CSE 291 (SP 2024) Homework 3

3.1 Miniproject

Exercise 3.1 — 20 pts (+5 bonus). Design a 2D or 3D elastic body simulation. Write a document to describe your system and numerical algorithm. Some examples include

- Soft body falls and bounces off the ground.
- Soft body with points constraints (fix a few vertex position, or let these vertices be animated) and let the rest of the soft body freely bounce. This can be an animation of picking up and shaking an elastic body.
- Seismic wave simulation (observe different modes of wave propagation like S-wave P-wave.)

You can use symplectic Euler, *e.g.* explicit RK4 or implicit Euler (incremental potential).

Bonus credits will be given to those with excellent exposition or complexity of the system. ∎

Hint *If you use incremental potential and wants to solve the linear system with Hessian, note that the Hessian should be a (large) sparse matrix. In theory, the potential energy should be designed so that the Hessian is (symmetric) positive definite. For these large sparse linear system, build matrices as sparse matrices data type and use iterative solver such as the conjugate gradient solver (*e.g. `scipy.sparse.linalg.cg` *in SciPy).*

Hint *Explicit methods (symplectic Euler, explicit RK4) don't require building large linear system.*

Hint *If you provide point constraints, then the remaining variables are the point positions that are not constrained. You can label vertices "free" (*f*) and "constrained" (*c*). The strain is computed using all vertex position, but the final force only needs to be evaluated on free vertices. If you use incremental potential, then the linear solve* $\mathbf{H}\delta = \mathbf{b}$ *for the optimization step* δ *with (approximated) Hessian* $\mathbf{H}$ *and differential* $\mathbf{f}$ *of loss function can also be reduced to a system only on free vertices. Slice the linear system into free and constrained part* $\begin{bmatrix} \mathbf{H}_{\mathrm{ff}} & \mathbf{H}_{\mathrm{fc}} \\ \mathbf{H}_{\mathrm{cf}} & \mathbf{H}_{\mathrm{cc}} \end{bmatrix} \begin{bmatrix} \delta_{\mathrm{f}} \\ \delta_{\mathrm{c}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{\mathrm{f}} \\ \mathbf{b}_{\mathrm{c}} \end{bmatrix}$*, and use the fact that* δ_{c} *is known (probably zero when it's in the optimization stage) to obtain* $\mathbf{H}_{\mathrm{ff}}\delta_{\mathrm{f}} = \mathbf{b}_{\mathrm{f}} - \mathbf{H}_{\mathrm{fc}}\delta_{\mathrm{c}}$.