[CSCI 2240] Assignment 3: Finite Element Simulation (fem)

Released: 3/6/20
Due: 3/30/20 @ 11:59pm

In this assignment, you'll animate deformable solid objects using the Finite Element Method (FEM). The name "Finite Element Method" comes from the fact that this approach divides a continuous chunk of material into a mesh made up of a finite number of discrete elements (in this case, you'll use tetrahedra). FEM allows for the simulation of physically-based materials in a principled way (as opposed to ad-hoc methods such as spring-and-mass simulations). You will implement the core features needed for a basic deformable object simulation (e.g. force computation, time integration, simple collision resolution) plus one or more extra features. To show off what your code can do, you'll submit one or more videos demonstrating your simulator in action.

Relevant reading

- The lecture slides!
- Baraff and Witkin's course notes on physically-based modeling are a good reference for the basics of dynamics and time integration.
- O'Brien and Hodgins provide a good introduction to FEM fundamentals for graphics (Sections 1 - 3).
- Adam Bargteil's Finite Element Notes are a nice supplement.
- This handout on computing strain and stress may also be informative.

Requirements

This assignment is out of **100 points**

Your simulator must implement at least the following features:

- Extract the surface mesh from your tetrahedral mesh **(10 points)**
  - You will need this to render your simulation (see below).
  - How might you do this? (i.e. how can you tell whether the face of a tetrahedron is a boundary face or an internal face?)
- Compute and apply force due to gravity **(5 points)**
- Compute and apply internal elastic forces **(30 points)**
  - Compute Green's strain for each element
  - Compute the stress for each element
  - Compute per-node forces
  - You can assume a lumped-mass model for your mesh (i.e. constant density within an element, mass of element distributed evenly to its four vertices).

- Compute and apply internal viscous damping forces **(10 points)**
- Resolve collisions **(10 points)**
  - You must implement collision between the mesh and a ground plane, as well as at least one other type of obstacle (e.g. spheres).
  - The simple ‘penalty force’ method described in Section 3.3 of O’Brien and Hodgins is sufficient.
- Integrate your simulation forward in time using the explicit midpoint method **(10 points)** (regular Euler integration recommended to start with)

You must also submit at least one video demonstrating your simulator in action **(10 points)**.
The video(s) must demonstrate all of the features you have implemented (including any extra features).
There are a few different ways you might go about making such videos:

- Screen capture an OpenGL rendering of your simulation, e.g. using the interactive viewer code that we provide below (see “Resources”).
- Export frame-by-frame meshes from your simulator and use your path tracer from Assignment 1 to render them.
- Use some other modeling/animation/rendering software to render exported meshes (e.g. Maya, Blender).

Particularly creative and/or nicely-rendered animations may receive extra credit. Think about interesting scenarios you could set up (i.e. ways to apply external forces to the mesh).
Please use a standard format and codec for your video files (e.g. .mp4 with the H264 codec).
To turn a set of frame images into a video, you can use FFMPEG:
https://hamelot.io/visualization/using-ffmpeg-to-convert-a-set-of-images-into-a-video/

You must also submit a plaintext README file **(5 points)**
This file should describe how to run your simulator (e.g. what command line arguments are needed?)
This file should also list all of the features your simulator implements.
Finally, it should describe what features are demonstrated by the video(s) you’ve submitted.

Successfully implementing all of the requirements results in a total of **90/100 points**.
To score **100/100** (or more!), you’ll need to implement some extra features.

Extra Features

Each of the following features that you implement will earn you extra points. The features are ordered roughly by difficulty of implementation.

- Share a cool tet mesh on Piazza **(2 points)**
- Make the visualizer pretty **(5 points)**
  - Miss programming shaders? Modify shader.frag to add some fancy effects. Skyboxes, shadows, FBO hacks, and more are all welcome.
- A higher-order explicit integrator **(5 points)**

    - This will allow you to take larger simulation timesteps.
    - Runge-Kutte 4
    - Verlet integration
- Adaptive time stepping **(5 points)**
    - Take the largest time step you can take while remaining within some error threshold.
    - Baraff and Witkin's notes are helpful here.
- Parallelize your code **(5 points)**
    - Many simulator operations are 'embarrassingly parallel' (force computations, integrator steps, etc.)
    - Even something as simple as OpenMP's parallel for loop can buy you significant speedups, if applied in the right places.
- Interactivity **(10 points)**
    - Modify the scene viewer (see "Resources" below) to allow the user to poke, push, drag, etc. a deformable mesh.
- Self collisions **(15 points)**
    - Or, collisions between two deformable meshes.
    - The O'Brien and Hodgins paper has some suggestions for how to do this.

Advanced Extra Features

These extra features are significantly more challenging to implement, and they involve reading other papers to implement. Some of these are probably big enough in scope to be closer to final project ideas, to be honest. I've listed them here for completeness and to potentially get some people interested in some of these ideas :)

- Corotated linear elasticity **(20 points)**
    - Take a look at this paper.
    - Factors out the rotational part of a tetrahedron's deformation when computing element stress. This allows for fast, stable simulations.
- Invertible elements **(20 points)**
    - Take a look at this paper.
    - Allows for tetrahedra to invert, return to their original shape, and remain stable.
- Plasticity **(20 points)**
    - Make some part of the deformation that a mesh undergoes permanent, so that it does not fully return to its original rest shape.
    - There are different ways to implement this depending on your elasticity model:
        - O'Brien et al. - Basic Green's strain formulation
        - Müller and Gross - Formulation for corotated linear elasticity
        - Irving et al. - Formulation for invertible elements
- Fracture **(20 points)**
    - Split the mesh when stresses become sufficiently large.

    - Try the method in the Müller and Gross paper. The method in the O'Brien and Hodgins paper, while more physically accurate, is much more complicated to implement.
- Semi-implicit integration **(25 points)**
    - Take very large time steps by 'backwards' simulation: find the step that, when run backwards from where you want the simulation to end up, takes the simulation to its current state.
    - Once again, Baraff and Witkin have a good introduction.
    - You'll need to compute the derivative of node forces w.r.t. node positions. Adam Bargteil's notes have some derivations for this.
    - You'll also need to set up and solve a sparse linear system. Eigen provides some good C++ libraries for this.

Any extra features you implement must be mentioned in your README and demonstrated in your video (you can submit multiple videos to show off different features, if that's easier).

Resources

The starter code for this assignment is here: https://github.com/brown-cs-224/Simulation-Stencil
This includes a simple interactive 3D viewer for visualizing (and dynamically updating) tetrahedral meshes, as well as several example tetrahedral mesh files you can load.
These files are in the .mesh format, a line-by-line file format that resembles the .obj file format.
Lines come in one of two types:

- "v x y z" -- a vertex at location (x, y, z).
- "t i1 i2 i3 i4" -- a tetrahedron whose vertices are located at indices i1, i2, i3, and i4 in the list of vertices.

If you want to create new tetrahedral meshes, you can do so using one of the following software packages:

- Netgen
- Tetgen
- Quartet

These output their own various file formats (as there is, alas, less standardization in tet mesh file formats than for tri meshes), so you'd need to convert those to the .mesh format.

Implementation & Debugging Tips

- Start by simulating a single tetrahedron and verifying that everything works in that case.
- If your mesh has no forces applied, the deformation gradient for each element should be the identity matrix.
- A sanity check: try initializing the position of one or more vertices to be different than the rest configuration; verify that the mesh moves back to its rest state when you run the simulation.
- See the lecture slides for tips on what to do if your simulation explodes.

- The lecture slides contain a few examples of material parameters. If you want more, you can check out the tables given here and here. Be aware that most of these materials are very stiff and will likely be difficult to simulate in a stable manner. This Wikipedia page lists formulas for converting between different types of material parameters. What you want are “Lamé's first parameter” (that’s λ) and “Shear modulus” (that’s μ).
- Here’s a set of parameters that have worked for students in previous years (with midpoint and rk4 integrators for the sphere and ellipsoid and a timestep of 0.001 in view.cpp):
    - const Vector3f _gravity = Vector3f(0.f, -.1f, 0.f);
    - const float _lambda = 1e3f; //incompressibility for the whole material
    - const float _mu = 1e3f; //rigidity for the whole material
    - const float _phi = 1e1f; //coefficients of viscosity
    - const float _psi = 1e1f;
    - const float _rho = 1200.f; //density

Submission Instructions

Submit your assignment by running `cs224_handin fem` from a CS department terminal. You should run the handin script from a directory containing all the files you wish to submit. This directory must include a file named ‘README’ for the submission to be accepted.