Spring Physics

How to simulate springs and dampers

Posted by Glenn Fiedler (http://web.archive.org/web/20181107181500/https://gafferongames.com/about) on Friday, September 3, 2004

Introduction

Hi, I'm Glenn Fiedler (http://web.archive.org/web/20181107181500/https://gafferongames.com/about) and welcome to **Game Physics (http://web.archive.org/web/20181107181500/https://gafferongames.com/categories/game-physics/)**.

In the previous article (http://web.archive.org/web/20181107181500/https://gafferongames.com/post/physics_in_3d/) we discussed how to simulate the motion of rigid bodies in 3D. Now we're going to discuss how to implement spring physics.

The physics behind springs is simple but extremely versatile and useful. You can use springs to link points together to model rope and string, cloth, and even blobs of jelly. Springs can also be used to implement basic collision response, and to create joints that constrain the motion of rigid bodies.

The more physics programming you do, the more springs pop up. Many physical phenomenon boil down to spring-like forces being applied such as buoyancy in water. Springs are everywhere so lets discover how to simulate them!

Spring and Dampers

The formula to use for simulating spring-like behavior is called Hooke's Law (http://web.archive.org/web/20181107181500/https://en.wikipedia.org/wiki/Hooke's_law).

```
F = -kx
```

Where **x** is the vector displacement of the end of the spring from it's equilibrium position, and k is a constant describing the tightness of the spring. Larger values of k mean that the spring is tighter and will therefore stretch less per unit of force, smaller values mean the spring is looser and will stretch further.

Newton's third law says that every force has an equal and opposite force. If two objects a and b are connected together by a spring then the spring will apply one force which pulls object a towards object b, and an equal and opposite force pulling object b towards a. However, if you want to attach one object to a

Spring forces alone are not much use though. You need to combine them with dampers to have a realistic simulation. Damping simulates energy loss, and it is used in physics simulations to make sure that springs don't oscillate forever but come to rest over time.

A spring-damper system can be modeled as follows:

```
F = - kx - bv
```

Where b is the coefficient of damping and **v** is the relative velocity between the two points connected by the spring. Larger values for b increase the amount of damping so the object comes to rest more quickly.

Variations on Springs

There are many different variations on spring-damper systems, but what I want to do is explain how to think generally in terms of what the forces are doing so that you can easily design spring and damper systems to achieve whatever effect you want.

The first thing is that springs don't only have to act to pull two points together so that they lie on top of each other. For example you can design spring forces that pull together or push apart two points apart so they maintain a desired separation distance from each other:

```
F = -k(|x|-d)(x/|x|) - bv
```

Where **|x|** is the distance between the two points connected to the spring, d is the desired distance of separation, and **x** / **|x|** is the unit length direction vector between the two points: **a** to **b**, when applying the force to point **a** and vice versa.

The overall effect of the force equation above is to have a force which pushes the two points connected by the spring apart if they are closer than distance d, and bring the two points together if they are further than d apart. Notice how the force becomes exactly zero when the two points are at the target distance? If you tune the k and b parameters correctly you can have a nicely behaving spring that quickly brings the two points together smoothly over time and comes to rest at the solution point.

But why apply springs to position only? If you want to accelerate a body over time such that it accelerates to a certain speed then you can calculate a spring force proportional to the difference between the current velocity and the target velocity, combined with a damping proportional to the current velocity so that it reaches its target over time instead of cycling about it. This is usually called a motor in physics simulation.

We can even apply the same concept to drive the spinning of an object at a certain speed by applying a spring torque proportional to the difference between the current angular velocity and the desired angular velocity, coupled with a damper force proportional to the current angular velocity.

Another commonly implemented spring constraint is to enforce a upright orientation of a body, for example, you could apply a spring torque proportional to the difference between the current orientation and an upright orientation, coupled with a damper proportional to angular velocity. Such a constraint is called a 'stay upright constraint' and its often used for sci-fi hover racing games.

reached. In order to make sure that your physics objects don't just oscillate around the solution but actually reach it, it is necessary to apply damping proportional to whatever physics state values are performing the evolution of the simulation towards the solution over time.

Attachment using a Spring

So lets get started with an actual concrete implementation of using springs a simulation. The first thing we will implement is an attachment joint that will allow the user to click and drag a point on the cube to move it around. We will implement this by modeling a tight spring attachment between a target point and an attachment point on the body of the cube. This is effectively a ball and socket joint implemented using only spring forces and is implemented using the standard equation we are used to:

```
F = -kx -bv
```

Where x is the vector difference between the current target point and the attachment point on the object, and v is the point velocity at the attachment point on the object. The important thing is that this velocity v being the point velocity means that it incorporates both the linear motion of the object plus any velocity at the attachment point due to angular velocity. As was shown in the previous article we can calculate this point velocity at follows:

```
v_point = v_linear + v_angular cross (p - x)
```

Where **p** is the point on the rigid body and **x** is the center of mass of the object. Secondly, this spring and damper force is not just applied linearly, but is applied at the attachment point on the object. This means that the spring force will apply both a linear force and a torque component as follows:

```
F_linear = F
F_torque = F cross (p - x)
```

The overall effect of this joint then is to bring the target and attachment points together while damping the motion of the object at the attachment point. This allows the object to move as long as remains still at the attachment. In other words the object is only allowed to move by rotating about the attachment point. Our simple ball and socket joint is now complete.

Collision Response with Springs

Next we will implement a basic collision response using springs. The trick here is to apply a spring and damper force that works against what we don't want, eg. objects moving towards each other and penetrating other objects.

So we have a collision detected and the usual information is returned to the physics system ready to apply collision response. This information is typically something like:

- A unit length collision normal
- The penetration depth along the normal
- The physics state for each of the colliding objects at the time of collision

of collision detection.

Once we have all our contact information a simple collision response can be implemented by applying a spring-like force to the colliding objects to keep them apart:

```
F = nkd - bn(n.v)
```

Where k and b are the spring-damper coefficients, **n** is the contact normal and **v** is the relative velocity between the two objects at the point of collision. Effectively this equation calculates a spring force that pushes out along the contact normal while reducing the relative velocity of the objects towards each other at the contact point.

Various different collision responses types can also be achieved using this equation, for example setting b to 0 gives a completely elastic collision response where all energy going into the collision returns in the bounce over time. Setting b to higher value tend to make the collision more inelastic because it removes more energy during the collision. Finally, by increasing and decreasing the spring constant k in concert with b you can make a collision that feels like anything from bouncing off a trampoline (low k and low b), to landing and sinking into quicksand (low k and high b), or landing with a splat on concrete (high k, high b).

The Weakness of Springs

It seems we can achieve a large variety of different collision effects using only springs and easily make joints and constraints. Its not all good news however because springs come with their own set of weaknesses which I will now explain.

The first weakness is that its difficult to tune the spring constants to get exactly the effect you want. For example, if you are attempting to model the real world physics of an object, you will need to experiment to find the spring k and b values that match the simulation. These values are usually dependent on other values of your simulation, such as the gravity constant, meaning that if you adjust gravity you'll need to retune your springs to get the same effect.

The next problem is that tighter the spring k you use, the more difficult it becomes to solve the differential equation. Using an RK4 integrator sure help with this, but even with RK4 there is a fundamental limit to how large you can make your spring k before your simulation will explode. At this point you need to either decrease your timestep or reduce your spring k.

The final, and major weakness is that springs are reactive not predictive. This is a subtle point but a very important one. A joint or constraint implemented using springs only works by correcting errors after they occur, and collision response using springs requires allowing some amount of penetration before it acts to correct it and so forth. More advanced techniques exist which can solve for the forces required to constrain the physics simulation without inducing error, such as LCP solvers or iterative methods, but they are out of scope of this simple article.

NEXT ARTICLE: Networked Physics (2004) (http://web.archive.org/web/20181107181500/https://gafferongames.com/post/networked_physics_2004/)

(http://web.archive.org/web/20181107181500/https://twitter.com/gafferongames)

(http://web.archive.org/web/20181107181500/https://github.com/gafferongames)