

Dan Morris
Notes on “Large Steps in Cloth Simulation”
(a.k.a “a tutorial on implicit integration”)

Overview

The goal is to simulate realistic behavior of cloth... constraints are :

- Reasonable computation time
- Forces applied externally
- Rigid object contacts handled properly

Explicit Methods: Forward Euler

So first let’s take the simplest possible approach, and see what’s wrong with it... assume I not only don’t know about differential equations, I don’t even know about matrices.

So I model my cloth as a surface in 3DS max or whatever, I decimate it down to the lowest resolution I need, and I treat each of the vertices as a particle with arbitrary mass, and each of the edges as a linear spring. Then I pick a nice timestep, probably the frame rate I need for rendering, and I set each particle’s initial velocity to zero. So I do :

```
for each particle p :  
    compute the force exerted by each edge connected to p according to  $f = kx$   
    add in any other forces I want to exert on this particle  
    compute  $a = f / m$   
    compute  $dv = ha$   
    compute  $v = v + dv$   
    compute  $dx = hv$   
    compute  $x = x + dx$   
end
```

Easy enough. This is the forward Euler explicit simulation method. Also note that this is an iterative formulation of exactly the matrix equation that’s written in equations (2), (3), and (3.5) :

$$\begin{pmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{M}^{-1} \mathbf{f}_0 \end{pmatrix}$$

What’s wrong with this?

- Instability... one of the most important buzzwords in physical simulation. What does this really mean?
 - This approach does nothing to prevent from – in the authors’ words – “wildly changing derivatives”
 - If particles get a little too far apart, the force between them will be transiently high, so they’ll have a transiently high acceleration. The resulting high velocity will last a *whole timestep*, instead of the very small amount of time it really should have lasted.

- Important to see the *real* problem here: if the derivative was either computed more accurately or applied for a short amount of time, the problem would go away.
- Doesn't allow realistic modeling of material properties... the only thing you can control is spring constants, which have no real physical interpretation. Leads to basically a weak simulation of a piece of rubber.
- Doesn't address how to handle self-collisions or collisions with other objects in the world. Those things aren't fundamentally incompatible with this method, but that's still something we need to "fix" in our simulation...

Implicit Methods: Backward Euler

So let's address the first problem: instability. Remember that the problem here is that the force we compute at the beginning of a time step is applied throughout the step. So instead why not ask what the force will be at the *end* of the time step, and compute that? This is the backward Euler method, presented in Equation (4).

$$\begin{pmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v}_0 + \Delta \mathbf{v} \\ \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}_0 + \Delta \mathbf{x}, \mathbf{v}_0 + \Delta \mathbf{v}) \end{pmatrix}. \quad (4)$$

But this is like a recursive definition... you're asking: "What change in position can we apply that will give us a force that leads to exactly that change in position?" It turns out this is really hard to solve for, so we just *estimate* what the force will be at the end of the timestep for a given Δx and Δv . How do we estimate this? We have to be able to compute the partial derivative of force with respect to changing positions and velocities... (note this may eventually constrain our force representation). This estimation is represented in equation (4.5), which (we can skip the algebra) gives us the most important equation in the paper, equation (6) :

$$\left(\mathbf{I} - h\mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2 \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right) \Delta \mathbf{v} = h\mathbf{M}^{-1} \left(\mathbf{f}_0 + h \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{v}_0 \right) \quad (6)$$

We have a linear system here in the usual form. The only unknown is Δv (a vector with three elements per particle in our cloth) (we can trivially compute Δx from Δv). The rest of the paper looks at what all the knowns are, and how we solve this equation. For now, let's put the solution method aside and assume we solve by inverting the whole matrix on the left side of the equation.

Realize that at this point we *could* apply our simple linear-spring force model at every element, and estimate all the relevant derivatives, and we would still have made some progress by using an implicit method. But we would also still have a simple model that didn't behave like cloth, so we'll start looking at the second of the two problems we posed above...

But first let's ask why *everyone* doesn't do this... if it provides stability, why would anyone ever use forward Euler (aside from simplicity of code)? Note that we don't have to *solve* anything for forward Euler... we just computed forces and ran a single multiplication. But now we have to *solve* a linear system, which is typically slow. That's why most folks – particularly folks doing interactive simulation – don't use this approach.

Back to figuring out what our Δv 's are... basically we need to figure out what all the f 's and ∂f 's in the above equations are, based on the current state of the system. This was easy in our $f = kx$ model, but what kinds of forces can we propose – given all the current positions and velocities of our particles – that give more realistic behavior?

Force Models for Cloth Simulation

To make things a little more complicated, they introduce one level of indirection before we actually get forces... instead of just postulating a force acting on each particle, we introduce a series of “constraints” on each particle, which is basically a potential function that the universe wants to minimize. Height in a gravity field would be a good example of such a potential function... we define each internal force in the cloth as the force that nature exerts to minimize these potential functions, which turns out to be $-k(\partial C(x)/\partial x_i)C(x)$ (eq. (7)).

There are two things that I don't fully understand here, so I'll be up front about those :

- These $C(x)$ constraint functions turn out to be pleasantly intuitive, but I'm still not sure *why* they use them. They say that “cloth's material behavior is customarily described in terms of a scalar potential energy function” but “sensible damping functions cannot be derived from energy functions”. I don't really understand (a) why I would use energy functions instead of directly postulating $f(x)$ functions (like we did in the mass-spring example) or (b) what the deal is with the damping, and why one formulation makes it easier.
- We need to compute lots of $\partial C(x)/\partial x$'s, but I don't fully understand how. I assume it's some kind of finite difference thing, but it seems important, and they never mention *how* they actually compute these derivatives.

Back to cloth simulation... what potential functions do we invent to give us forces that model cloth?

- Stretch: the cloth “wants” the area of each triangle to remain constant, and usually wants it to be equal to whatever it is in the mesh's rest (planar) configuration. This is equation (10):

$$\mathbf{C}(\mathbf{x}) = a \begin{pmatrix} \|\mathbf{w}_u(\mathbf{x})\| - b_u \\ \|\mathbf{w}_v(\mathbf{x})\| - b_v \end{pmatrix} \quad (10)$$

Note the constraint function formulation... the “universe” wants to make this function zero, so a force is defined as the force that will minimize this function, or.

- Shear: cloth resists in-plane shearing (although it typically “prefers” this to stretch). How do we formulate this as a constraint? Basically we try to minimize the angle that each triangle’s shape is changed, so we write (really, this turns out to minimize shape change, but you have to stare at it a bit):

$$C(\mathbf{x}) = a\mathbf{w}_u(\mathbf{x})^T \mathbf{w}_v(\mathbf{x})$$

Here u and v are basically the two-d texture coordinates of the cloth, i.e. the coordinates of the un-deformed configuration. This brings up an important point: every piece of cloth is modeled as a plane that is ultimately deformed against its will into some other configuration (e.g. a sleeve). Objects like shirts consist of multiple planes stitched together, and stitching amounts to creating vertices that are shared among multiple planes and therefore respond to forces from multiple planes.

- Bend: cloth resists bending out of the plane (although it also “prefers” this to stretch). As a constraint, we want the angle θ between any two adjacent triangles to be minimized. So we write :

$$C(\mathbf{x}) = \theta$$

I assume that this constraint is modified at seams, where θ should be equal to something other than zero. They don’t really talk about this. Maybe no constraint at all is placed on bending around seams.

- Damping: every simulation in the world has some damping to prevent instability. They’re really excited about their damping model, but it’s basically a force that limits acceleration due to each of the other forces, and I’m not going to discuss it in more detail.

Constraints

We could take the forces we proposed above and run our implicit integrator on them, and get a good simulation of cloth floating in space. But really another key constraint we want to impose is that the cloth can’t penetrate other solid objects (like the guy wearing the shirt), and can’t penetrate itself when it folds over.

They handle self-collisions in a very simple manner; they add another $f(x)$ function for each penetrating vertex that represents a linear spring intended to push intersecting pieces of cloth apart. This is simple and non-physically-based, but I think it works because the examples they use simply don’t have that many cloth-cloth collisions.

Now what about cloth/solid collisions... we can't take the same simple approach, since the cloth is *constantly* in contact with rigid materials, and the lack of realistic behavior would be a real problem. Instead, we say that wherever a piece of cloth is effectively attached to a solid object, we want to (a) make sure it doesn't get pushed inside the object and (b) constrain its velocity to be the same as that of the object (e.g. my sleeve should move at the same velocity as my arm wherever they're in contact).

They achieve (a) in a rather obvious way... if a particle is inside an object after some timestep, they just push it outside. Doing this in the simplest possible way actually creates instability... remember there are really strong springs that don't like triangles to get stretched, and moving particles around without asking the integrator for permission makes these springs unhappy. So they actually incorporate these simple, explicit position changes into their equation, by just adding an arbitrary vector \mathbf{y} to the $\Delta\mathbf{x}$ vector that goes into their equation, giving a modified version of our master equation... that is, equation (6) becomes equation (18) :

$$\left(\mathbf{M} - h \frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right) \Delta \mathbf{v} = h \left(\mathbf{f}_0 + h \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{v}_0 + \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{y} \right). \quad (18)$$

To achieve (b) (velocity constraints), they postulate (but don't end up using) a clever method... the \mathbf{M}^{-1} term in eq (6) seems so simple that we ignore it. It's a diagonal matrix where each element on the diagonal is the mass of the corresponding particle. But what is the physical interpretation of futzing with this matrix? Effectively, a particle with an m^{-1} value of zero has an infinite mass and can't be accelerated... and a particle with an m^{-1} value just for its x-component can't be accelerated along the x axis. In other words, by modifying the "mass" of the object, we can create arbitrary zero-acceleration constraints. That means the solution we get from the solver won't accelerate those particles at all (Δv_i will be zero)... what if we want to make a particle's velocity be exactly equal to that of some solid object? We set its inverse mass to zero, and make up an arbitrary \mathbf{z} representing the velocity that this particle *must* have. This is equation (14):

$$\left(\mathbf{I} - h \mathbf{W} \frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2 \mathbf{W} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right) \Delta \mathbf{v} = h \mathbf{W} \left(\mathbf{f}_0 + h \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{v}_0 \right) + \mathbf{z} \quad (14)$$

This is really just a clever example of exploiting our ability to futz with what seemed to be a straightforward term in the main equation (\mathbf{M}^{-1}). In fact they don't end up using this, because the resulting equation (14) turns out to be nonlinear and hella slow to solve.

What they *actually* do is to integrate their velocity constraints directly into their equation solving process... remember that at the end of the day, we have to actually *solve* equation (6) or (18), and they decide to use an iterative conjugate gradient method to do this. An iterative method has the nice property that you can step into every iteration and make changes to the "current" result, to enforce constraints.

Note the important decision they made here... before they made this decision (to use the CG solver to enforce constraints), everything in this paper was totally independent of the solver you used. So if I found a good solver at goodsolvers.com, I could use their simulation method, their forces, etc. But this decision ties them tightly into the particular solver, and requires them to implement their own solver.

Adaptive time stepping

The big advantage of implicit solvers is that you can take big time steps without instability. In the end, some people will argue that solving one giant linear equation for each frame is actually *more* expensive than taking 10 forward Euler steps, but of course certain applications will like one approach or the other.

In either case, the bigger your timestep, the more likely your system is to be unstable and/or inaccurate. So a good thing to do is to take whatever time step you want until the s&!* hits the fan, then dial back the timestep to get back your accuracy. This is exactly what Baraff and Witkin do... for cloth, they find that any significant in-plane stretch really means that s&!* has gone ill, not that the cloth is just stretching, since cloth doesn't like to stretch. So they any time they observe a big stretch, they go back in time to the previous timestep and start simulating again with a bigger timestep. Then they wait a while and see if they can dial the timestep back up... if s&!* goes ill again, they dial it back down, etc.

Results

Their results look quite good; the cloth looks like cloth, and they're clearly able to model wrinkles and cuffs and other nice things. They also state that their method compares favorably to explicit methods. However, it's hard to tell (a) how geometrically complex their models are, (b) how often their simulations went unstable (i.e. how many "takes" they needed to get it to look good, and (c) how realistic it really looks when it gets animated. Also, their cloth model is still somewhat simple... other papers would come out later that simulated additional properties of cloth, like creases, buckling, real cloth-cloth collisions, and tearing.