Towards a simpler, stiffer, and more stable spring

Written by Michael Schmidt Nissen

Version 0.5.2, December 18th 2014

Quick definition guide

x	Displacement	θ	Angular displacement
v	Velocity	ω	Angular velocity
a	Acceleration	α	Angular acceleration
f	Force	τ	Torque
m	Mass	I	Moment of inertia
k	Stiffness coefficient	Δt	Delta time (time step)
d	Damping coefficient	j	Impulse

The trouble with springs

The damped spring based on Hooke's law of elasticity is often regarded as a black sheep in the field of game physics, and some papers and websites even warn the reader against implementing them. There are a couple of good reasons for this.

The most straight-forward reason is that springs are notoriously difficult to tune. It can be very time consuming to adjust the spring stiffness and damping coefficients in order to make a simulation behave just right. For large systems of interconnected particles and springs the task can be daunting. If a variable is changed, for instance a spring coefficient, the mass of a particle, or simulation time-step, you might have to re-tune the entire system from scratch.

A more profound reason for avoiding springs is that they are potentially unstable. When increasing the spring coefficients above an unknown threshold, the simulation will start to oscillate chaotically and explode in all directions. Everyone who ever worked with springs has experienced this frustrating phenomenon. The problem is that there’s no good method

to determine in advance if a given spring system will explode or not. Thus, the programmers or game designers ability to balance a spring system often rely on experience and gut feeling, which makes it look more like black magic than actual physics.

In this article I'll introduce a modified spring equation that is easy to tune, displays maximum stiffness and damping, and is guaranteed to keep the spring system stable under all conditions. The math behind the equation is surprisingly simple, which makes me wonder why it doesn't appear to have been mentioned or implemented earlier.

Making a better spring

The improved spring equation stems from trying to answer a simple question: Is it possible to make a spring that reaches its rest state in just one simulation loop? It turns out that the answer is yes! This wouldn't be possible in the real world, since it would require infinitely high stiffness and damping coefficients or zero mass. But in the not-quite real world of discrete, time-stepping physics simulation, this is achievable.
To explain how this is done, let's look at a one dimensional harmonic oscillator with a damped spring attached to a fixed point in one end and to a moving body in the other. Assume that the body is displaced distance **x** from its rest position. Now let us try to find the exact amount of force required to move the mass this distance in one simulation loop.

Distance can be expressed as velocity multiplied by time

$$x = -v\Delta t \qquad (1)$$

The minus sign simply means that we are going in the opposite direction of the displacement. Velocity can be expressed as acceleration multiplied by time

$$x = -a\Delta t^2 \qquad (2)$$

Newton's 2nd law of motion state that acceleration equals force over mass

$$x = -\frac{f}{m}\Delta t^2 \qquad (3)$$

Now we can isolate force

$$f = -\frac{m}{\Delta t^2}x \tag{4}$$

And since the spring force in Hooke's law of elasticity is defined as

$$f = -kx \tag{5}$$

It becomes immediately clear that

$$k = \frac{m}{\Delta t^2} \tag{6}$$

Which is the exact spring stiffness coefficient value required to move the particle back to its rest position in one simulation loop. However, since we are not doing anything to stop the particle, it will keep oscillating back and forth through the rest position. We need to add damping, which is done with the second part of the spring equation. By calculating the amount of force needed to zero out the particle's velocity we can make it stop at the rest position.

Velocity is equal to acceleration multiplied by time step

$$v = -a\Delta t \tag{7}$$

This is equal to force over mass

$$v = -\frac{f}{m}\Delta t \tag{8}$$

When isolating force we get

$$f = -\frac{m}{\Delta t}v \tag{9}$$

And since the damping force is defined

$$f = -dv \tag{10}$$

We immediately see by inspection that the damping coefficient is

$$d = \frac{m}{\Delta t} \tag{11}$$

This is the exact damping coefficient value needed to stop the particle in one simulation loop. Now we can write the complete damped spring equation.

$$f = -\frac{m}{\Delta t^2} x - \frac{m}{\Delta t} v \tag{12}$$

This spring equation has some very interesting properties. The first thing we notice is the lack of coefficients. We've simply replaced **k** with **$m/\Delta t^2$** and **d** with **$m/\Delta t$**. When implementing the equation we see that it really does work! The spring reaches equilibrium in one loop, completely independent of particle position, velocity, mass, and simulation time-step. This is the stiffest possible spring, and it displays behavior more similar to a rigid constraint than a soft, bouncy spring. The equation also has another interesting property. It simply cannot blow up no matter what values you feed it. Practically speaking, we can regard the spring as being unconditionally stable.

Re-introducing the spring coefficients

Now we have a really powerful spring equation. It is easy to implement, very stable, and reaches its rest state in just one loop. But in our quest towards a better spring it has lost its springiness. We need to get the softness and bounciness back again, and for this purpose we will re-introduce the spring and damping coefficients. To avoid confusion, the new coefficients are named $\mathbf{C_k}$ and $\mathbf{C_d}$. While the original coefficients could represent any positive numerical value, these both lie in the interval between zero and one.

$$f = -\frac{m}{\Delta t^2} C_k\, x \; - \frac{m}{\Delta t} C_d\, v \qquad [0 \leq C_k, C_d \leq 1] \tag{13}$$

As we can see, the new coefficients are simply the fraction of completely rigid behavior we would like the spring to display. Soft, bouncy springs would usually have values in the range of 0.001 – 0.00001. In the other end of the scale, values of just 0.1 – 0.01 is enough to display rigid behavior. Setting both values to 1.0 would of course still satisfy the constraint in one loop.

Please notice that spring behavior is determined exclusively by these two coefficients. Particle mass or simulation time-step has no influence on how the spring behaves, and changing them wouldn't make it necessary to tune the system again!

Interestingly, the spring will get *less* rigid and *less* stable if we increase $\mathbf{C_k}$ or $\mathbf{C_d}$ above 1. If we keep increasing either or both of the coefficient values, the system will start to oscillate chaotically, and at some point it will explode. In other words, we have determined the exact upper limit for the two spring coefficients, which we define

$$k_{max} = \frac{m}{\Delta t^2} \tag{14}$$

$$d_{max} = \frac{m}{\Delta t} \tag{15}$$

This allows us to simplify the spring equation

$$f = -k_{max}\, C_k\, x - d_{max}\, C_d\, v \tag{16}$$

There is an important conclusion to be drawn from this. It is a misunderstanding to think that spring stiffness and damping can be increased towards infinity by simply increasing **k** and **d**. A system of very stiff springs doesn't necessarily blow up because the integration algorithm can't handle it, but because the coefficients might have been set to a value that simply does not make sense.

Two connected particles with different mass

It is only slightly more complicated to constrain two free-moving particles with the improved spring. To do this, we need to introduce the concept of *reduced mass*. This is a quantity that can be used to compute interactions between two bodies as-if one body was

stationary, which allows us to use the equation we've already derived. The reduced mass for two particles with mass $\mathbf{m_1}$ and $\mathbf{m_2}$ is defined

$$m_{red} = \frac{m_1 m_2}{(m_1 + m_2)} \tag{17}$$

Since the inverse mass quantity is often already pre-computed for other purposes, it can also be useful to define reduced mass

$$m_{red} = \frac{1}{(\frac{1}{m_1} + \frac{1}{m_2})} \tag{18}$$

For two connected particles, the maximum coefficient values are

$$k_{max} = \frac{m_{red}}{\Delta t^2} \tag{19}$$

$$d_{max} = \frac{m_{red}}{\Delta t} \tag{20}$$

When replacing mass with reduced mass we get

$$f = -\frac{m_{red}}{\Delta t^2} x - \frac{m_{red}}{\Delta t} v \tag{21}$$

This spring equation will bring the two connected particles to equilibrium distance in one loop and make them stand still relative to each other. However, since energy and momentum are conserved, the particles may rotate around each other, which will make the bodies come to rest at a larger distance, depending on how fast they rotate.

Angular springs

The spring equation can also be used for angular springs, also commonly named rotational or torsional springs. Rather than keeping two particles at a fixed distance by applying opposite equal forces, the angular spring will try to keep two rotating bodies at a fixed angle by applying opposite equal torques. The equation introduces the concept *reduced moment of inertia*, which is calculated in a similar way as reduced mass

$$I_{\text{red}} = \frac{I_1 I_2}{(I_1 + I_2)} \tag{22}$$

Or alternatively

$$I_{\text{red}} = \frac{1}{(\frac{1}{I_1} + \frac{1}{I_2})} \tag{23}$$

The maximum coefficient values for angular springs are

$$k_{\max} = \frac{I_{\text{red}}}{\Delta t^2} \tag{24}$$

$$d_{\max} = \frac{I_{\text{red}}}{\Delta t} \tag{25}$$

When replacing the variables of linear motion with those of angular motion we get

$$\tau = -\frac{I_{\text{red}}}{\Delta t^2} C_k \, \theta - \frac{I_{\text{red}}}{\Delta t} C_d \, \omega \tag{26}$$

This spring equation will keep two rotating bodies at any given rest angle. If both coefficients are set to 1, the constraint will be solved in one loop. The spring allows for angular displacements larger than one full turn, making it possible to "wind up" bodies like the coil spring of a mechanical clock.

Impulse-based springs

Today a lot of physics engines and simulation methods are based on impulses - direct changes in velocities - rather than forces and acceleration. The linear and angular spring equations described above works equally well if we redesign them to work with impulses. Here are the equations without further ado

$$j_{\text{linear}} = -\frac{m_{\text{red}}}{\Delta t} C_k \, x - C_d \, v \tag{27}$$

$$j_{\text{angular}} = -\frac{I_{\text{red}}}{\Delta t} C_k \, \theta - C_d \, \omega \qquad (28)$$

The impulse based springs work just like the force based ones already described. The only notable difference is that when dealing with continuous forces that change gradually over time – like in springs – the impulse based equations are limited to 1st order of magnitude accuracy when it comes to conserving energy and momentum. Force based equations are, on the other hand, only limited by the integration algorithm, and can return results that are much more accurate.

Limitations and pitfalls

When connecting a multitude of springs and particles into larger bodies, we run into the same trouble as any other type of distance and velocity constraint. Rather than cooperating, the constraints tend compete against each other, and this spells trouble. When a spring moves two particles to satisfy distance and velocity, it usually means dissatisfying one or several other springs. It is outside the scope of this article to dig deeply into this problem, but the author would like to provide a bit of advice on how to prevent the worst disasters.

When two or more springs are connected to the same particle, which is the case in any kind of rope, mesh, or deformable body, setting the coefficients to the maximum value of 1.0 will lead to stability problems. Although the spring equation is unconditionally stable when particles are connected to just one spring, this is sadly not the case for higher number of springs. The author of this article has after some lengthy tampering worked out that a safe upper limit for both the stiffness and damping coefficient is

$$\mathrm{C}_{\max} \approx \frac{1}{(\mathrm{n}+1)} \qquad (29)$$

Where **n** denotes the highest number of springs attached to any of the two particles connected by the spring. So for example, in a rope where any particle is connected by at most two springs, $\mathbf{C_k}$ and $\mathbf{C_d}$ can both safely be set to 0.33, and in a square mesh, where any particle is at most connected by four springs, they can be set to 0.2.

Reference

http://en.wikipedia.org/wiki/Reduced_mass