Physics 750: Exercise 9

Thursday, October 5, 2017

1. Use the `curl` command to download from the class website everything you'll need for this exercise.

```
$ WEBPATH=http://www.phy.olemiss.edu/~kbeach/
$ curl $WEBPATH/courses/fall2017/phys750/src/exercise9.tgz -O
$ tar xzf exercise9.tgz
$ cd exercise9
```

2. As an example of how to proceed, I have included a program `solar.cpp`, which simulates three small "planets" orbiting a larger "star" in an artificial (and unstable) solar system. All four bodies interact via the graviational force (with $G = 1$) and are confined by their initial conditions to the x-y plane. From the `exercise9` directory, invoking

```
$ make
g++ -o solar_openGL solar.cpp -O2 -ansi -pedantic -Wall -lglut -DUSE_OPENGL
g++ -o solar solar.cpp -O2 -ansi -pedantic -Wall
```

will build both graphical (`solar_openGL`) and non-graphical (`solar`) versions of the program. A single source file `solar.cpp` contains OpenGL code that is selectively activated with the `USE_OPENGL` macro.

Running `./solar_openGL` opens a new window and animates the planetary motion. The window displays everything inside the coordinate square with vertices $(-1, -1)$ and $(1, 1)$. You can adjust the speed of the animation by recompiling with different values of `const int delay`.

The nongraphical version writes the four trajectories to files in a 7-column format. (N.B. The `<<` operator is overloaded in the `particle` class definition, so `cout << gas[n]` is a legal way to output information about the nth particle.) Try the following:

```
$ ./solar
$ ls tr*
trajectory0.dat trajectory1.dat trajectory2.dat trajectory3.dat
$ gnuplot view.gp
```

3. We now want to investigate the behaviour of a two-dimensional gas of particles interacting via Van der Waals forces. Let's set up a separate program to do that.

```
$ cp solar.cpp LennardJones.cpp
$ pico makefile
[edit makefile accordingly]
```

The energy of the gas has the form

$$E = \sum_i \frac{1}{2} m v_i^2 + \sum_{i<j} U(r_{ij}),$$

where the indices i, j range over the particle number, r_{ij} is the distance between two particles, and $U(r)$ is given by the famous Lennard-Jones potential:

$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right].$$

The force between a pair of particles,

$$\boldsymbol{F} = -\boldsymbol{\nabla} U(r) = \frac{24\epsilon}{r}\left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]\hat{r},$$

vanishes at $r_0 = 2^{1/6}\sigma$. Particles closer than r_0 repel each other and those farther apart attract. Implement this new force law in your code using the values $\epsilon = 1$, $\sigma = 0.21$. You may have to adjust your time step.

Unlike the solar system, the gas will need to be contained. Restrict the particles to a 2×2 box by implementing periodic boundary conditions for the coordinates $-1 < x < 1$ and $-1 < y < 1$. You will have to modify the concept of distance accordingly.

Next, initialize the gas by arranging 100 particles at rest in a 10×10 square grid with uniform spacing:

$$\begin{bmatrix} (-0.9, 0.9) & (-0.7, 0.9) & (-0.5, 0.9) & \cdots & (0.9, 0.9) \\ (-0.9, 0.7) & (-0.7, 0.7) & (-0.5, 0.7) & & (0.9, 0.7) \\ \vdots & & & & \vdots \\ (-0.9, -0.9) & (-0.7, -0.9) & (-0.5, -0.9) & \cdots & (0.9, -0.9) \end{bmatrix}$$

Introduce small random deviations (of order 0.01) in the positions. You should use the mersenne twistor algorithm from C++11's `random` library

```
#include <random>
using std::mt19937;
using std::uniform_real_distribution;

#include <chrono>
using std::chrono::high_resolution_clock;

#include <functional>
using std::bind;

static mt19937 rng; // mersenne twistor random number generator
// create a function that generates a random number in [-1,1)
static auto rnd = bind(uniform_real_distribution<double>(-1,1),rng);
```

to place particles at `vec2(x+0.01*rnd(),y+0.01*rnd())`.

With these initial conditions, the system is actually in the solid phase. But the initial square lattice pattern is not the ground state; it is merely meta-stable. The slight random deviations you added allow the atoms to settle into into a new crystalline arrangement. Describe that arrangment.

4. Keep track of the total elapsed time t and have the program exit when $t > 20$. Use the `exit(0)` function call in the C standard library:

   ```
   #include <cstdlib>
   using std::exit;
   ```

   Generate a histogram of all the pair distances in the time window $10 < t < 20$. This quantity is called the radial distribution function. Plot it and discuss its features. Explain the geometric significance of the first few peaks.

5. Initialize the particles in their 10×10 grid as before, but reduce the atomic "size" to $\sigma = 0.1$ (in effect creating a more dilute gas) and add small deviations (of order 4) to the initial particle velocity, e.g., `vec2(4.0*rnd(),4.0*rnd())`. Measure a velocity histogram in each of the time intervals $0 < t < 1/8$, $1/4 < t < 1/2$, $1 < t < 2$, $4 < t < 6$, and $16 < t < 20$. Compare these with the two-dimensional Maxwell-Boltzmann distribution,

   $$f(v) = \frac{mv}{kT}\exp\left(-\frac{mv^2}{2kT}\right).$$

   Estimate the time scale for the system to thermalize. Can you perform a fit to extract the temperature kT?

6. What does the radial distribution function look like for the dilute gas? Plot it alongside the analytical expression for a uniform, non-interacting gas. How do the two plots differ and what does that reveal?

**7. For each of the 100 particles in the gas phase, add a partner of mass 3.0 randomly positioned a distance $d = 0.05$ away with the same initial velocity. Use the method of Lagrange multipliers to enforce the distance constraint at every time step going forward.

For example, suppose that particle i and j form a bonded pair. The corresponding Lagrange multiplier λ_{ij} appears in the equations of motion and works to keep the particles locked together at a fixed distance d. Since we view λ_{ij} as constant over each time step, we can add its contribution to the updates as follows:

$$\begin{aligned} \boldsymbol{r}_i^{(n+1)} &:= \tilde{\boldsymbol{r}}_i^{(n+1)} + \frac{\lambda_{ij}(\Delta t)^2}{m_i}(\boldsymbol{r}_j^{(n)} - \boldsymbol{r}_i^{(n)}) \\ \boldsymbol{r}_j^{(n+1)} &:= \tilde{\boldsymbol{r}}_j^{(n+1)} + \frac{\lambda_{ij}(\Delta t)^2}{m_j}(\boldsymbol{r}_i^{(n)} - \boldsymbol{r}_j^{(n)}). \end{aligned}$$

Here, the tilde quantities denote the new positions computed (with velocity Verlet) in the absence of constraint forces. The requirement that $|\boldsymbol{r}_i^{(n+1)} - \boldsymbol{r}_j^{(n+1)}| = d$ yields a quadratic equation in λ_{ij} whose physical root is

$$\begin{aligned} \lambda_{ij}(\Delta t^2) &= \left(\frac{1}{m_i} + \frac{1}{m_i}\right)^{-1}\left(\frac{\sqrt{(\delta\tilde{\boldsymbol{r}}\cdot\delta\boldsymbol{r})^2 + |\delta\boldsymbol{r}|^2(d^2 - |\delta\tilde{\boldsymbol{r}}|^2)} - \delta\tilde{\boldsymbol{r}}\cdot\delta\boldsymbol{r}}{|\delta\boldsymbol{r}|^2}\right) \\ &= \frac{\mu_{ij}}{d}\left(\sqrt{d^2 - |\delta\tilde{\boldsymbol{r}}|^2\sin^2\theta} - |\delta\tilde{\boldsymbol{r}}|\cos\theta\right). \end{aligned}$$

The vectors $\delta\boldsymbol{r} = \boldsymbol{r}_i^{(n)} - \boldsymbol{r}_j^{(n)}$ and $\delta\tilde{\boldsymbol{r}} = \tilde{\boldsymbol{r}}_i^{(n+1)} - \tilde{\boldsymbol{r}}_j^{(n+1)}$ meet at an angle θ.

If you choose to implement the inner product version of the expression above, you may find it convenient to call the `dot` function provided in `vec2.hpp`. Also, if you compile with `-DUSE_CONSTRAINT`, the `display` function will know to draw a line between the two atoms in each molecule. This should work correctly provided that you obey the convention that `gas[0]` and `gas[1]` are bonded, `gas[2]` and `gas[3]` are bonded, etc.

8. For the diatomic gas, compute the following three energies averaged over all molecules and over long times: (i) the translational energy of the molecular centres of mass, (ii) the rotational energy, and (iii) kinetic energy of the atoms considered individually. How do the values compare? Are they consistent with the equipartition theorem?