}_k$$
$$\mathbf{q}_{k+1} = \text{normalize}(\mathbf{q}_k + h \omega \times \mathbf{q}_k)$$

In 3D there is not a satisfactory way to write down the rotation in terms of angles, and the best solution is to use a quaternion. We won't delve into the details here but in 3D, where $\mathbf{x}$, $\mathbf{v}$, and $\boldsymbol{\omega}$ are vectors with 3 components and $\mathbf{q}$ is a quaternion with 4 components, the derivative of $\mathbf{q}$ is

$$\dot{\mathbf{q}} = \tfrac{1}{2} \boldsymbol{\omega} \mathbf{q} \quad \text{— where the multiplication is a quaternion product.}$$

The redundancy of representing the rotation with a 4D quaternion is exactly analogous to representing the 2D orientation with a 2D unit vector, and comes with the same need to keep the quaternion normalized. The corresponding Euler position update looks like this:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\,\mathbf{v}_k$$
$$\mathbf{q}_{k+1} = \text{normalize}(\mathbf{q}_k + \frac{h}{2}\boldsymbol{\omega}\,\mathbf{q}_k)$$

Momentum and angular momentum

To describe the dynamics of a rigid body we need to know about the body's mass and how it is distributed over the body's volume or area, which can be described by a mass density $\rho(\mathbf{r}_b)$ in body space. The body's mass distribution leads to some quantities that are useful for describing motion. The simplest is the total mass

$$m = \int_B \rho(\mathbf{x})\, d\,\mathbf{x}$$

The mass gives us the relationship between translational velocity and translational kinetic energy:

$$E_k = \tfrac{1}{2} m\,\mathbf{v}^2 = \tfrac{1}{2}\mathbf{v}\cdot m\,\mathbf{v} = \tfrac{1}{2}\mathbf{v}\cdot\mathbf{p} \quad \text{— where we have named } m\,\mathbf{v}\text{, which is linear momentum, } \mathbf{p}.$$

Let's dig a little more into how the kinetic energy of the body comes from the kinetic energy of the body's mass. Let's also do this in 2D to start. A small area dA of the body has a mass of $\rho(\mathbf{r})\,dA$ and a kinetic energy of

$$\frac{1}{2}\rho(\mathbf{r})\mathbf{v}^2(\mathbf{r})\,dA \quad \text{— so } 0.5\rho(\mathbf{r})\mathbf{v}^2(\mathbf{r}) \text{ is the area density of kinetic energy in the body.}$$

In this case $\mathbf{v}(\mathbf{r})$ is constant so when we integrate to get the total kinetic energy we get

$$\begin{aligned} E_k &= \frac{1}{2}\int \rho(\mathbf{r})\mathbf{v}^2(\mathbf{r})\,dA \\ &= \left(\frac{1}{2}\int \rho(\mathbf{r})\,dA\right)\mathbf{v}^2(\mathbf{r}) \\ &= \frac{1}{2}m\,\mathbf{v}^2 \quad \text{where} \quad m = \int \rho(\mathbf{r})\,dA \end{aligned}$$

This was a roundabout way of deriving something that was maybe obvious: when all the mass is moving in the same direction we can just add up the mass to get the scale factor between velocity and momentum.

Now let's think about what happens with a rotational motion. Suppose the body is rotating about the origin with angular velocity ω. (Since I am in the 2D case I'm thinking of ω as a scalar.) Now the velocity depends on the point, but we can still compute the total kinetic energy by integrating over the body:

$$\mathbf{v}(\mathbf{r}) = \omega \times \mathbf{r} \text{ and } \|\mathbf{v}(\mathbf{r})\| = \omega\|\mathbf{r}\|$$

$$\begin{aligned} E_k &= \frac{1}{2}\int \rho(\mathbf{r})\mathbf{v}^2(\mathbf{r})\, dA \\ &= \frac{1}{2}\int \rho(\mathbf{r})\omega^2\mathbf{r}^2\, dA \\ &= \frac{1}{2} I\omega^2 \quad \text{where} \quad I = \int \rho(\mathbf{r})\mathbf{r}^2\, dA \end{aligned}$$

This scale factor I is called the *moment of inertia*. It's a measurement of mass weighted by squared distance from the body's center of mass. We can write this in a few ways again, all very analogous to the case of linear kinetic energy above:

$$E_k = \tfrac{1}{2} I\omega^2 = \tfrac{1}{2}\omega \cdot I\omega = \tfrac{1}{2}\omega \cdot L \quad \text{— where we have named } I\omega, \text{ which is angular momentum, } L.$$

Just for completeness, the center of mass can be defined similarly to the moment of inertia, but without the square and with a mass normalization.

$$\mathbf{r}_c = \frac{1}{m}\int \rho(\mathbf{r})\mathbf{r}\, dA \quad \text{— and remember our convention that } \mathbf{r}_c = 0 \text{ in body coordinates.}$$

Rotational kinetic energy and angular momentum in 3D (can skip for A2!)

In 3D, the moment of inertia depends on the axis of rotation. In this case the velocity $\mathbf{v}(\mathbf{r})$ of a point depends on both the rotation speed and its distance from the axis: $\mathbf{v}(\mathbf{r})^2 = (\boldsymbol{\omega} \times \mathbf{r})^2$. Another way of writing this is:

$$\begin{aligned} \mathbf{v}(r)^2 = (\boldsymbol{\omega} \times \mathbf{r})^2 &= \|\boldsymbol{\omega}\|^2\|\mathbf{r}\|^2 - (\boldsymbol{\omega} \cdot \mathbf{r})^2 && \text{Lagrange's identity} \\ &= (\boldsymbol{\omega}^T\boldsymbol{\omega})(\mathbf{r}^T\mathbf{r}) - (\boldsymbol{\omega}^T\mathbf{r})(\boldsymbol{\omega}^T\mathbf{r}) \\ &= \boldsymbol{\omega}^T(\mathbf{r}^T\mathbf{r})\,\boldsymbol{\omega} - \boldsymbol{\omega}^T(\mathbf{r}\mathbf{r}^T)\,\boldsymbol{\omega}^T \end{aligned}$$

Using this to compute the kinetic energy for a body with angular velocity $\boldsymbol{\omega}$ leads to

$$\begin{aligned} E_k &= \frac{1}{2}\int \rho(\mathbf{r})\mathbf{v}(\mathbf{r})^2 dV \\ &= \frac{1}{2}\int \rho(\mathbf{r})\left(\boldsymbol{\omega}^T(\mathbf{r}^T\mathbf{r})\,\boldsymbol{\omega} - \boldsymbol{\omega}^T(\mathbf{r}\mathbf{r}^T)\,\boldsymbol{\omega}^T\right) dV \\ &= \frac{1}{2}\boldsymbol{\omega}^T\left(\int \rho(\mathbf{r})(\mathbf{r}^T\mathbf{r} - \mathbf{r}\mathbf{r}^T) dV\right)\boldsymbol{\omega} \\ &= \frac{1}{2}\boldsymbol{\omega}^T\mathbf{I}\,\boldsymbol{\omega} \end{aligned}$$

Writing this in terms of the rotation speed $\|\boldsymbol{\omega}\|$,

$$E_k = \tfrac{1}{2}\hat{\boldsymbol{\omega}}^T\mathbf{I}\hat{\boldsymbol{\omega}} \cdot \|\boldsymbol{\omega}\|^2$$

Comparing this to the 2D case we can see that the moment of inertia for a rotation around the axis $\hat{\boldsymbol{\omega}}$ is $\hat{\boldsymbol{\omega}}^T\mathbf{I}\hat{\boldsymbol{\omega}}$. The matrix $\mathbf{I}$ is called the *inertia tensor* and it's a symmetric matrix that tells you the moment of

inertia of the body around any rotation axis. We can factor the energy as the product of velocity and momentum in this case too:

$$E_k = \tfrac{1}{2}\boldsymbol{\omega} \cdot \mathbf{I}\boldsymbol{\omega} = \tfrac{1}{2}\boldsymbol{\omega} \cdot \mathbf{L} \quad \text{— the vector } \mathbf{I}\boldsymbol{\omega} \text{ is the angular momentum, called } \mathbf{L}$$

In 3D, like in 2D, a body's angular momentum is conserved (in world coordinates) if no external forces act on it. But unlike in 2D, because rotating the body changes $\mathbf{I}$ (in world coordinates), this does not necessarily mean that its angular velocity, rotation axis, or rotation speed remains constant.

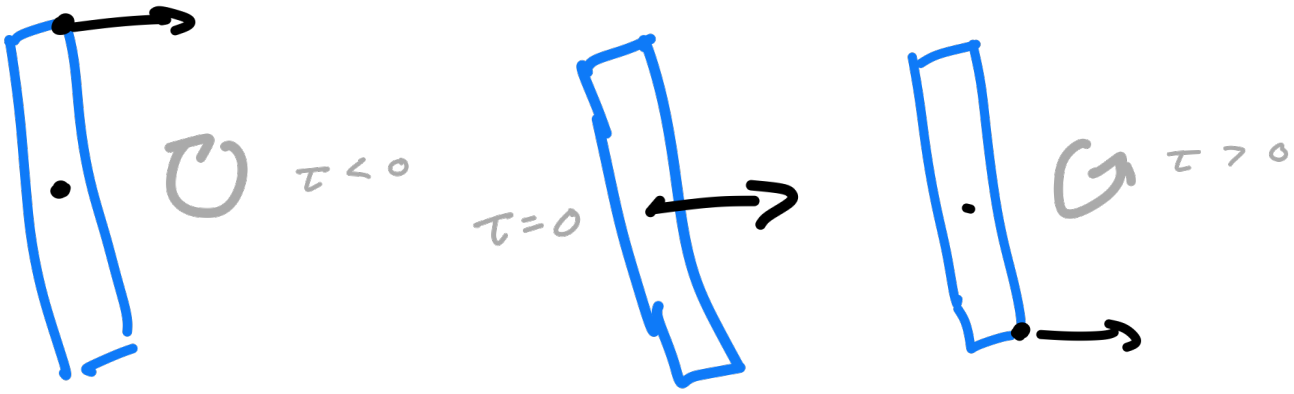
Force and torque

The motion of a rigid body is affected by forces applied to the body; the effect of such forces is more complicated than for particles because we have to consider the effect of force not only on velocity but also on angular velocity.

The effect of an applied force on the center-of-mass velocity is not affected by a body's ability to rotate; regardless of what happens with angular velocity, force and acceleration are still related in the same familiar way:

$$\mathbf{f}_{\text{ext}} = m\dot{\mathbf{v}} = \dot{\mathbf{p}} \quad \text{— the external force equals the rate of change of the body's momentum}$$

The effect of an applied force on angular velocity is less simple, because the effect depends on where the force is applied relative to the point around which rotation is measured (in our case this is always the center of mass). Consider the three 2D cases illustrated here:



In all three cases the same horizontal force $\mathbf{f}$ is applied at a point $\mathbf{r}$, but when $\mathbf{r}$ is above the rotation center, the force makes the angular velocity decrease (i.e. more clockwise); when the force is applied in line with the center, it does not affect the angular velocity; and when $\mathbf{r}$ below the rotation center, the force makes the angular velocity increase (i.e. more counterclockwise). The effect is proportional to the distance of the point from the line through the center parallel to the direction of the force, and quantitatively we can express it simply:

$$I\dot{\omega} = \mathbf{r}' \times \mathbf{f} \quad \text{— this is for 2D only, note the scalar } I \text{ and } \omega$$

Here $\mathbf{r}' = \mathbf{r} - \mathbf{x}$ is the vector from the rotation center to the point $\mathbf{r}$. The product $\mathbf{r}' \times \mathbf{f}$ is called the *torque* and denoted τ. So the 2D equations of motion for a rigid body are

$$\mathbf{f}_{\text{ext}} = m\dot{\mathbf{v}} = \dot{\mathbf{p}}$$
$$\tau_{\text{ext}} = I\dot{\omega} = \dot{L}$$

Equations of motion for 3D (can skip for A2)

In 3D the momentum equations generalize:

$$\mathbf{f}_{\text{ext}} = \dot{\mathbf{p}}$$
$$\boldsymbol{\tau}_{\text{ext}} = \dot{\mathbf{L}}$$

For this reason, in 3D simulations it's simplest to use $\mathbf{L}$ as the state variable, rather than $\boldsymbol{\omega}$ — and then, for consistency, to also use $\mathbf{p}$ in place of $\mathbf{v}$. You could do the same in 2D simulations also, although it seems to be typical to stick with velocity in that case.

If you use $\mathbf{L}$ as your state variable, you can still compute $\boldsymbol{\omega} = \mathbf{I}^{-1}\mathbf{L}$ whenever you need it. Note that when you do this, since $\mathbf{L}$ is in world coordinates and you know $\mathbf{I}_b$ in body coordinates, you need to compute $\mathbf{R}\mathbf{I}_b^{-1}\mathbf{R}^T\mathbf{L}$, but you can precompute $\mathbf{I}_b^{-1}$ since it is constant over time.

Impulses and torque impulses

When dealing with collisions of particles we found it useful to use the idea of an impulse: a force that is applied over a short time interval so that we only can see the effect of the product of force and time, which is an impulse. If we apply a constant force over a time Δt, it produces a change in momentum:

$$\mathbf{f}\Delta t = \dot{\mathbf{p}}\Delta t = \Delta\mathbf{p} = \mathbf{j} \quad \text{— where } \mathbf{j} \text{ is the name we give to the impulse } \mathbf{f}\Delta t$$

In the same way, if we apply a constant torque over a time Δt, it produces a change in angular momentum:

$$\boldsymbol{\tau}\Delta t = \dot{\mathbf{L}}\Delta t = \Delta\mathbf{L} = \mathbf{r}' \times \mathbf{j} \quad \text{— } \mathbf{r}' \times \mathbf{j} \text{ is the } \textit{torque impulse} \text{ which doesn't get its own symbol}$$

A subtle point here is that this idea of an impulse delivering a constant torque over a time interval depends on the time interval being short relative to the body's rotation speed, so that $\mathbf{r}'$ can be considered constant.

These rules will be useful for collision resolution:

$\mathbf{j} = \Delta\mathbf{p} = m\Delta\mathbf{v}$	in any dimension
$\mathbf{r}' \times \mathbf{j} = \Delta\mathbf{L}$	in 2D or 3D
$\mathbf{r}' \times \mathbf{j} = \Delta L = I\Delta\omega$	in 2D only

Collisions between rigid bodies

Finally I want to work out the impulse that results from a collision between two rigid bodies at a single point, which lets us make animations of rigid bodies bouncing off of things and will later be the core of a method for dealing with systems of contacts between many bodies.

Let's start by reviewing contacts between pairs of spherical particles. Suppose we have a pair of colliding spheres A and B, with velocities $\mathbf{v}_a$ and $\mathbf{v}_b$ and relative velocity along the normal direction (i.e. along the line between the centers) given by

$$v_n = \mathbf{n} \cdot (\mathbf{v}_a - \mathbf{v}_b)$$

We resolved the collision by applying impulses $\gamma \hat{\mathbf{n}}$ to particle A and $-\gamma \hat{\mathbf{n}}$ to particle B, which led to

$$\begin{aligned}\mathbf{v}_a^+ &= \mathbf{v}_a^- + m_a^{-1} \gamma \hat{\mathbf{n}} \\ \mathbf{v}_b^+ &= \mathbf{v}_b^- - m_b^{-1} \gamma \hat{\mathbf{n}}\end{aligned}$$

where superscript minus means "before the collision" and superscript plus means "after the collision." The normal component of the difference is

$$v_n^+ = \hat{\mathbf{n}} \cdot (\mathbf{v}_a^+ - \mathbf{v}_b^+) = v_n^- + (m_a^{-1} + m_b^{-1})\gamma.$$

To determine the impulse magnitude γ we used a simple heuristic (the "restitution hypothesis") that the post-contact normal velocity is opposite the pre-contact normal velocity but scaled down to account for energy loss in the collision:

$$v_n^+ = -c_r v_n^-$$

There is nothing particularly fundamental about this equation; it is just a heuristic model for how energy gets stored as elastic deformation and then returned to kinetic energy during the collision. Substituting,

$$\begin{aligned}-c_r v_n^- &= v_n^- + \left(m_a^{-1} + m_b^{-1}\right)\gamma \\ \left(m_a^{-1} + m_b^{-1}\right)\gamma &= -(1 + c_r) v_n^- \\ \gamma &= -(1 + c_r) m_{\text{eff}} v_n^-\end{aligned}$$

where $m_{\text{eff}} = \left(m_a^{-1} + m_b^{-1}\right)^{-1}$.

Standing back from this simple derivation we can see three steps:

1. Write the normal velocity in terms of the object velocities.

2. Write the objects' new velocities in terms of the collision impulse.

3. Use the restitution hypothesis to solve for the collision impulse.

Let's play out the same program for a pair of rigid bodies. The same steps will work, and we'll just need some extra terms to account for rotation in each step. I'll write out everything in 2D, but nothing here

really changes in 3D: the angular velocity will be a vector but the cross products all just go through the same way.

Step 1: Suppose we have two bodies A and B that are in contact; the first thing we need is equations to compute the relative normal velocity at the collision from the system state. In this case the relevant system state includes the linear velocities $\mathbf{v}_a$ and $\mathbf{v}_b$ and the angular velocities ω_a and ω_b, and we'll also need to know the contact point $\mathbf{r}$ and the contact normal $\hat{\mathbf{n}}$, which by convention points from B towards A. The velocities of the two colliding points are:

$$\mathbf{v}_a + \omega_a \times \mathbf{r}_a \quad \text{where } \mathbf{r}_a = \mathbf{r} - \mathbf{x}_a$$

$$\mathbf{v}_b + \omega_b \times \mathbf{r}_b \quad \text{where } \mathbf{r}_b = \mathbf{r} - \mathbf{x}_b$$

so the normal component of the relative velocity is

$$v_n = \hat{\mathbf{n}} \cdot \left(\mathbf{v}_a - \mathbf{v}_b + \omega_a \times \mathbf{r}_a - \omega_b \times \mathbf{r}_b\right).$$

Step 2 in the program is to write the objects' new velocities $\mathbf{v}_a^+$ and $\mathbf{v}_b^+$ in terms of the collision impulse, and compute the post-collision relative velocity in the normal direction. Our collision impulse will be $\gamma\,\hat{\mathbf{n}}$ for A and $-\gamma\,\hat{\mathbf{n}}$ for B, so following the previous section:

$$\Delta\mathbf{v}_a = m_a^{-1}\gamma\,\hat{\mathbf{n}} \qquad \Delta\omega_a = I_a^{-1}\mathbf{r}_a \times \gamma\,\hat{\mathbf{n}}$$

$$\Delta\mathbf{v}_b = -\,m_b^{-1}\gamma\,\hat{\mathbf{n}} \qquad \Delta\omega_b = -\,I_b^{-1}\mathbf{r}_b \times \gamma\,\hat{\mathbf{n}}$$

The new normal-direction relative velocity is:

$$\begin{aligned}
v_n^+ &= \hat{\mathbf{n}} \cdot \left(\mathbf{v}_a^+ - \mathbf{v}_b^+ + \omega_a^+ \times \mathbf{r}_a - \omega_b^+ \times \mathbf{r}_b\right) \\
&= \hat{\mathbf{n}} \cdot \left((\mathbf{v}_a^- + \Delta\mathbf{v}_a) - (\mathbf{v}_b^- + \Delta\mathbf{v}_b) + (\omega_a^- + \Delta\omega_a) \times \mathbf{r}_a - (\omega_b^- + \Delta\omega_b) \times \mathbf{r}_b\right) \\
&= v_n^- + \hat{\mathbf{n}} \cdot \left(\Delta\mathbf{v}_a - \Delta\mathbf{v}_b + \Delta\omega_a \times \mathbf{r}_a - \Delta\omega_b \times \mathbf{r}_b\right) \\
&= \hat{\mathbf{n}} \cdot \left(m_a^{-1}\gamma\,\hat{\mathbf{n}} + m_b^{-1}\gamma\,\hat{\mathbf{n}} + I_a^{-1}(\mathbf{r}_a \times \gamma\,\hat{\mathbf{n}}) \times \mathbf{r}_a + I_b^{-1}(\mathbf{r}_b \times \gamma\,\hat{\mathbf{n}}) \times \mathbf{r}_b\right) \\
&= v_n^- + \left(m_a^{-1} + m_b^{-1} + I_a^{-1}\hat{\mathbf{n}} \cdot (\mathbf{r}_a \times \hat{\mathbf{n}}) \times \mathbf{r}_a + I_b^{-1}\hat{\mathbf{n}} \cdot (\mathbf{r}_b \times \hat{\mathbf{n}}) \times \mathbf{r}_b\right)\gamma \\
&= v_n^- + m_{\text{eff}}^{-1}\gamma
\end{aligned}$$

where we have given the inverse of the quantity in parentheses the name m_{eff} for effective mass.

Now we know what normal velocity will result from a collision impulse of magnitude γ. The last step is to use this with the restitution hypothesis to decide what value γ should have:

$$\begin{aligned}
-c_r v_n^- = v_n^+ &= v_n^- + m_{\text{eff}}^{-1}\gamma \\
\gamma &= -\,(1 + c_r) m_{\text{eff}} v_n^-
\end{aligned}$$

This formula looks the same as for particles; all the difference has been bundled inside the effective mass.

So to deal with isolated collisions between two objects, we can do the following:

1. Detect the collision and determine the collision point $\mathbf{x}$ and normal $\hat{\mathbf{n}}$.

2. Compute the relative collision offsets $\mathbf{r}_a = \mathbf{r} - \mathbf{x}_a$ and $\mathbf{r}_b = \mathbf{r} - \mathbf{x}_b$.

3. Compute m_{eff} and then γ.

4. Compute the new velocities $\mathbf{v}_a^+ = \mathbf{v}_a^- + m_a^{-1}\gamma\,\hat{\mathbf{n}}$ and $\mathbf{v}_b^+ = \mathbf{v}_b^- - m_b^{-1}\gamma\,\hat{\mathbf{n}}$.

That's all there is to it! In the next lecture we'll talk about how to deal with many simultaneous collisions.