CGGS Coursework 2 (Simulation): Soft-Body Simulation

Amir Vaxman

Errata

- None.

Introduction

This is the 2^{nd} coursework for the "simulation" choice. The purpose is to write `C++` code for a simplified simulation of deformable bodies, with the linear finite-element method (FEM) approach to soft-body simulation, learned mostly in lectures 13 and 14. Visualization will be done with PolyScope as in all other Practicals. The concrete objectives are:

- implement the stiffness, mass, and damping matrices that are used in the velocity integration.
- Implement discrete-time integration for the soft-body FEM equation, to obtain velocities and positions.
- Extension: implement the corotational approach for large deformations.

General guidelines

This coursework uses the same Eigen + PolyScope setup as in CW 1 (simulation). Your job is to complete several functions, given by signatures in the code, as described in the sections below. The code that you need to write is all within the header files `Scene.h` and `Mesh.h`. The main script is `Section12` and the sections' respective grading scripts are `Grading1` and `Grading2`.

This CW will depart from the setting of CW1 (Simulation), as we will not do collisions or gravity; instead, we will focus on shape physical deformation, reacting to initial velocities.

Important note: This is a very basic simulation and moreover with some "hacks" to make the simulation much simpler to implement. Expect the results to be more or less plausible but far from what you would perceive as physical. We are also using a low-tet count since none of the elements of our algorithm are

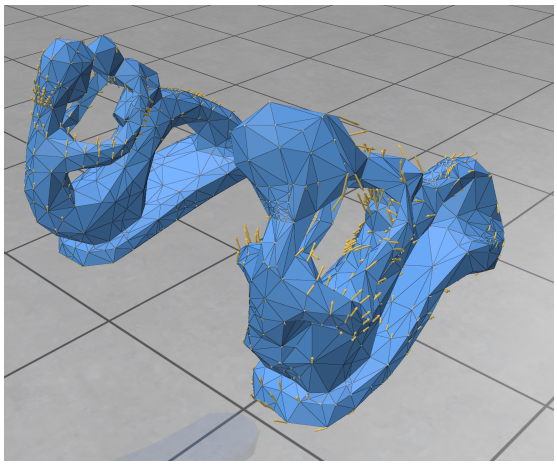

Figure 1: Results of the simulation for the `fertility-scene.txt`, where the same mesh with different material properties exhibits different behaviors. You can view the current velocity as the GUI feature `Velocity Field`

optimized, which adds to the coarseness. Note especially the artifacts of using linear FEM where objects often arbitrarily shear and scale.

1 File and Scene Formats

Soft-body deformation is considerably more complex than rigid-body deformation, and therefore we adhere to a specific formatting of the code. There are two classes involved:

- The `Mesh` class implements a single mesh in the scene. This is where you will implement the individual FEM matrices (See below).
- The `Scene` class is where the main action takes place: this is where you will integrate velocities and positions.

1.1 Global and local indexing

The FEM matrices used in the CW will work on *all vertices in the scene at once*. Furthermore, it works on them as a single vector in the vertex-dominant format $xyz, xyz, xyz, \cdots$, vertex after vertex. Thus, we keep two copies of the original and current vertices in the scene: one in the individual meshes, in size $3|V_m| \times 1$ format (for mesh m), as `mesh.origVertices` (original locations) and `mesh.currVertices` (deformed locations), and a *global* one in the `scene` class, as `globalOrigVertices`, `globalCurrVertices`, and `globalVelocities` for the current vertex velocities. They are all of the size $|V|$ where $|V| = \sum_m |V_m|$ over all meshes. The vertices in the meshes are mapped to the global one by order

of mesh and then vertices in the mesh. The (already implemented) function `global2Mesh()` copies global values back to local ones, and `mesh2global()` does the opposite. Except for the definition of the individual FEM matrices, you only work with the global indexing in this CW. Integration is done on them. All the translation between local and global has already been handled (and is not very interesting in the context of what you need to do).

1.2 Scene file format

Soft-body deformation requires quite a few parameters. The loaded scene file is then of the following format:

```
#num_meshes
#name0 density0 YoungMod0 PoissonRatio0 #isfixed0 #xyz_position0, #orientation0
#name1 density1 YoungMod1 PoissonRatio1 #isfixed1 #xyz_position1, #orientation1
...
#num_velocities
#num_mesh #vertex_in_mesh vx vy vz
...
```

The position and orientation are just for the original setup of the object, and you shouldn't use them. The three important parameters are the density ρ, Young's modulus Y, and Poisson ratio ν, corresponding to the quantities learned in class. The meshes are of the `.mesh` format which encodes tetrahedral meshes. The second part of the scene file prescribes initial velocities for the prescribed vertices that go into the $t = 0$ `globalVelocities`. They simulate initial impulses that drive the meshes to immediate deformation, before they settle down. You will see (Fig. 1) how the same mesh reacts differently according to material properties. Experiment with changing these!

2 Linear FEM Matrices

In this section, you will compose the stiffness matrix K_m, mass matrix M_m, and dampening matrix D_m for every mesh m, where this should be encoded in `Mesh::compute_soft_body_matrices()`. You will find details on how to compose them in the lectures and lecture notes. They are all square *sparse* matrices of size $3|V_m| \times 3|V_m|$ for the vertices V_m, where they will eventually be appended to big matrices K, M, D of the entire scene, in `Scene::init_scene()`, which you should also implement. Use `sparse_block_diagonal()` to piece the individual mesh matrices together to big full-scene matrices. In the latter function, you must then also initiate the linear solver `ASolver`, so that it will be efficiently solving the velocity integration equation in every time frame.

Note that you will need to compute the Lamé parameters λ, μ from Y and ν, to form the stiffness tensor C. The dampening matrix will use the α, β parameters in the Rayleigh dampening: $D = \alpha M + \beta K$.

The matrices you created will be checked by the grader, and this is worth 50% of your grade directly. There is no report component in the section specifically, as it does not culminate in a visible simulation result just yet.

3 Constrained Time Integration

Your task is to integrate velocities from $v(t)$ to $v(t + \Delta t)$ using the implicit-integration equation learned in class:

$$(M + \Delta t D + (\Delta t^2)K)v(t + \Delta t) = Mv(t) - \Delta t K(x(t) - x(0)) + f_e$$

The left-hand side should have been factorized by `ASolver` in `init_scene()`, to make the solving fast. the function to complete is `Scene::integrate_global_velocity()`. Results should be fed into `globalVelocities`. Also complete `Scene::integrate_global_position()` for integrating `globalCurrPositions`. Again, only update global positions and not individual mesh positions. This should use the `ASolver` that was pre-factorized at $t = 0$ for the fixed left-hand side, which will then be used to solve fast for the changing right-hand sides.

This Section will be 20% of your grade, with 10% for the automatic grader, and 10% for the report detailing your insights on the results (and artifacts and advantages and disadvantages) of linear FEM on the existing scenes. You are encouraged to generate new scenes to demonstrate your insights more clearly.

4 Extension: corotational elements

This section is a more advanced extension of the practical, which is worth the final 30% of your total grade. It will not be checked automatically, but rather entirely depend on your demonstration and explanations in the report.

You will have noticed that linear FEM allows for quite a few scale and shear artifacts of deformation. To mitigate this, you should implement the corotational approach to computing the stiffness matrix taught in class. For this, in every iteration, you will need to compute the rotational part of the forces in each tet, and factor them in the individual element K_e before mixing it into the mesh K_m and then the big scene K. Demonstrate the difference clearly in the report, which should be evident where big deformations are involved. Note that this will not allow you to prefactor the left-hand side that keeps changing, so it will be slower.

5 Submission

You should submit the following:

- A report that must be at most 3 pages long including all figures, according to the instructions above, in PDF format.

- For Sections 1 and 2, only `Scene.h` and `Mesh.h`. For Section 3, package everything *separately from the other sections*.
- Any extra data files of scenes or constraints if you need them.
- You are encouraged to submit videos of your simulation with specific close-ups to demonstrate specific effects, and refer to them in the report.

The submission will be in the official place on Learn, where you should submit a single ZIP file of everything.