ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computational Engineering

Spring semester 2014

Prof. P. Koumoutsakos, G. Tauriello
ETH Zentrum, CLT F 12, C 11
CH-8092 Zürich

Solution 2

Issued: 25.3.2014

Question 1: Code reading

The class `Simulation` includes a private class variable `particles` of type `ParticleVector` which stores all particles. The i-th particle ($i = 0, \ldots, n-1$) can be accessed as `particles[i]` and n is the return value of `particles.getNumberOfParticles()`. The particles obtained by the call `particles[i]` are of type `Particle` and can be modified using the methods defined there. An example of how to use the `Force` class is given as a comment in `Force.h`.

The main loop of the time integration is performed in `Simulation::run`. The actual update of positions and velocities is done in `Particle::update`. The time integration scheme being used is forward Euler. Note that here the `_computeAccelerations` method makes sure that we compute all accelerations before updating the particles' positions.

Question 2: Computing the pairwise forces

```
1  // compute force felt by this particle due to particle p
2  Force computeLocalInteraction(const Particle& p) const
3  {
4      // Instantiate a new force
5      Force f;
6
7      // Plummer radius.
8      const double pr = 0.3;
9
10     // Compute vector connecting particles
11     const double r[3] = {
12         p.x[0] - x[0],
13         p.x[1] - x[1],
14         p.x[2] - x[2]
15     };
16
17     // Compute distance, direction of the force and its magnitude
18     const double rSq = r[0]*r[0] + r[1]*r[1] + r[2]*r[2];
19     const double magnitude = m*p.m/std::pow(rSq + pr*pr, 1.5);
20
21     // Set components through mutator
22     for (int i=0; i<3; i++) f[i] = magnitude*r[i];
23
24     // Return force
25     return f;
26 }
```

Listing 1: Particle.h

Question 3: Computing the accelerations

```cpp
// Helper: compute accelerations
void Simulation::_computeAccelerations()
{
    // NOTE: code guarantees that all particles' accelerations
    //       are set to 0 before this function is called!!

    // Get number of particles
    const int n = particles.getNumberOfParticles();

    // Compute local interactions and update accelerations
    for (int i = 0; i < n; ++i) {
        // loop to consider each pairwise interaction only once
        for (int j = i+1; j < n; ++j) {
            // get alias to particles
            Particle& p1 = particles[i];
            Particle& p2 = particles[j];

            // compute Fij
            const Force Fij = p1.computeLocalInteraction(p2);

            // add accelerations to p1 and p2
            p1.addAcceleration(Fij);
            p2.addAcceleration(Fij.opposite());
        }
    }
}
```

Listing 2: Simulation.cpp

Question 4: Run code

The initial and final frame for a simulation starting with `ic.txt` are shown in Fig. 1. When we compare the final frames of `ic.txt` and `ic2.txt` (Fig. 2 we notice that they diverge a lot even though their initial position was almost identical. This is expected for this system as it is chaotic, essentially showing extreme sensitivity to the initial condition. Chaotic systems trajectories show exponential divergence over time.

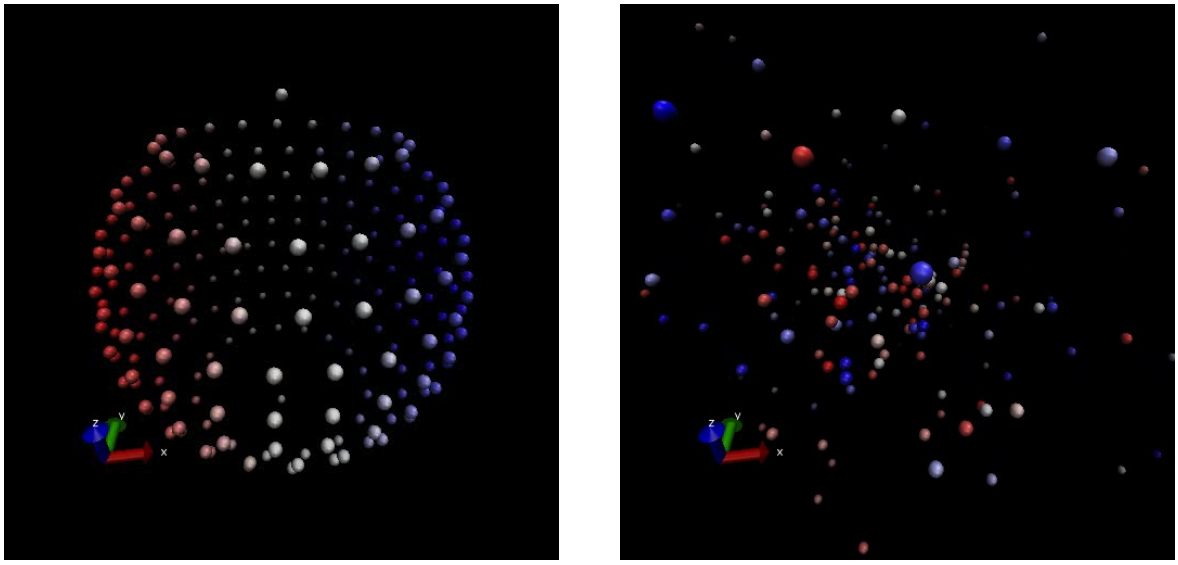

Figure 1: Initial and final frame for ic.txt

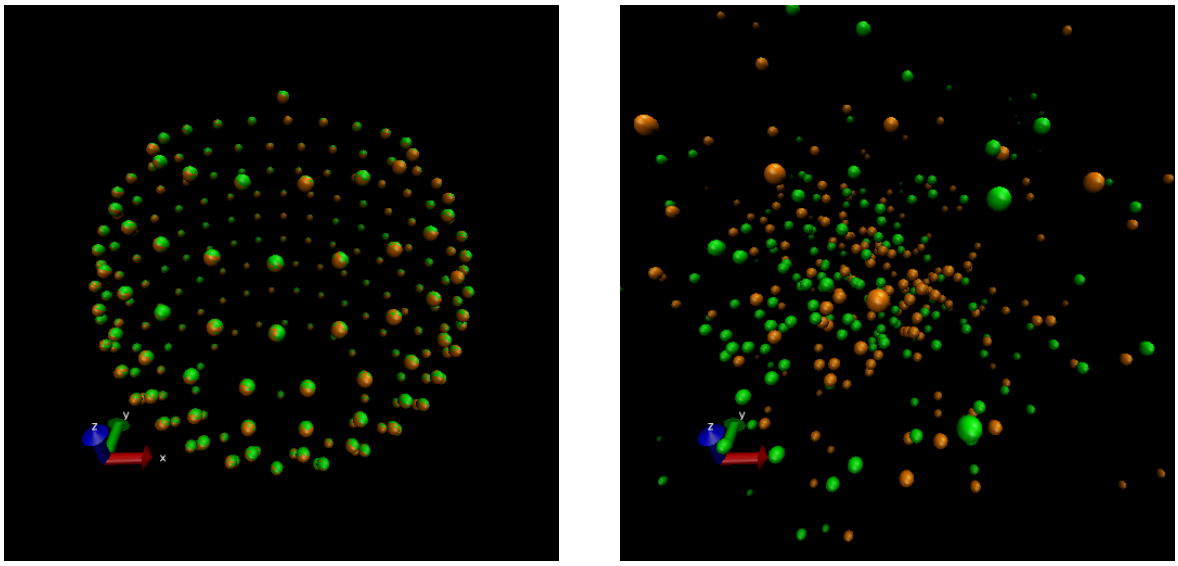

Figure 2: Initial and final frame comparing ic.txt and ic2.txt. Green and orange colored particles indicate two different runs with almost identical initial conditions