Attention:

Chris Hecker
definition six, incorporated
checker@d6.com
http://www.d6.com/users/checker

PS. The email address at the end of the article is incorrect. Please use checker@d6.com for any correspondence.

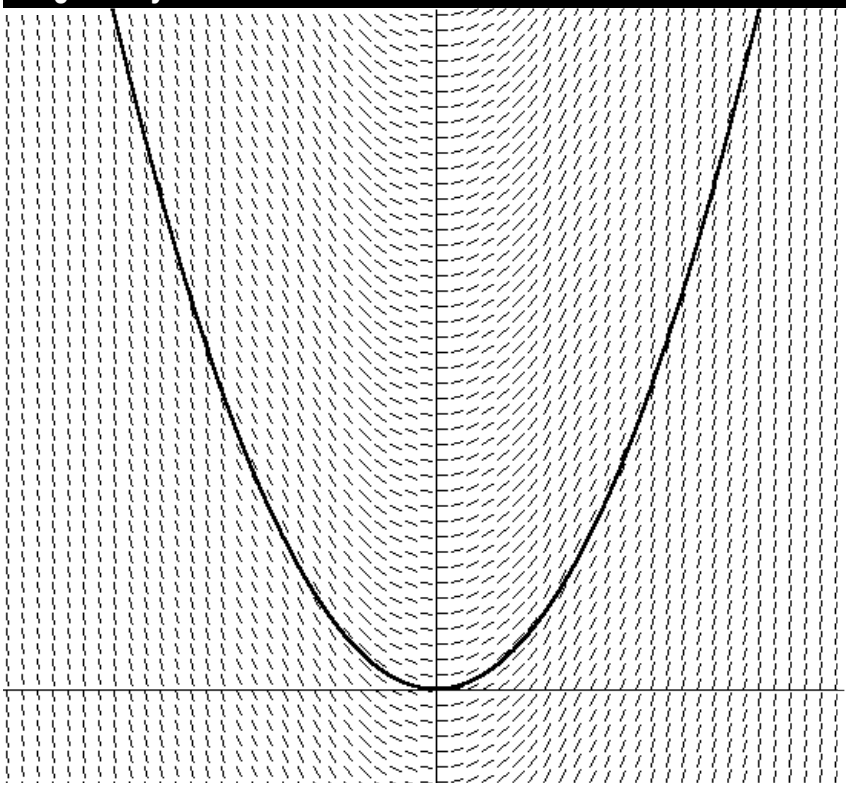
Physics, The Next Frontier

No doubt about it: each year games become graphically more realistic. Everybody's doing (or at least showing screenshots of) texture-mapped 3D game worlds these days, and, when the hardware people finally get their act together, every developer will be able to draw zillions of perspectively correct textured and shaded polygons per second. Technically speaking, what will be left to do for high-end games? Will every developer with a copy of "Learn to Use 3D Hardware in 21 Days" be able to write an impressive game?

Not by a long shot. High-end developers will continue to raise the bar in many different technologies, like graphical database complexity, artificial intelligence, and networking. While these are indeed important topics, we can't really discuss any of them in depth without going into game-specific details. However, there is one generally applicable technology I think will become a key differentiating factor in the near future: physics.

Here's an example: remember those huge rotating gears in one of the early shareware levels of Duke Nukem 3D? Imagine if a general physics engine was controlling them instead of an animation loop. Suddenly the gears become more than just eye candy as one comes off its axle and rolls down the hall after you, Indiana-Jones style. Or imagine shooting a gear with a missile, causing it to roll down the hall and crush your friend who was about to frag you from behind! A real physics engine makes situations like these possible.

Figure 1. $y=x^2$



The physics simulation is also what makes a game world feel solid—it puts the "there" there, if you know what I mean. All the graphics wizardry in the world won't help players immerse themselves in your game if they interpenetrate each other or the walls of the level, or if they don't feel like they have any mass or momentum. The original animators at Disney discovered that this feeling of mass is a large part of what set apart the believable animation from the bad. According to the epic book *Disney Animation: The Illusion of Life* (Abbeville Press, 1981), by Frank Thomas and Ollie Johnston, Disney animators even hung a sign around the studio to constantly remind themselves: "Does your drawing have weight, depth, and balance?"

But doesn't almost every game already have a physics engine? Sure, it keeps your car from falling out of the world through the track, it keeps your characters from floating away when they jump, and it knocks your ship to the side when a missile explodes nearby. However, most physics engines in today's games are pretty weak, doing just enough to keep that car from falling out of the world, but not enough to take the game to the next level of interaction—where a wrecked car's

debris might explode onto the track, careening into the wall and other cars, tires rolling into oncoming traffic.

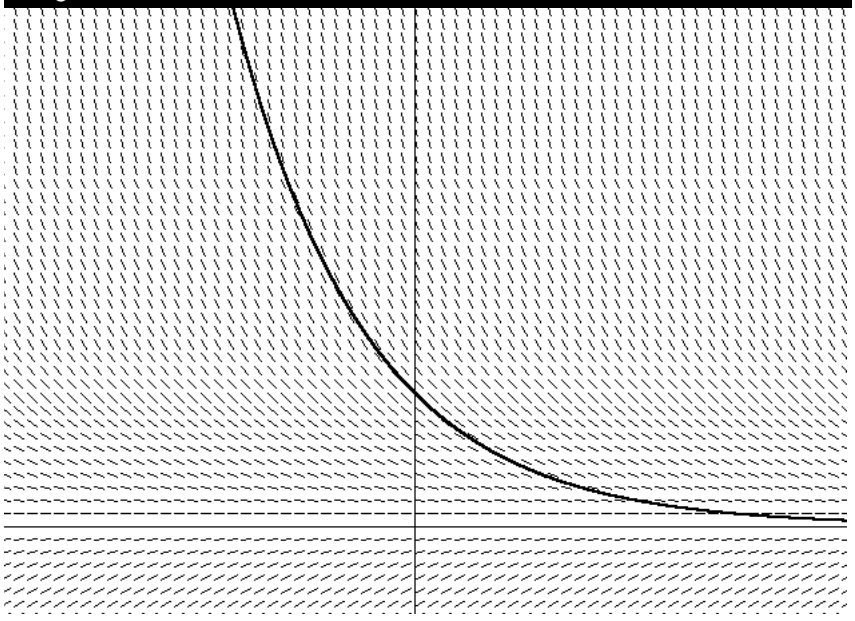
Other often ignored physical possibilities include everything from simple rotational effects induced by being hit off center, to having the creatures in the game be self-balancing and motivating—rather than based on static animations—so they can react to new physical stimuli. I think most developers ignore these possibilities because they don't understand the math behind physics and have been too busy writing perspective texture mappers to learn it. The onslaught of 3D hardware will take care of the latter issue, and the new series I'm starting with this article will try to take care of the former. By the time we're done, you'll know enough to write a physics engine that immerses players in your game, either through extreme physical realism or through fanciful but consistent surrealism.

A word of caution: physics is math—you can't separate the two and still get interesting work done. Before this scares people away, let me point out that not only is the math behind physics totally elegant and beautiful, it's also applied. That is, it's not abstract math for math's sake. Each equation we use has real physical meaning. We create the equations from the physical model, and in return the equations tell us how that model behaves over time.

Mass-ive Undertaking

Physics is a vast field. We're actually only interested in a small subset of it called dynamics, and even more specifically, rigid body dynamics. Dynamics

Figure 2. dv/dt = -v/m



Chris Hecker

We've been faking physics in games for a while. Now, technology is advancing to the point where implementing a real physics engine within a game is possible. Hecker's new series explores how.

can be defined most easily in terms of a closely related field, kinematics—the study of movement over time. Kinematics doesn't concern itself with what's causing movement or how things get where they are in the first place, it just deals with the actual movement itself. Dynamics, on the other hand, is the study of forces and masses that cause the kinematic quantities to change as time progresses. How far a baseball travels in 10 seconds if it's traveling 50 kilometers per hour in a straight line is a kinematics problem; how far a baseball travels in the earth's gravitational field if I smack it with a bat is a dynamics problem.

The "rigid body" part of rigid body dynamics refers to constraints we place upon the objects we're simulating. A rigid body's shape does not change during our simulation—it's more like wood or metal than jello. We can still create articulated figures, such as a human being, by building each section of the figure from a rigid body and putting joints between them, but we won't account for the flexing of bones under strain or similar effects. This will let us simplify our equations while still allowing for interesting dynamic behavior.

Even with our tight focus, rigid body dynamics will take a series of articles to explain. We're going to start our journey by learning the basics of programming a computer to move a 2D rigid body around under the influence of forces. I explicitly say, "program a computer," because in addition to the equations we'll develop for the kinematics and dynamics, we'll also learn how to solve these equations in our programs using floating-point math, which is a vital subject all to itself. I say, "a 2D rigid body," because we're going to stick with two dimensions for the next article or so. The principles—and in fact most of the equations—carry across to 3D, but certain things are simpler in 2D, so we'll get comfortable there before moving up a dimension. In future articles, we'll learn about handling rotational effects, collision detection and response, and of course how to do all this in three dimensions. Enough about what we're going to do, let's get started!

Derivative Work

This may come as a surprise to you, but you actually can't directly move an object by pushing on it. I know, you're thinking about proving me wrong by pushing this magazine into the trash for printing such nonsense, but it's true: pushing on the magazine does not directly affect its position. In fact, pushing doesn't even directly affect its velocity. What pushing does directly affect, however, is the magazine's acceleration, and this fact is one of the most important findings in the history of science.

In order to use this amazing fact to do anything interesting, we first need to talk about the relationship between position, velocity, and acceleration. It turns out these quantities are very closely related (as you probably know): velocity is the rate of change of position over time, and acceleration is the rate of change of velocity. The primary tool for studying these changes in time is calculus. While you might be able to pick it up as we go along, I'll assume you know some calculus. We're going to use only simple scalar and vector calculus (derivatives and integrals), but it won't hurt if you're familiar with the subject as a whole. For reference, my favorite calculus textbook is *Calculus with Analytic Geometry* by Thomas and Finney (Addison-Wesley, 1996).

Position, velocity, and acceleration are the kinematic quantities we care about in this article. The position of a rigid body in 2D is obviously an X,Y pair denoting the world coordinates of some known point on the body. The derivative of the position vector is the velocity vector for that point, and it tells us what direction the point is moving (and the body if we ignore rotation, which we are for now) and how fast it's going. Vector calculus is just scalar calculus on each element of the vector, so the derivative of the X element of the position is the X element of the velocity, and so forth. We denote the position vector with r and the velocity vector with v or with the "dotted" position vector (in general, a dot means differentiated with respect to time, a double dot means twice differentiated, and so on):

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} = \dot{\mathbf{r}} \qquad \text{(Eq. 1)}$$

On the contrary, if we integrate the velocity vector over time, it tells us how the position vector changed over that time.

Acceleration is handled similarly; it's the derivative of velocity, or the second derivative of position:

$$\frac{d^2\mathbf{r}}{dt^2} = \ddot{\mathbf{r}} = \frac{d\dot{\mathbf{r}}}{dt} = \frac{d\mathbf{v}}{dt} = \dot{\mathbf{v}} = \mathbf{a}$$

(Eq. 2)

Integrating the acceleration over time gives the velocity, and twice integrating the acceleration gives the position.

These kinematic relationships tell us that if we can find the acceleration on an object, we can integrate it with respect to time to get its velocity and position. As we'll see, we perform this integration numerically in our simulation code and come out with a new position for our rigid body each frame. Voila, animation!

Here's a short 1D example we can analytically integrate. Let's say we want to find the change in our position over the time period from the end of last frame to the current time so we can draw our current position. Let's further say we know the acceleration on our rigid body during this time was a constant 5 units/sec^2. We'll use the time since the end of last frame as the integrating variable, t:

$$\mathbf{v}(t) = \int \mathbf{a}dt = \int 5dt = 5t + C$$

(Eq. 3)

The above equation shows us the velocity as a function of the time since the last frame. We discover the constant of integration, C, is the initial velocity at the beginning of this integration period by plugging in t=0:

$$\mathbf{v}(0) = 5(0) + C$$
$$v_0 = C$$
$$\mathbf{v}(t) = 5t + v_0 \qquad \text{(Eq. 4)}$$

Now we'll integrate our velocity equation to find the position (again solving for the constant of integration):

$$\mathbf{r}(t) = \int \mathbf{v}(t)dt = \int 5t + v_0 dt = \frac{5}{2}t^2 + v_0 t + r_0$$

(Eq. 5)

So, Eq. 5 says we can calculate the current position under the given acceleration if we know the initial position and velocity (which we assume we have from the end of the last frame) and the time elapsed. We plug in the time, and out pops the current position. We'll also want to plug the time into Eq. 4 to calculate the ending velocity so we can use it as an initial condition for the next frame.

May The Force Be With You

Now that we have an idea of how to integrate kinematic equations to get animation, we need to determine the right accelerations to use in the first place. This is where dynamics comes in. Remember how I said pushing on something only directly affects its acceleration? Well, pushing is just a euphemism for applying a force—one of the two key quantities in dynamics—and we can turn to Newton to see how forces affect accelerations. Newton's laws relate force, F, to the derivative of the mass—the second key dynamical quantity—times the velocity. The mass times the velocity is called the linear momentum, denoted by p:

$$\mathbf{F} = \dot{\mathbf{p}} = \frac{d\mathbf{p}}{dt} = \frac{d(m\mathbf{v})}{dt} = m\dot{\mathbf{v}} = m\mathbf{a}$$

(Eq. 6)

The mass is constant for the speeds we care about, so it drops out of the derivative in Eq. 6, and we get the famous F=ma (although I believe Newton originally stated the definition of force as the derivative of momentum).

If we were only dealing with single point masses, Eq. 6 would be all we'd need to do dynamics. For a given applied force, we find the acceleration of the point by dividing the force by the mass. This gives us the acceleration to use in our integration, and so we can solve for the movement as in our example above. However, we're dealing with rigid bodies with mass distributed over their area (and volume when we go to 3D), so we need to do a bit more work.

First, let's picture our rigid body as a set of point masses. We define the total momentum, $\mathbf{p}^T$, of the rigid body as the sum of all the momentums of all the points that make up the body (I'm using superscripts to denote which quantities belong to which points):

$$\mathbf{p}^T = \sum_i m^i \mathbf{v}^i \qquad \text{(Eq. 7)}$$

We can greatly simplify the dynamic analysis of rigid bodies by introducing a point called the center of mass (CM). The vector to the center of mass is the linear combination of the vectors to all the points in the rigid body weighted by their masses, divided by the total mass of the body, M:

$$\mathbf{r}^{CM} = \frac{\sum_i m^i \mathbf{r}^i}{M} \qquad \text{(Eq. 8)}$$

Using this definition of the center of mass, we can simplify Eq. 7 by multiplying both sides of Eq. 8 by M, differentiating both sides, and then substituting the result into the Eq. 7:

$$\frac{d(M\mathbf{r}^{CM})}{dt} = \sum_i \frac{d(m^i \mathbf{r}^i)}{dt} = \sum_i m^i \mathbf{v}^i = \mathbf{p}^T$$

(Eq. 9)

The right hand side of Eq. 9 is just the total momentum by definition in the Eq. 7. Now look at the left hand side: it is the velocity of the center of mass times the mass of the whole body, so bringing the right hand side across gives us:

$$\mathbf{p}^T = \frac{d(M\mathbf{r}^{CM})}{dt} = M\mathbf{v}^{CM} \qquad \text{(Eq. 10)}$$

Eq. 10 says the total linear momentum is equal to the total mass times the velocity of the center of mass, meaning there's no need to do the summation in Eq. 7 to find the momentum as long as we know the total mass and the location and velocity of the center of mass. For continuous rigid bodies all the finite summations above turn into integrals over the body, but the center of mass still exists and simplifies the total momentum equation down to Eq. 10, so we don't have to care—for the purposes of finding the linear momentum we can treat all bodies as a single point mass and velocity.

Similarly, the total force is the derivative of the total momentum, so the concept of the center of mass can be used to simplify the force equation in the same way:

$$\mathbf{F}^T = \dot{\mathbf{p}}^T = M\dot{\mathbf{v}}^{CM} = M\mathbf{a}^{CM} \qquad \text{(Eq. 11)}$$

In short, Eq. 11 tells us we can treat all the forces acting on our rigid body as if their vector sum is acting on a point at the center of mass with the mass of the entire body. We divide a force (say, gravity) by M to find the acceleration of the center of mass,

and then we integrate that acceleration over time to get the velocity and position of our body. Since we're ignoring rotational effects until the next article, we now have all the equations we need to do rigid body dynamics. Note that Eq. 11 doesn't contain any information about where the forces were applied to the body. That information drops out when dealing with linear momentum and the center of mass, and we just apply all forces to the CM to find its acceleration. When we calculate rotation under forces in the next article, we'll see how the force application position is used.

Ode to Joy

At this point, we could run through another analytical integration example using Eq. 11 to calculate the acceleration of our center of mass instead of arbitrarily picking the value 5. However, for non-toy problems, analytical integration usually isn't an option because the integrand is too complex, so we end up doing what's called *numerical integration of ordinary differential equations* (ODEs). Wow, now that sounds like real math! Once you've learned this stuff, it's time to ask for a raise. Luckily, numerical integration of ODEs isn't quite as complicated as it sounds. To figure out what it means, let's take the phrase apart from the inside out.

First, a differential equation is an equation where derivatives of the dependent variable appear in the equation in addition to the dependent variable itself and the independent variable. That's a mouthful, but here are some examples: if we have an equation for a time varying 1D force like f = 2t, f is the dependent variable and t is the independent variable; f's value depends on t's value. However, what if the equation for the force on our body depends on the velocity of our body? Air friction is a force like this—the faster the plane goes, the more air friction it encounters. Again in a 1D example, what if f = -v, meaning the friction force decelerates our body at a rate proportional to our velocity? Now we have a problem, because if we solve for the acceleration by writing f = ma = -v and then divide through by m, we get (remember the acceleration is the derivative of the velocity):

$$a = \frac{dv}{dt} = -\frac{v}{m}$$

(Eq. 12)

This is a differential equation because the equation for the velocity has the derivative of the velocity in it. Eq. 12 is called an ordinary differential equation because it contains only ordinary derivatives of the dependent variable (as opposed to partial derivatives, which create PDEs, which we won't talk about).

Now for the next part of our phrase: integration. How do we integrate dv/dt to find v in terms of t when the equation

for dv/dt has v in it already? It sounds impossible, but actually almost every equation in physics is a differential equation, so ODEs have been studied a lot. Differential equations pop up in physics so much because very often the rate of change of a quantity depends on the value of the quantity. For example, we already said that the deceleration (the rate of change of velocity) induced by the force of air friction is dependent on the velocity. Other physical examples include cooling (the rate of heat loss depends on the current temperature) and radioactive decay (the rate of decay depends on how much radioactive material is present).

The final word in our phrase, numerical, is our savior. I say this because the theory of analytically integrating differential equations, even ordinary ones, is huge and pretty complicated. However, by a strange twist of fate, integrating ODEs numerically on a computer is actually relatively easy to understand. In the space I have left I'm going to describe the simplest numerical integrator, Euler's method, and leave improving it to a later article.

Almost all numerical integrators, but none so blatantly as Euler's method, are based on the plain old calculus definition of the first derivative as a slope: dy/dx defines the slope of y with respect to x. For example, if we have the linear equation $y = 5x$, then $dy/dx = 5$, meaning the slope is a constant 5 for all values of x, as you'd expect for a line. A slightly more complex example is the parabola $y = x^2$. In this case, $dy/dx = 2x$, which is a function defining a new slope at each x coordinate. I've graphed $y = x^2$ in Figure 1. In addition, I've also overlayed the vector field of the slopes in Figure 1, by drawing the solution to the slope equation, $dy/dx = 2x$, as a short vector at each coordinate on the grid. Notice how the vector field is tangent to the parabola at all points—this is the definition of satisfying the equation $dy/dx = 2x$. You should also notice that there are a lot of different parabolas that would satisfy the vector field tangency, each one translated on the y axis a bit. Each of these parabolas is generated by using a different value for the constant of integration you get when you integrate $dy/dx = 2x$. The parabola I drew corresponds to a 0 constant of integration, since $y = x^2$. If I chose 1 for the constant, I'd get $y = x^2 + 1$, which is an identical parabola translated up by 1 unit in y.

Now think about what would happen if you didn't know the vector field in Figure 1 defined a parabola, and you just plopped yourself down somewhere on the grid. Well, if you are going to satisfy the slope equation, you have to follow the vector field at each point, so along you go, changing direction as the vector field changes direction. Wouldn't you know it—after a short bit you've traced

out a parabola, or at least part of one, depending on where you started. You may not realize it, but you just integrated the equation for the vector field. You found a specific parabola (which one depends on where you started, or your initial condition) using only the equation for the derivative (evaluating dy/dx is how you followed the vector field).

Doing the same thing for a real differential equation is just as easy. For a differential equation of the type dy/dx = f(x,y), the definition of the derivative dy/dx as a slope means f(x,y) defines a slope for each coordinate in the x,y graph. If you graph the vector field given by dy/dx = f(x,y) you can follow it, just like you did for the parabola, by sampling the derivative at each point and going in that direction. Figure 2 shows the vector field for Equation 12, our air friction equation, with velocity as the vertical axis and time as the horizontal axis (I arbitrarily picked m=1 for this graph). It also shows one of many possible solution curves. You can see that if you pick an initial position in the graph (which corresponds to an inital velocity in the equation), as time passes your velocity will decay down towards zero as friction robs you of speed. You can also see that the rate at which your velocity is decaying depends on the current value of your velocity: the faster you're going, the faster it decays. This makes sense, since we picked Equation 12 to give us exactly this result.

Doing these integrations numerically is quite similar to doing them on a graph. Euler's algorithm for numerical integration simply follows the vector field from an initial position by evaluating the derivative equation (-v/m for our air friction example) to find the slope at the current point, and then stepping forward in time by a fixed amount, h, on that tangent line. It then evaluates the derivative at the new position to get a new slope, and takes another time step:

$$y_{n+1} \approx y_n + h\frac{dy_n}{dx}$$

Or, explicitly in terms of our air friction equation:

$$v_{n+1} \approx v_n + h\frac{-v_n}{m}$$

Obviously, Euler's method accumulates a little error each time it steps, since the real vector field (and therefore the solution curve) is curving away at any point and Euler's algorithm is just stepping along the tangent line. But if the stepsize, h, is small enough, Euler does okay. We'll discuss this error more in the future.

That's really all there is to numerically integrating with Euler's method. However, you might be wondering how we integrate the velocity to get the position now that we're numerically integrating the acceleration to get the velocity. We just use Euler's method again to integrate dr/dt = v at the same time we integrate dv/dt = a, alternating as we go. We end up with two coupled ordinary differential equations (another good one for that raise):

$$\mathbf{v}_{n+1} \approx \mathbf{v}_n + h\dot{\mathbf{v}}_n = \mathbf{v}_n + h\frac{\mathbf{F}_n}{M}$$

$$\mathbf{r}_{n+1} \approx \mathbf{r}_n + h\dot{\mathbf{r}}_n = \mathbf{r}_n + h\mathbf{v}_n$$

This gives us an iterative algorithm for computing the position from some arbitrarily wacky force on our object (which could depend on the velocity as we've seen, or time, or even on the position of the body and other bodies, or all at once!). Euler's method doesn't care what the force looks like, as long as you can compute it at each step. Euler treats the value of the force over the mass as a slope, and steps merrily along.

I'm out of space, so I don't have room to give references. Next time I'll list some great books, and we'll get into how to do rotations with rigid bodies. ■

Although his body is not quite as rigid as he'd like, Chris Hecker has a dynamic personality. If forced, he'll answer e-mail at checker@bix.com.