Homework 4: Finite Element Methods

Computational Fabrication / Advanced Computer Graphics
6.807/6.839
Assigned: 10/15/2020
Due: **10/29/2020 at 11:59 PM**

Before starting this homework, please read Codebase.pdf first to get familiar with the homework codebase and running environment.

Please update your codebase by executing following command in your codebase folder:

```
git add ./*.hpp ./*.cpp
git commit -m "My assignment 3 code."
git pull
```

The main entrance of this assignment is in `assignment4/main.cpp`. You can compile the code by first entering `build` folder and executing following command:

```
cmake ../ -DCMAKE_BUILD_TYPE=Release -DHW=4
make
```

You can run your solution by

```
./assignment4/run4
```

If your program outputs "Welcome to Assignment 4", you are done with updating your codebase.

1 Introduction

In this assignment, we'll implement the finite element method (FEM) for linear material and nonlinear material, and integrate it with the rest of our computational design stack.

2 Background

2.1 The High Level

Before jumping into the specifics of the assignment, let's give a quick, high-level overview of FEM.

Associated with any soft object are two aspects: material and geometry. Materials include aspects like material density, Young's Modulus (a measurement of stiffness), and Poisson Ratio (a measurement of compressability). When uniform, density, and in fact mass in general, only affects the dynamics of the system, so we will ignore it in this assignment.

Geometry, meanwhile, encodes a structure to the problem. Surface meshes are translated into volumetric meshes (*e.g* tetrahedral and hexahedral). Each element is affected by the positions of its nodes - and thus by any touching elements. As we'll see, stiffnesses and elastic forces are calculated by summing over all of the independent elements and adding their contributions to the appropriate nodal degrees of freedom. For gradients of vector values (Jacobians), we will see that the sparsity pattern, and where the individual contributions come from, is intimately tied to the mesh structure and choice of finite element type.

Although materials and geometry both go into the final formulation of the static simulation, they have discrete meaning. Materials describe the *intrinsic* local continuum mechanics of bulk objects at any given point on the object, while geometry and meshing describe how this bulk material is translated into different local stiffnesses throughout the object.

Beyond these, there are two other elements specific to the *environment* that need to be considered. The first is boundary conditions. Which aspects of an object (degrees of freedom) are fixed to the world, unable to move? Any parts of an object that are fixed will not deform. The second is external (loading) forces. Forces, such as those caused by gravity or, say, placing an external weight on the object, are necessary for the object to experience *any* deformation, and intuitively have the most "say" in how the object deforms. For instance, it should be intuitive that if you drop a marble on a vat of jello, the jello will locally sink downward below the marble - not to the right or left, or forward or back - and it certainly won't advect upward.

All of these elements come together to form a system with n vertices (nodes):

$$\mathbf{K}\mathbf{U} = \mathbf{f} \tag{1}$$

where $\mathbf{K} \in \mathbb{R}^{3n \times 3n}$, $\mathbf{U} \in \mathbb{R}^{3n}$, $\mathbf{f} \in \mathbb{R}^{3n}$ (here the 3 comes from the fact that, in 3D, the mesh will have 3 degrees of freedom per node; in 2D, the mesh would have 2 degrees of freedom per node; in other dimensions it translates accordingly). Here, the $\mathbf{K}$ depends on materials and the mesh structure, $\mathbf{f}$ depends on the external loads, and $\mathbf{U}$, for linear systems, denotes how much each degree of freedom has moved from its resting (unloaded) position when in static equilibrium. Rows and columns corresponding to fixed degrees of freedom are removed from the system prior to solve. At the end of the day, if we can formulate $\mathbf{K}$ and $\mathbf{f}$, we can solve for $\mathbf{U}$ and calculate the final mesh vertex positions $\mathbf{q} = \mathbf{q}_{rest} + \mathbf{U}$.

2.2 The Slightly Lower Level

Let's get a bit more technical. Recall Newtonian mechanics - given a system with degrees of freedom $\mathbf{q} \in \mathbb{R}^{3n}$, define an energy function $\mathbf{E}(\mathbf{q})$. The forces in the system are the negative gradients of the energy with respect to the degrees of freedom of the system; namely $\mathbf{f}_e = -\nabla_{\mathbf{q}}\mathbf{E}(\mathbf{q})$. Here, we use the subscript e to denote that this is the elastic force of the system.

At equilibrium, it should be true that $\mathbf{f}_e(\mathbf{q}) + \mathbf{f}_{ext} = 0$, where the subscript ext denotes the external forces of the system. In other words, given the external force vector f_{ext}, we need to find a position vector q so that $\mathbf{f}_e(\mathbf{q}) = -\mathbf{f}_{ext}$.

For linear model, we have $\mathbf{K} = -\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is constant over all q. We can solve this system by simply solving a linear system:

$$\begin{aligned} &\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})(\mathbf{q} - \mathbf{q}_{rest}) = -\mathbf{f}_{ext} \\ \Rightarrow &\mathbf{K}(\mathbf{q} - \mathbf{q}_{rest}) = \mathbf{f}_{ext} \\ \Rightarrow &\mathbf{KU} = \mathbf{f}_{ext} \end{aligned} \tag{2}$$

Thus, the game goes something like this:

1. Derive a form of the energy density of the system.
2. Integrate that energy over the hexahedral mesh.
3. Differentiate energy with respect to $\mathbf{q}$ to calculate the elastic force.
4. Differentiate the elastic force with respect to $\mathbf{q}$ to generate the $\mathbf{K}$ matrix.
5. Solve a linear system.

For nonlinear elasticity materials, $\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is not a constant value any more, so we need to use Newton's method to find the root of this nonlinear equation.

Please read the attached document **FEM_cheatsheet.pdf** carefully to learn how to derive the $\mathbf{K}$ matrix. (Optional) You can also refer to [1] as a detailed tutorial.

3 Assignment

This homework has three main parts. In the first part, you'll implement linear FEM to compute the deformation of the tetrahedral mesh under some external force. This amounts to computing a stiffness matrix $\mathbf{K}$, the loads on the vertices $\mathbf{f}$, formulating a linear expression, $\mathbf{KU} = \mathbf{f}$, and solving for $\mathbf{U}$, a vector of vertex displacements. In this part, you need to fill some functions including computing the stress differential and formulating the stiffness matrix $\mathbf{K}$.

In the second part, you'll use the nonlinear material model provided by us and implement a simple Newton solver to solve for a nonlinear material model.

In the third part, you'll integrate this assignment with the rest of your codebase. Specifically, you'll take one of the meshes you've used in assignment 3, apply your customized boundary conditions and compute the deformation of it.

3.1 FEM with Linear Material Model (40 points)

We direct the reader to the attached document **FEM_cheatsheet.pdf**, which provides a more extensive derivation of several constitutive models. Here, we provide a brief overview of the material for the assignment.

The first component we need to have is the linear material model. As a function of the deformation gradient, the energy density Ψ of a linear material can be written as:

$$\Psi(\mathbf{F}) = \mu\epsilon : \epsilon + \frac{\lambda}{2}\mathrm{Tr}^2(\epsilon) \tag{3}$$

where $\mathbf{F}$ is the deformation gradient (some places, like in the code, we use $\mathbf{f}$ for deformation gradient, but we'll use $\mathbf{F}$ in the writeup so we don't confuse it with force). ϵ is the small strain tensor ":" is the tensor contraction, and μ and λ are Lamé parameters, which can be calculated from the Young's Modulus and Poisson's ratio (please see the table here for conversions: https://en.wikipedia.org/wiki/Lam%C3%A9_parameters).

Differentiating, we achieve:

$$\mathbf{P}(\mathbf{F}) = \frac{\partial \mathbf{\Psi}(\mathbf{F})}{\partial \mathbf{F}} = \mu(\mathbf{F} + \mathbf{F}^T - 2\mathbf{I}) + \lambda\,\mathrm{Tr}(\mathbf{F} - \mathbf{I})\mathbf{I} \tag{4}$$

As can be seen, the Piola-Stress Tensor ($\mathbf{P}$) (a measure of pressure, or force per cross-sectional area) is linearly dependent on the deformation gradient. This is good, because it implies that the elastic force is linear in the mesh deformation - implying that our eventual $\mathbf{K}$ matrix will be constant. In order to construct the stiffness matrix $\mathbf{K}$, you need to implement a function to compute the stress differential $\frac{\partial \mathbf{P}(\mathbf{F})}{\partial \mathbf{F}}$.

The remaining part necessary for implementing FEM is to implement the stress differential function in **Common/FEM/linear_material.hpp**. The stress differential should be the derivative of $\mathbf{P}$ with respect to $\mathbf{F}$. Your code only has to work for dim=3, though if you implement it for general dim your system can be extended to other dimensional systems.

The second step is to construct the stiffness matrix, $\mathbf{K} = -\nabla_{\mathbf{q}}\mathbf{f}_e$. The function to compute it is **ComputeStiffnessMatrix** in **Common/FEM/tet_deformable_body.hpp**. Computing $\mathbf{K}$ needs to numerically integrating the stiffness over the tetrahedral mesh. In order to do that, the discretization rule should be applied. We compute the contribution from each tetrahedron element individually. The computation can be performed *via* the chain rule, as shown on the attached note. A good property of such discretization is that the $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ is constant over the whole tetrahedron, which helps us simplify the integration inside each element. After you've computed the contribution from each single tetrahedron, the global stiffness matrix is the sum of contributions of all stiffness matrices of each tetrahedron (indexed to the correct dimension of the system).

We have implemented the simulation function for linear material **test_linear_material** in **Assignment4/main.cpp**. The function computes the static equilibrium force for a beam as shown in Figure 1. The left side of the beam is fixed and the external force is applied on right region of the bottom. The function calls your implemented stiffness matrix function to get $\mathbf{K}$ matrix and solve $\mathbf{KU} = \mathbf{f}_{ext}$ by Eigen library. You can visualize the computed results in the data folder by **Meshlab**.

You can change the size of your beam at the top of `Assignment4/main.cpp` to test your code on problems in different scale.

HINT: To aid you in debugging all of your steps so far, we have provided results from our solution code. The first beam is with size $4 \times 2 \times 2$, and the second beam is with size $30 \times 10 \times 10$. We provided both the K matrices and the final deformed mesh for them. You can find them in the folder `data/assignment4/std/`, and they are `K_*_linear_std.txt` and

`deformed_*_linear_std.stl`. For the youngs modules and poisson's ratio, the generated results use the default value in the code.

Summarize your task in this section here:

> 1.1 (10 points) Compute $\frac{\partial \mathbf{P}(\mathbf{F})}{\partial \mathbf{F}}$ for linear material: Implement the **StressDifferential** function in **Common/FEM/linear_material.hpp**.
>
> 1.2 (30 points) Compute stiffness matrix **K**: Implement the **ComputeStiffnessMatrix** function in **Common/FEM/tet_deformable_body.hpp**.

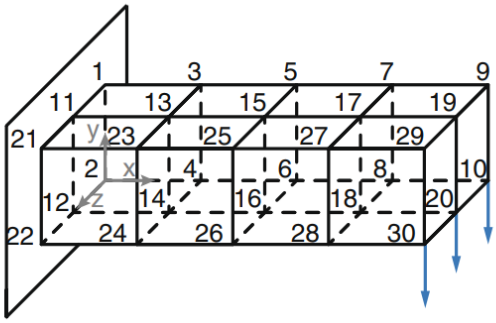

Figure 1: An example of how your voxel grid loading and contraints should look (dimensions may vary)

3.2 Nonlinear Material (40 points)

Linear material model gives a descent approximation when deformation is small. When large deformation is desired, linear material model will lead to unrealistic results. In this part, we are going to try nonlinear material model.

Unlike linear material model, the static equilibrium deformation for nonlinear material cannot be solved by a single linear equation. For nonlinear elasticity materials, $\mathbf{K}(\mathbf{q}) = \nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})$ is not a constant value any more, so we need to use Newton's method to find the root of this nonlinear equation. Roughly speaking, in order to solve $\mathbf{f}_e(\mathbf{q}) = -\mathbf{f}_{ext}$, Newton's method linearize the function around current solution $\mathbf{q}_i$ and update the solution $\mathbf{q}_{i+1}$ by solving the linear system:

$$\nabla_{\mathbf{q}}\mathbf{f}_e(\mathbf{q})|_{\mathbf{q}_i}(\mathbf{q} - \mathbf{q}_i) + \mathbf{f}_e(\mathbf{q}_i) = -\mathbf{f}_{ext} \tag{5}$$

By iteratively solving such local linear system, Newton's method will converge to the solution to the original nonlinear system.

We have implemented a nonlinear material model (Neohookean material) for you, and you need to implement a Newton's method in **test_nonlinear_material** function.

Here's the list of the tasks in this section.

2.1 (30 points) Implement Newton's method in **test_nonlinear_material** to solve static equilibrium force for nonlinear model. This function is in **Assignment4/main.cpp**.

2.2 (10 points) Compare linear model and nonlinear model by choosing appropriate parameters (size, boundary condition) for beam. Describe the pros and cons of both methods in your report based on your designed example.

HINT: To aid you in debugging all of your steps so far, we have provided results from our solution code. The first beam is with size $4 \times 2 \times 2$, and the second beam is with size $30 \times 10 \times 10$. We provided the final deformed mesh for them. You can find them in the folder `data/assignment4/std/`, and they are `deformed_*_nonlinear_std.stl`. For the youngs modules and poisson's ratio, the generated results use the default value in the code. Your results are not necessary numerically same as the solution for nonlinear case because of the different implementations (termination and line search) of Newton's method, but they should be visually same.

3.3 Integrating Assignment 3 (20 points)

With your linear and nonlinear solver in tow, you are now prepared to simulate one of your 3D meshes from a previous assignment. Recall in assignment 3, we first translate a triangle mesh or voxelized mesh to a tetrahedral mesh and then do the deformation by BBW. Now, you are going to use fem to simulate your favorite object in assignment 3. You can choose your favorite object in assignment 3 and write code to output its tetrahedral mesh. In this assignment, you need to write code to read that tetrahedral mesh, apply your own customized boundary condition on it, and compute the static equilibrium deformation. The tetrahedral mesh can be saved in any format you want (you just need to make sure that you can consistently read that file and translate it to the correct tetraheral mesh in assignment 4).

3.1 (20 points) Implement to test one object from assignment 3 with your customized boundary condition. The functions to be filled are **test_customized_model** and **set_customized_boundary_conditions**. You can choose to use linear material model or nonlinear material model. You may need to write code to write tetrahedral mesh to file in assignment 3 and to read tetrahedral mesh from file in assignment 4.

Tips: If your system has too many degrees of freedom, it might take a very long time to solve. Make sure you always construct the stiffness matrix $\mathbf{K}$ as a sparse matrix.

3.4 Extra Credit

3.4.1 Dynamics (30 points)

FEM can be applied to more than just static simulation; it can also be applied to dynamical systems as well. Given a mass matrix $\mathbf{M}$ and some scalar timestep Δt, one can formulate the dynamical system:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{f}_c + \mathbf{f}_{ext} \tag{6}$$

for some constant loading $\mathbf{f}_{ext}$.

One can solve the following system of equations, and integrating forward using an Euler step:

$$\dot{\mathbf{q}}_k + \mathbf{M}(\mathbf{q})^{-1}(\mathbf{f}_c(\mathbf{q}) + \mathbf{f}_{ext})\Delta t = \dot{\mathbf{q}}_{k+1} \tag{7}$$

$$\mathbf{q}_k + \dot{\mathbf{q}}\Delta t = \mathbf{q}_{k+1} \tag{8}$$

In order to implement dynamic FEM, then, you will need to:

1. Construct the mass matrix $\mathbf{M}$.
2. Integrate forward using a forward or backward Euler step (we left certain timestep labels ambiguous so you could choose which).

In doing this, you may need to augment the definition of your deformable bodies to include velocities.

Be warned - for such a system, if completely unconstrained, the solution to such a problem will be, as in the static case, rigid motion. Therefore, you will, as before, have to fix certain degrees of freedom of your system by deleting the appropriate rows/columns of certain matrices and vectors.

4 Submission Guidelines

All assignments must be submitted onto the Canvas submission system by the deadline. The submission should be in one single .zip file, including two parts.

1. All the code in `assignment4` folder and `Common` folder should be submitted.
2. You should provide the file of the tetrahedral mesh which can be read by your assignment 4 code in part 3.1.
3. A short .pdf report describing how your code works and the results.

Late assignments will incur a 25% penalty for each late day (3 late days are permitted without penalty throughout the semester). No assignments will be accepted after the solutions are released (typically within a few days of the deadline; contact the TA for details).

4.1 Writeup Instructions

The writeup should be relatively short (around $1-2$ page). In your writeup, please tell us the following:

- At a high-level, what did you implement in order to complete the assignment?
- How does that code work, algorithmically?
- Some pictures to show your results.
- What did you like about the assignment?
- What did you not like about the assignment?
- Any other issues about which we should be aware?

Please submit the writeup as a .pdf. Also, feel free to include any relevant images (embedded in the .pdf).

Please submit the writeup as a .pdf. Also, include any relevant images (embedded in the .pdf).

References

[1] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, pages 20:1–20:50, New York, NY, USA, 2012. ACM.