CGGS Coursework 2: Debugging Geometry and Simulation

Amir Vaxman

Errata

- None.

Introduction

This is the 3^{nd} coursework, which is mandatory for everyone in the course. This coursework is quite different from the first two: you are already getting the full code for two algorithms: ARAP deformation, and multi-body rigid simulation. Furthermore, there are no graders. However, both have a few minor bugs "planted" in them, and your role is to figure out these bugs and fix them, with the principles learned in Lecture 16. Visualization will be done with PolyScope as in all other Practicals.

General guidelines

This coursework uses the same Eigen + PolyScope setup as in all other coursework. The codes are however highly simplified, and everything important is in the respective `main.cpp` files of the Section1,2 subprojects. You do not have to submit *any* code for this CW, and there is no automatic grader; you will submit a report (of 3 pages as usual), describing the process which you took to debug and find the problem in each of the algorithms. You should describe the following for each algorithm:

- How you assessed that the result is wrong.
- How you isolated the problem.
- The fix you applied.
- How you tested that the result had been corrected.

Lecture 16 will include all principles for assessing correctness, and best practices for debugging. It is best if you explicitly quote these principles (as in the slide titles, if you wish), when explaining your process. Each section is worth

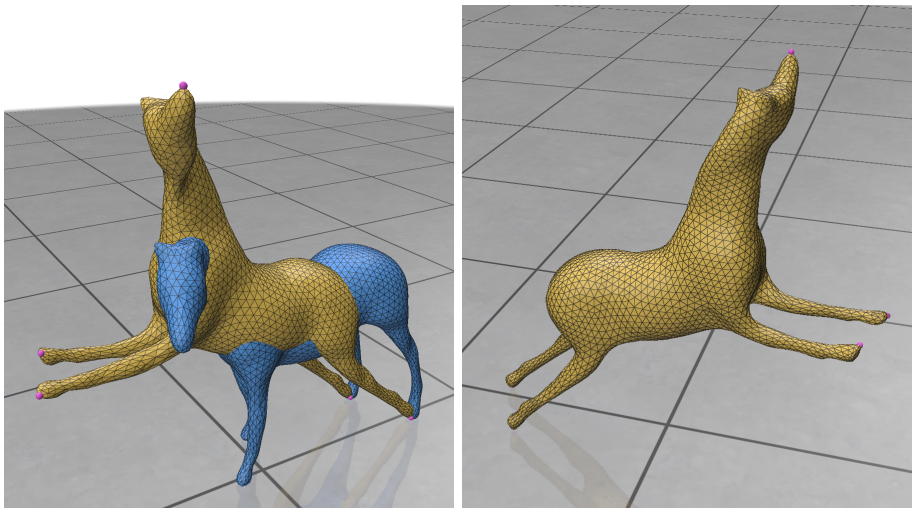

Figure 1: The correct solution for ARAP, with the fixed handles provided in the original script.

50% of the coursework's grade, to a full 100%. In the following, we explain both algorithms used, and exemplify the correct result.

We planted **exactly 3 bugs** in each section; if you found them, consider your job done.

1 As-Rigid-as-possible Deformation

You get a full implementation of the ARAP algorithm, controlled from the function `ARAP_deformation()`. The algorithm is described in Lecture 15 and the lecture notes. We urge you to explore the code and its relation to the theory. Explore the data structures that the function obtains as input, where specifically `oneRings` that encodes the neighboring vertices to a vertex. For simplicity, the algorithm hard-codes the number of iterations run in `numIterations`. The correct result is as it looks in Figure 1.

2 Multi-body Rigid Simulation

This section, operated within the call back function, simulates a set of randomly placed spheres rigidly, where they are treated as $1[Kg]$ objects of a uniform `radius`, and where the major differences from what you already have been simulating are:

- There is a attracting force of gravity $F_{12} = -F_{21} = \frac{G \cdot n_{12}}{(r_{12})^2}$ between any two spheres 1 and 2 with normalized vector n_{12} between them and distance r_{12}. G is dictated by `gravityConstant`.

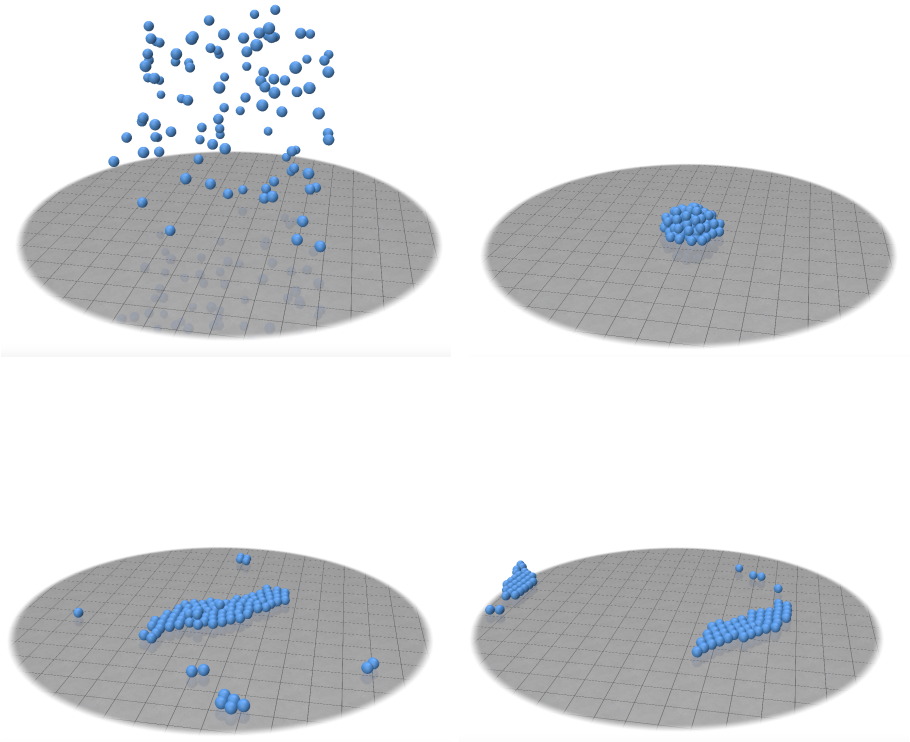

Figure 2: Left to right, top to bottom: the progression of the simulation with the default parameters. Note the interesting bunching and rotation patterns emerging before the structure breaks apart on the floor.

- Collision between the objects and the floor is done explicitly, since these are simple spheres. It has a few tweaks to stabilize the simulation (like disabling gravity when objects are too close, to avoid near-infinity forces).

Otherwise, the algorithm proceeds in a classical way of working with impulses and restitution (Lecture 7), except the masses are hard-coded as $1[Kg]$ and there is no inertia tensor, nor friction. The results should look as in Figure 2.

3 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.
- Any extra data files of if you need them.

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