Penalty Methods

Andrew Witkin
School of Computer Science
Carnegie Mellon University

1 What's a constraint?

In optimization, constraints are typically part of a problem specification, e.g. "I want the nicest possible shrubbery, subject to the constraint that it not be too expensive." In physical simulation, a constraint is typically part of the description of a physical system, e.g. "the leg bone's connected to the hip bone." The main point of constraints is to provide simple, idealized models of possibly complicated interactions among objects. Constraints let us specify interactions in terms of their *effects* on behavior, without worrying about the grisly details of the interaction forces that produce the effect. For instance "point $\mathbf{p}_1$ on object **A** is attached to point $\mathbf{p}_2$ on object **B**", or just $\mathbf{p}_1 - \mathbf{p}_2 = 0$, is an idealized summary of what happens when you pin two objects together with a universal joint—whatever you do, the two points stay together. To actually use a constraint as part of a simulation, we need to somehow convert this kind of statement into *forces* that make it happen.

We'll start with the *penalty method.* The penalty method is a Bad Method, at least if you want objects to really behave the way the constraints said they should, but it's very simple, and lays the groundwork for the Good Methods we'll study afterwards. The basic idea of the penalty method is to convert a behavioral description into a position- and velocity-dependent force law that tries to keep the constraints satisfied—essentially, a generalization of damped springs. The problem with this approach is that the generalized springs need to be *very* stiff to really hold things together, which leads to horrible numerical difficulties.

The first thing we need is a way to specify the constraints. To do this, we use a "behavior" or constraint function—a function of all the particles' positions that's always supposed to be zero (it can depend on one, two, or many particles, and it can return a vector or a scalar.) I.e. a constraint function has the form

$$C(\mathbf{x}_1, \cdots, \mathbf{x}_n) = 0.$$

For instance, a vector constraint that says two particles should coincide is $\mathbf{x}_1 - \mathbf{x}_2 = 0$, and a scalar constraint that says they should be distance r apart is $|\mathbf{x}_1 - \mathbf{x}_2| - r = 0$.

2 Penalty constraints

The basic idea of the penalty method is to treat the square (or squared magnitude) of the constraint function as a *potential energy*, giving rise to a force on each participating particle which is minus the derivative of energy with respect to that particle, making all the particles roll "down hill" on the energy landscape. An additional twist is that we need to add damping to inhibit oscillations. Since the penalty forces are just regular forces, they compete with every other applied force—gravity, user-applied forces, etc. In order to make the constraints win, down to a respectable tolerance, we

have to multiply the energy by a big enough number to swamp everything else out, which is what makes penalty methods lose.

Given the constraint function $C(\mathbf{x}_1, \cdots, \mathbf{x}_n)$, we define the *energy* or penalty function as

$$E = \frac{k_s}{2}|C|^2,$$

where k_s is the weighting or stiffness coefficient, analogous to a spring constant. The division by 2 serves to cancel the factor of 2 that arises when we differentiate. Neglecting damping, the penalty force on particle $\mathbf{x}_i$ is

$$\mathbf{f}_i = -\frac{\partial E}{\partial \mathbf{x}_i} = -k_s C \frac{\partial C}{\partial \mathbf{x}_i}.$$

(Note: If C is scalar, $\partial C/\partial \mathbf{x}_i$ is a vector. If C is a vector, $\partial C/\partial \mathbf{x}_i$ is a matrix.)

The damping term is a generalization of simple viscous drag ($\mathbf{f}_d = -k_d\dot{\mathbf{x}}$) except that the damping force is proportional to the time derivative of C, which is given by

$$\dot{C} = \sum_i^n \frac{\partial C}{\partial \mathbf{x}_i}\dot{\mathbf{x}}_i.$$

We apply the damping force in the same direction that we apply the main part of the penalty force, leading to the full penalty-force expression

$$\mathbf{f}_i = -(k_s C + k_d \dot{C})\frac{\partial C}{\partial \mathbf{x}_i},$$

where k_d is a coefficient of damping.

Here's a stupid example: To try to keep a single particle $\mathbf{x}$ at the origin, we can let $C(\mathbf{x}) = \mathbf{x} = 0$. Then $E = -k_s\mathbf{x}\cdot\mathbf{x}$, $\partial C/\partial \mathbf{x}$ is the identity matrix, I, and $\dot{C} = \dot{\mathbf{x}}$. So $\mathbf{f} = -(k_s C - k_d\dot{C})\partial C/\partial \mathbf{x} = -k_s\mathbf{x} - k_d\dot{\mathbf{x}}$, which is what you'd expect.

And a marginally less stupid example: to hold two particles together, we have $C = \mathbf{x}_1 - \mathbf{x}_2$, so $\partial C/\partial x_1 = I$, $\partial C/\partial x_2 = -I$, and $\dot{C} = \dot{\mathbf{x}}_1 - \dot{\mathbf{x}}_2$. So $\mathbf{f}_1 = -k_s(\mathbf{x}_1 - \mathbf{x}_2) - k_d(\dot{\mathbf{x}}_1 - \dot{\mathbf{x}}_2)$.

Exercise: Derive the penalty force for the fixed-distance constraint $C(\mathbf{x}_1, \mathbf{x}_2) = |\mathbf{x}_2 - \mathbf{x}_1| - r$, where constant r is the desired distance. You should come up with the standard formula for a damped spring with rest length. If not, one of us screwed up.

3 Why penalty methods lose.

Here's a really simple analysis of the numerical problem. Assume we're in 1-D, and we want to keep a particle at $x = 0$. We'll let the particle's motion be governed by the first-order equation $\dot{x} = f - kx$, where f is an applied force, and the term $-kx$ is the penalty term with coefficient k. Now suppose that the biggest force we'll ever apply is f_{max}, and the biggest constraint error we'll tolerate is ϵ. Then, at equilibrium (i.e. $\dot{x} = 0$), the force f_{max} should induce a particle displacement ϵ. For this to happen, the penalty force has to cancel the applied force f_{max} so that $f_{max} = k\epsilon$, or $k = f_{max}/\epsilon$. If dividing by ϵ makes you nervous, you're on the right track.

Now, to keep things really simple, assume we're taking time steps using Euler's method so that $\Delta x = h\dot{x} = h(f - kx)$. If we apply f_{max}, let the system equilibrate at $x = \epsilon$, then suddenly let go ($f = 0$), what happens? Well, if $hk > 1$ the system will bounce past zero on the next time step, and

there will be oscillation. If $hk > 2$, the system will bounce past $-\epsilon$, and the oscillation will *diverge!* Since $k = f_{max}/\epsilon$, this means that to avoid oscillation the step size must not exceed $h = \epsilon/f_{max}$. If we want accurate constraints, ϵ has to be a displacement that's too small to notice. This means that if we apply the biggest force in the world f_{max}, the distance the particle will move in one time step is $\Delta x = hf_{max} = \epsilon$, which is a distance that's too small to notice. In other words, we've made the step size so small that we never get anywhere. And if we try to make the step size bigger, we explode.