CSE291 HW 3: Elasticity

Ritoban Roy-Chowdhury

May 30, 2023

Governing Equations

My code implements an elastic body simulation, using a finite element method. An elastic body is defined by a deformation map ϕ from some material coordinate $M \subset \mathbb{R}^n$ to some world coordinate $\mathbb{R}^n$. The differential of this map $F = d\phi$ is the deformation gradient, and then, we postulate that there exists a stress-strain relation defining the second Piola-Kirchoff stress $S = 2\mu E + \lambda \operatorname{tr}(E) I$, where the strain tensor E is computed from F as $E = \frac{1}{2}(F^T F - I)$. Then, the equation of motion for the elastic body is given by $\rho_M \ddot{\phi} = \nabla \cdot P$, where $P = FS$ is the first Piola-Kirchoff stress.

Discretization

We discretize this equation of motion using a finite element discretization (which really should be called finite volume discretization, but I'll come back to that). We first spatially discretize the domain into several triangle/tetrahedra. Then, for each triangle/tetrahedron, we can compute vectors for it's edges in both material and world space, and place them as the columns of a matrix. If D_m represents this matrix in material space and D_w represents this matrix in world space, then by defintion $FD_m = D_w$, so we can compute F as $D_m^{-1} D_w$. Then, we can compute the stress at each element using the equation above. Finally, we can discretize the divergence operator by performing the sum at a vertex V as $\mathbf{f_V} = \sum_{E \text{ on opposite faces for tetrahedra adjacent to } V} P_V \mathbf{n}_E A_E$, where $\mathbf{n}_E A_E$ is the area-weighted normal on the face opposite to the vertex.

For time integration, I used backwards Euler, using an incremental potential update. In particular, I construct a minimization problem, which I solve with gradient descent (I actually planned to use Newton's method, and the code implements Newton's method, but I didn't have time to implement the Hessian correctly, so I just use the identity matrix for the Hessian, so Newton's method just becomes gradient descent). In particular, at each timestep, we perform the minimization

$$\operatorname{argmin}_{\text{vertex positions } q} \frac{M}{\Delta t^2} |q - (q^{(n)} + \Delta t v^{(n)})|^2 + U(q).$$

We perform this minimization by taking the derivative of this experession, and simply taking steps in the direction of the gradient, with a very small step size α (I use $\alpha = 10^{-6}$). This is actually surprisingly stable (much better than symplectic Euler or RK2, both of which I experimented with).

This minimization gives us a new set of positions $q^{(n+1)}$. From this, we compute the velocities from the previous timestep as $\frac{1}{\Delta t}[q(n+1) - q(n)]$, which we need to compute $q_{pred} = q^{(n)} + \Delta t v^{(n)}$ for the next time step. We set these new positions, and then if we've performed enough substeps for a frame, we render the frame.

Example

My example is a very simple demonstration of elasticity. The elastic body is a 2d box, discretized as a 5×5 triangular grid mesh. It is relased from a height of 10 units above the ground. The ground is modelled using a potential barrier function

$$\phi(x, a) = \begin{cases} -(x-a)^2 \ln\left(\frac{x}{a}\right) & 0 \leq x \leq a \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

following Li et al. [2020]. Then, our potential will be $U_{\text{plane}}(\mathbf{v}) = \phi(v_y - h, a)$, where h is the height of the plane, and a is the width of the potential barrier (I use $a = 0.1$ for the examples below), similar to my previous rigid body assignment. Altogether, this gives an elastic box that falls on the ground and bounces around for a bit.

I also include a constant gravity force of 1 unit per second squared in the $-y$ direction.

Interestingly, in both this assignment and the previous one, I've noticed this potential function has a tendency to give a visible "kick" to the object, especially when using explicit integration. I'm not sure why this is, but I suspect part of it might be that if the object penetrates below the ground, then the distance gets clamped to a small ϵ, which impacts a constant, non-conservative force on the object (though this isn't entirely consistent with what I've seen when I add a debug statement to the clamping case, so I don't think this is the whole story).

Implementation Notes

The code implements Newton's method in a decent amount of generality using generics (there are several simple test cases implemented, including a multivariate quadratic form optimization example, which Newton's method is able to generally do in just one step). Then, the actual simulation just becomes a matter of specifying the derivative of the potential, which really just invovles calculating forces as described above. The derivative of the kinetic energy term works out to be $\frac{m(q-q_{pred})}{\Delta t}$, while the derivative of the potential energy term involves computing the discrete divergence as described above for the hypothetical set of vertex positions where Newton's method computes the derivative at that step.

I decided to experiment with yet another rendering library for this assignment (I've used a different one for each project so far in this course :D) – this time, unfrotunately, the library's 3d rendering capabilities were very slow and inconvenient, so I ended up just doing a 2d simulation.

Future Work

Unfortunately, I didn't have time to implement every I wanted to in this project – in particular, I didn't get time to implement the Hessians for Newton's method, so effectively, I'm just doing gradient descent. I'd also like to implement a backtracking-based line search algorithm, instead of just manually tuning a step size (I tried implementing this, but again, I ran out of time). Finally, I believe my code should work in both 2d and 3d, but unfortunately I didn't get any time to experiment with the 3d case, partially because I chose to experiment with yet another rendering library for this project, and that library turned out to not have great 3d support (it also didn't have good support for exporting videos, hence the scuffed screen recording).

A comment on terminology

We've been calling this method "finite element", but as I discussed with Prof. Chern in office hours, it really is quite different from the standard test function centric approach to finite element (which involves constructing a large linear system for the stiffness of your PDE – our description of finite element does not require a global solve at all). I did some research and found that Teran et al. [2003] is actually the original paper that introduces this approach to graphics, and they use the term *finite volume*, rather than finite element, and this usage is consistent with the normal usage of the term finite volume in the computational physics community (in particular, the divergence integral over the domain is discretized as an integral over a boundary using Stokes theorem).

They also claim that finite volume method is equivalent to finite element, using tetrahedral elements with linear basis functions, with constant strain over each tetrahedra – they claim this follows from a "straightforward computation" – it's not entirely clear to me how it's straightforward, but I guess that does explain where the terminology comes from. It would probably be more clear and consistent if graphics people went back to calling it "finite volume", but it is what it is.

References

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4):49, 2020.

Joseph Teran, Sylvia Blemker, V Ng Thow Hing, and Ronald Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 68–74. Citeseer, 2003.