Physics, Part 2: Angular Effects

I just want to block the door with something heavy, so the bad guy can't get in. Is that too much to ask? I want to flip over his car with a carefully placed explosion, I want to jam the huge gears to which I'm strapped before they crush me, and I want to rig up a seesaw-type thing to catapult a nice flaming present over his castle's protective wall. You might think that my antagonist is the one stopping me from doing these things, but the person stopping me is actually the programmer behind the game's physics engine, because at the heart of each of these tasks lies an angular effect. Few games today try even to model angular effects, let alone try to get them right.

The main reason for the lack of support for angular (you might call them rotational) effects in today's games is that programmers perceive angular physics to be difficult to understand and implement. High-school physics courses (where we all learned F=*m*a) usually don't cover angular effects, and it's not immediately clear how to translate a force applied to an object into a spin for that object. While the dynamics of angular motion are slightly more difficult to understand than the dynamics of linear motion, they're not *that* much more complicated. Anybody who can implement a linear physics engine based on the material we covered in the last column will be able to implement one that supports angular effects based on the information in this column. Hopefully, once this knowledge is out there, we'll start to see games that take advantage of the expressive power of angular effects, or at the very least, let you shoot your friend's feet out from under her in a deathmatch game.

Recap

Whenever I'm writing a series of columns on a single topic, I always reread my last column before starting the latest one so I can figure out where I left off. I just got finished doing that with the first part of this series on physics, and wow, we covered a lot of ground, and without any code or references to boot! Before we get started, let's quickly review the material from last time.

Table 1 contains the important results for doing linear rigid body dynamics. Eq. 1 shows that the position vector (denoted by r), the velocity vector (v), and the acceleration vector (a) are all related by derivatives (and integrals in the opposite direction). As a reminder, we denote differentiation with respect to time with a dotted vector, so "$\dot{\mathrm{r}}$" is the same as dr/dt, and "$\ddot{\mathrm{r}}$" is the same as the second time derivative. Eq. 2 shows how force is related to linear momentum (mass times velocity), mass, and acceleration. Eq. 3 gives the definition of the center of mass, which is the point where all the masses and distances balance each other out. Eq. 4 says that the total linear momentum for a rigid body is the sum of all the momentums, which, luckily for us, simply equals the momentum of the center of mass (CM). Eq. 5 is the real gem; it uses Eq. 4 to show that the acceleration of our object's CM is related to the total force–the vector sum of all forces currently acting on our object–by a simple scalar, the total mass of the object.

Table 1. Important Equations from Part 1 of This Series

Eq. 1	Relationship of position (r), velocity (v), and acceleration (a)	$\frac{d^2\mathbf{r}}{dt^2} = \ddot{\mathrm{r}} = \frac{d\dot{\mathrm{r}}}{dt} = \frac{d\mathbf{v}}{dt} = \dot{\mathrm{v}} = \mathbf{a}$
Eq. 2	Force (F) equals the derivative of linear momentum (p), or mass (*m*) times acceleration	$\mathbf{F} = \dot{\mathrm{p}} = \frac{d\mathbf{p}}{dt} = \frac{d(m\mathbf{v})}{dt} = m\dot{\mathrm{v}} = m\mathbf{a}$
Eq. 3	Center of Mass (CM)	$M\mathbf{r}^{CM} = \sum_i m^i \mathbf{r}^i$
Eq. 4	Total linear momentum equals the momentum of CM	$\mathbf{p}^T = \sum_i m^i \mathbf{v}^i = \frac{d(M\mathbf{r}^{CM})}{dt} = M\mathbf{v}^{CM}$
Eq. 5	Total force equals the total mass (*M*) times CM acceleration	$\mathbf{F}^T = \dot{\mathrm{p}}^T = M\dot{\mathrm{v}}^{CM} = M\mathbf{a}^{CM}$

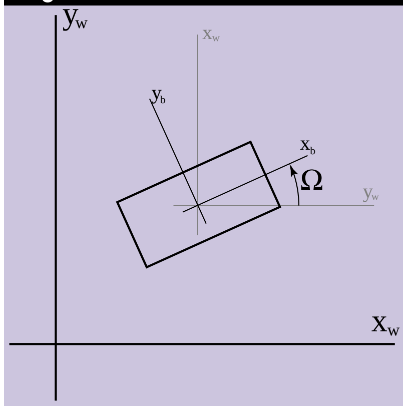
So, to summarize the results from the last column, we first find the total force on our CM by summing all the forces applied to the body (including things like gravity, the bad guy's tractor beam, the nearby explosion, our engine thrust, whatever). Then, we divide this vector sum by the total mass to get the CM acceleration, and then integrate that acceleration over time (using the numerical integration techniques mentioned at the end of the last column) to get our body's new velocity and position.

Although Eq. 5 is a nice piece of work, you'll notice that it contains no concept of where the forces act on the body, which is the key to figuring out how those forces rotate the body. Eq. 5 isn't wrong—it's exactly right for calculating the linear acceleration—we're just missing half the story. But let's start at the beginning…

What's Your Angle?

The last column ignored rotation, so we only needed the position vector and its derivatives to describe our rigid body's configuration in 2D. We now need to add another kinematic quantity, orientation (denoted by Ω, capital omega), to that configuration so we can support our angular effects. To define Ω, we need to pick a coordinate system fixed in our rigid body and a fixed world coordinate system, and specify Ω as the angular difference between them in radians, as shown in Figure 1. In the figure, x_w,y_w are the world axes, and x_b,y_b are the body axes. Ω is positive in the counterclockwise direction. At this point, it should be clear why we're learning 2D dynamics before moving up to 3D: The orientation in 2D is just a scalar (the angle between the coordinate systems in radians), while specifying an orientation in 3D is much more complicated.

Figure 1. The Definition of Ω



As our body rotates in the world, Ω changes. This change leads us to our next new kinematic quantity, angular velocity (denoted by ω, lowercase omega). In contrast with the position and its linear velocity, we don't usually signify the angular velocity by "$\dot{\Omega}$." However, we sometimes signify the velocity's time derivative, or angular acceleration—which is our final new kinematic quantity—with "$\dot{\omega}$," and sometimes with an α (lowercase alpha). Don't blame me, I don't make these rules, and every book I read has a slightly different convention. Our angular analog to Eq. 1 is

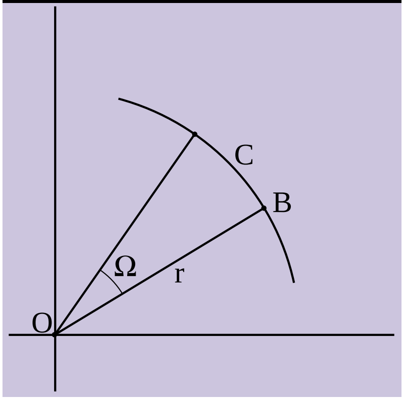
$$\frac{d^2\Omega}{dt^2} = \frac{d\omega}{dt} = \dot{\omega} = \alpha \qquad \text{(Eq. 6)}$$

Much like Eq. 1, if we differentiate ω with respect to time, we get α; and if we integrate α over time, we get ω, and so on. Clearly, as in our analytic integration example for linear movement in the previous column, if we know the angular

Chris Hecker

To properly model physics in your game, you have to understand rotational effects. See how angular momentum, torque, and other forces can be modeled in a game.

Figure 2. C=Ωr



acceleration α, we can integrate it twice to find the new orientation; but the key is we need to know α to do this.

As you might expect, our goal for this column is to derive angular analogs for each of the linear physics equations in Table 1, and then link the linear and angular equations together so we can take a given force on our object and use it to calculate the linear acceleration a, and the angular acceleration α. Finally, we can numerically integrate these accelerations to find our body's new position and orientation.

The first way we'll link the linear and angular quantities together is with a neat and not-so-obvious trick using angular velocity. When we're doing dynamics, we often need to find the velocity of an arbitrary point on our object. For example, when we cover collision response, we'll need to know the velocity of the colliding points to figure out how hard they hit each other. If our bodies aren't rotating, the velocity of any point in the body is the same; we can just keep track of the velocity of the body's CM and be done with it. However, if our bodies are rotating, then every point in them might have a different velocity. Obviously, we can't keep track of the velocity of each of the infinity of points in our rigid body, so we need a better way.

A simple way to find the linear velocity of any point inside an object uses that object's angular velocity. Let's first cover the case of a body rotating with one point, the origin O, fixed, so the body is rotating but not translating. Eq. 7 shows how to calculate the velocity for a point B on this rotating body.

$$\mathbf{v}^B = \omega \mathbf{r}_\perp^{OB} \qquad \text{(Eq. 7)}$$

Eq. 7 introduces a bunch of new notation, so let's take it apart one piece at a time. First, I'm using superscripts to denote which parameters "belong" to which points, so v^B is the velocity of point B in our body. Similarly, r^{OB} means the vector from the origin of our body O to point B. The funny upside-down T subscript is the "perpendicular operator," which takes a vector (like r in Eq. 7) and rotates it counterclockwise by 90 degrees. In other words, it creates a new vector that's perpendicular to the old vector. In 2D, the perpendicular of a vector (x,y) is just (-y,x), as you can easily verify on a piece of graph paper. I'll say more on this operator shortly. Finally, the perpendicular vector is scaled by the angular velocity ω to give the linear velocity v^B. So, in English, Eq. 7 says the velocity of a point on a rotating body is calculated by scaling the perpendicular vector from the origin to the point by the angular velocity. How in the heck did I come up with this thing? Well, I read about it in a book, but that's obviously not very illuminating, so let's prove for ourselves that it works.

We'll prove Eq. 7 does what I say it does in two stages. First, we'll prove that the magnitude of the resulting velocity vector is correct; then, we'll prove it's pointing in the right direction. To prove the first part, we'll use Figure 2. Figure 2 shows our point B moving Ω radians during the body's rotation, with the radius vector from the origin to B as *r* units long. B has moved C units of arclength on the circle, where C=Ω*r* by the definition of radians. (Radian measure is the measure of arclength scaled by the radius of the circle. The circumference of a circle is the well-known formula 2π*r*, because it's 2π [or 360 degrees] worth of arclength.)

A point's speed is its change in position over time. Thus, we can find B's speed—which is another way of saying the magnitude of its velocity vector—by differentiating the equation for its movement with respect to time. C=Ω*r* is the equation for its movement.

$$\frac{d(\Omega r)}{dt} = \frac{d\Omega}{dt} r = \omega r \qquad \text{(Eq. 8)}$$

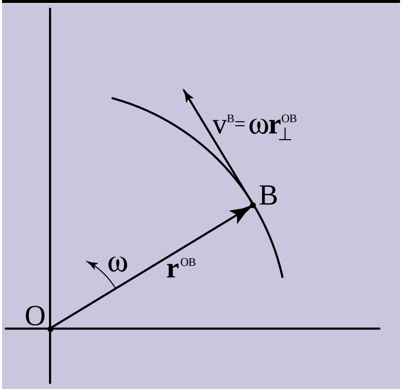
The radius drops out of the derivative because it's constant (B is simply rotating, not translating as well), and the time derivative of Ω is ω by Eq. 6. Thus, the magnitude of B's velocity vector is ω*r*.

If we look at Eq. 7, we see that it gets the magnitude correct because the perpendicular operator clearly does not effect a vector's length, and r^{OB} is the radius vector from the origin to B. We're halfway done.

To show that the direction of the velocity in Eq. 7 is correct, we'll start by convincing ourselves the velocity vector's direction must be perpendicular to the radius vector. This assumption makes sense intuitively, because a point rotating around another fixed point can only move perpendicularly to the vector between the points at any instant; it can't move closer or farther away, or the movement wouldn't be a simple rotation. We could make this assumption rigorous using a tiny bit of vector calculus, but I'm running out of space, so we'll consider ourselves convinced. (If you want to prove it to yourself, try differentiating the dot product of a fixed length vector with itself.)

Finally, we need to make sure the sign of the velocity vector is correct, since there are actually two vectors of the same length that are perpendicular to the radius: v and -v. Since we're measuring Ω in the counterclockwise direction, ω is positive when the point is rotating counterclockwise. The perpendicular operator

Figure 3. Linear Velocity from Angular Velocity



points in the counterclockwise direction relative to the radius vector. So, as Figure 3 shows, Eq. 7 checks out.

We can extend Eq. 7 to cover simultaneously rotating and translating bodies. We will consider any movement of a rigid body as a simple translation of a single point in the body and a simple rotation of the rest of the body around that point. This is known as Chasles' Theorem, for those keeping score.

Chasles' Theorem breaks up our motion into linear and angular components. We consider the origin O in Eq. 7 as the single translating point, then use ω to keep track of the rotation around O, which gives us the general form of Eq. 7.

$$\mathbf{v}^{B} = \mathbf{v}^{O} + \omega\mathbf{r}_{\perp}^{OB} \qquad \text{(Eq. 9)}$$

Eq. 9 says we can calculate the velocity of any point in a moving body by taking the known linear velocity of our body's origin and adding to it the velocity generated from the body's rotation.

A Moment-us Occasion

Now we're in a position to work on the angular analog of Eq. 2, the force equation. We'll start by defining the angular momentum, L^{AB}, of one point, B, about another point, A, as follows:

$$L^{AB} = \mathbf{r}_{\wedge}^{AB} \lozenge \mathbf{p}^{B} \qquad \text{(Eq. 10)}$$

The angular momentum of a point differs from the linear momentum of a point in that the angular version is measured from a specific position in space. That is, while linear momentum is just a property of a given point (its mass times its velocity), the angular momentum of the point must be measured from another place in the world. The superscript notation in Eq. 10 shows this. The notation L^{AB} says that the first superscript, A, is the point about which the momentum is measured, and the second superscript, B, is the point whose angular momentum is being measured. Think about an arrow from the first superscript to the second; A is "looking at" B's momentum. This arrow from A to B is the radius vector between the two points, designated by r^{AB}. So, the angular momentum of a point is the dot product of the "perpendicularized" radius vector with the point's linear momentum. This operation is called the "perp-dot product." (It's sort of the 2D analog to the 3D cross product, but that discussion will have to wait for another time.)

If you take Eq. 10 and draw out what it's doing on a piece of paper—I've drawn an example in Figure 4—you'll see it produces a number that's a measure of how much of B's linear momentum is "rotating around" A. That is, if B's linear momentum is aiming right at A or directly away from A, Eq. 10 is 0 (since r-perpendicular will form a right angle with p, and the dot product will be 0). As more of B's momentum is directed perpendicular to A, Eq. 10 produces a larger angular momentum. As you can see in Figure 4, the dot product in Eq. 10 is measuring the cosine of θ between r^{AB}-perpendicular and p^{B}, which is what you'd expect from a dot product. However, if we look at it another way, the perp-dot product is measuring the sine of ϕ between our original unperpendicularized r^{AB} and p^{B} (the sine is another clue to the similarity between the perp-dot and the 3D cross product). Whichever way we look at it, Eq. 10 is producing a measure of how much of B's linear momentum is in the "rotating-around direction" with respect to A.

In the same way we used the linear momentum's derivative to define force, we'll use the angular momentum's derivative to define force's angular twin, torque (denoted by τ, lowercase tau).

$$t^{AB} = \frac{dL^{AB}}{dt} = \frac{d\left(\mathbf{r}_{\wedge}^{AB} \lozenge \mathbf{p}^{B}\right)}{dt}$$
$$= \mathbf{r}_{\wedge}^{AB} \lozenge m\mathbf{a}^{B} = \mathbf{r}_{\wedge}^{AB} \lozenge \mathbf{F}^{B} \qquad \text{(Eq. 11)}$$

To save space, I actually cheated a bit in Eq. 11 and left out a couple of tricky steps involving the product rule for derivatives. Still, when all is said and done, the torque ends up being related to the force at a specific point by the perp-dot product.

At last, we find a dynamics equation that uses the point where a force was applied, which is ignored in the equations for linear movement. Eq. 11 uses the perp-dot product to measure how much of the force applied at point B is "rotating around" point A; that "rotating-around force" is called the torque. Eq. 11 lets us calculate the torque—and hence the angular momentum, if we integrate that torque—from an applied force and its position of application.

However, we still don't have any relationship between the torque and the kinematic angular quantities we need to spin our object around—such as the angular acceleration, angular velocity, or orientation; so we can't really do anything with our newfound dynamic quantities until we've derived a few more equations.

The Moment We've All Been Waiting For

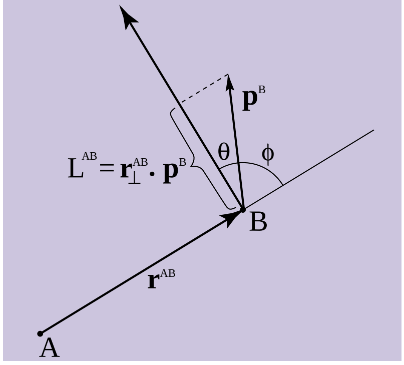
Before we can examine the relationship between the dynamic and kinematic quantities, we need to define the total angular momentum, much as we have defined the total linear momentum in Eq. 4. I didn't forget the angular equivalent of the CM in Eq. 3; it will come to us in the total angular momentum equation.

The total angular momentum about point A is denoted L^{AT} and is defined by Eq. 12.

$$L^{AT} = \hat{A}_{i}\, \mathbf{r}_{\wedge}^{Ai} \lozenge \mathbf{p}^{i}$$
$$= \hat{A}_{i}\, \mathbf{r}_{\wedge}^{Ai} \lozenge m^{i}\mathbf{v}^{i} \qquad \text{(Eq. 12)}$$

Eq. 12 is a summation of all the angular momentums of all the points, as measured from point A. On the right-hand side, I've used the definition of linear momentum to expand p into mass times velocity (*m*v) because we're going to manipulate this term to turn Eq. 12 into something more manageable. As it stands,

Figure 4. Angular Momentum



if we want to calculate the total angular momentum for our object, we'd have to sum all of the angular momentums for each of the points. For a rigid body composed of surfaces rather than separate points, we'd have to perform an integration instead of a discrete summation.

Luckily, we can simplify this calculation by introducing a new quantity, called the moment of inertia, in the same way we introduced the CM to simplify the equations for linear movement. We start by remembering that Eq. 7 gives us an alternate way of writing the velocity of a point in terms of the angular velocity. If we treat the point A in Eq. 12 as the origin in Eq. 7, and the point index i in Eq. 12 as the point B in Eq. 7, we can substitute Eq. 7 into Eq. 12 and write

$$\begin{aligned} L^{AT} &= \sum_i \mathbf{r}_\perp^{Ai} \cdot m^i \omega \mathbf{r}_\perp^{Ai} \\ &= \omega \sum_i m^i \mathbf{r}_\perp^{Ai} \cdot \mathbf{r}_\perp^{Ai} \\ &= \omega \sum_i m^i (\mathbf{r}_\perp^{Ai})^2 = \omega I^A \end{aligned} \qquad \text{(Eq. 13)}$$

I'll describe Eq. 13 one step at a time. First, we substitute Eq. 7 into Eq. 12 to get the first summation in Eq. 13. This substitution lets us write the angular momentum in terms of the angular velocity. Next, we bring the ω out of the summation because it's the same for all the points (the angular velocity is defined for the body, not the points individually), and we write the mass for point i on the left-hand side so we can see that we're really taking the dot product of the radius vector with itself. This dot product is just the radius vector's length squared. (The dot product of any vector with itself is the length squared; remember the perpendicular operator doesn't change a vector's length.) Finally, we write the letter I^A to designate the moment of inertia about point A. The moment of inertia for a 2D rigid body is a particularly nice number, because the points that make up the body can't change their mass or their distance from the measurement point. These two properties make the summation in Eq. 13 constant for each body, so we can calculate it offline before we begin. To rephrase in English, I^A is the sum of the squared distances from point A to each other point in the body, and each squared distance is scaled by the mass of each point. Much like the CM, if the body is continuous rather than made from discrete points, the summations above would turn into integrals. However, the moment of inertia would still exist and be defined the same way.

The definition of the moment of inertia about a point is a mouthful, but think of I^A as a measure of how hard it is to rotate the body about point A. For example, think about a pencil (a 2D pencil). If we measure the moment of inertia about the middle of the pencil, we get a certain value by summing the mass-scaled squared distances. However, if we measure the inertia about the tip of the same pencil, we get a much larger value, because the squared term in Eq. 13 makes the masses that are farther away (toward the eraser of our pencil) contribute much more to the value. This is saying mathematically what we all know intuitively: Turning a pencil about its center is a lot easier (read: takes less force) than turning it about one of its ends.

Finally, we're ready to provide a useful link between the angular dynamics equations and the angular kinematics equations. If we differentiate Eq. 13, we get the total torque on the left side, and on the right side we get the moment of inertia times the angular acceleration. (I^A is constant so it drops out of the derivative.)

$$t^{AT} = \frac{dL^{AT}}{dt} = \frac{d(I^A w)}{dt}$$
$$= I^A \dot{w} = I^A a \qquad \text{(Eq. 14)}$$

This equation is the angular equivalent of Eq. 5; it's basically F=*m*a for angular dynamics. It relates the total torque and the body's angular acceleration through the scalar moment of inertia. If we know the torque on our body, we can find its angular acceleration—and therefore, the angular velocity and orientation via integration—by dividing the torque by the moment of inertia.

The Dynamics Algorithm

We may not recognize it through the flurry of equations, but that's all of it. We've developed enough equations to do great 2D dynamics with arbitrary forces and torques moving and spinning our objects around. How do we use all these equations? Here's the basic algorithm:

1. Calculate the CM and the moment of inertia at the CM.
2. Set the body's initial position, orientation, and linear and angular velocities.
3. Figure out all of the forces on the body, including their points of application.
4. Sum all the forces and divide by the total mass to find the CM's linear acceleration (Eq. 5).
5. For each force, form the perp-dot product from the CM to the point of force application and add the value into the total torque at the CM (Eq. 11).
6. Divide the total torque by the moment of inertia at the CM to find the angular acceleration (Eq. 14).
7. Numerically integrate the linear acceleration and angular acceleration to update the position, linear velocity, orientation, and angular velocity (see last issue).
8. Draw the object in the new position, and go to Step 3.

There are only two steps in the above algorithm that I haven't yet explained. First, how does one calculate the moment of inertia in Step 1 for a continuous object? Second, how do you figure out the forces on an object for Step 3? The answer to the first question can be found in the sample program referenced at the end of this article (you perform an integration over the surface of the object). Most dynamics books have the moments of inertia for common shapes listed in the back, so you don't usually have to derive them from scratch.

The answer to how to compute the forces in Step 3 depends on the application, but a few general guidelines apply. First, forces like gravity that always point in the same direction (down, in gravity's case) don't induce torques on an object since they pull on all points at the same time and in the same direction; thus, we just apply those forces directly to the CM. A spring-like force applied to a specific point on an object will induce torques, so we handle it normally. As we saw in the last issue, drag is just a force directed in the opposite direction of your velocity. You could do a simple drag model and just apply the force to the CM, or you could figure out which parts of your object would have drag and apply specific drag forces to those parts, which might induce torque on your object. The forces experienced during a collision are slightly more complicated, and will have to wait until the next column. Forces from rocket engines would probably be treated as forces with a point of application. (That way, if one of your engines fails, you'll start to spin unless you adjust your rudder to provide another force to counteract the torque!) If you have something like a tractor-beam, should it act like gravity and be torque-free, or should it be applied at a specific point on the object so the object turns toward the beam as it's pulled? You'll have to decide that. The key is not to be afraid of experimenting with different forces calculated in different ways—now that you've got a real 2D dynamics simulator, you can try all sorts of forces.

I've placed a bunch of references and some code on my Web site because there's no more room left here. The sample app implements the 2D dynamics algorithm on some objects attached by a spring; they spin and move around, and even collide with walls with rotations, which we'll cover next time. Check out http://ourworld.compuserve.com/homepages/checker for a list of references and the sample application for Win32 and Macintosh. ■

Every once in a while, Chris Hecker experiences a moment of inertia, but it usually passes pretty quickly. Forces may be applied at checker@bix.com.

OOPS!

I got a great e-mail from Jan Vondrak (JVON4518@barbora.mff.cuni.cz) the other day. Jan pointed out, much to my chagrin, that the final assembly code for the texture-mapping series had a big performance flaw in it (in addition to the [illegible] list at the top of the file): Very soon after issuing the `fdiv` that ostensibly overlaps with the rasterization loop, I issue an `imul`. Well, on the Pentium, `imul` uses the floating point unit, so it's going to stall on the `fdiv`, and I won't overlap. Oops! I was so rushed just getting the code working that I didn't notice this bug. I moved the `fdiv` below the `imul`s and got a speedup, and as the comment in the file says, that code is fertile ground for optimization. Thanks to Jan for pointing this out!