High-performance Computing

Coursework Assignment

Deadline: 24th March 2021 - 23:00

Instructions

Please take note of the following when completing this assignment:

- Read all the tasks carefully and plan ahead before you start designing and implementing your code.
- You may use any of the tools and libraries available on the provided Linux environment.
- Your submitted code **must** compile and run correctly on the provided Linux environment.

Make regular backups of your code onto a separate computer system; no allowance will be made for data loss resulting from human or computer error.

1 Introduction

The objective of this coursework is to write a parallel numerical code for solving a two-dimensional smoothed particle hydrodynamic (SPH) formulation of the Navier-Stokes equations. SPH models the behaviour of fluids by approximating the fluid properties using smooth kernel density functions. Each particle therefore only influences neighbouring particles within a given radius and properties of the fluid are smoothed between particles.

The position and velocity of each particle are updated at each time step using an explicit time-integration scheme. We calculate these state variables by updating our approximation to the fluid density, pressure and viscous forces on each particle and use these to calculate the acceleration of the particles.

The equations for SPH are derived from the Navier-Stokes equations. Primarily we need to enforce conservation of linear momentum. The derivation of these equations can be found in textbooks.

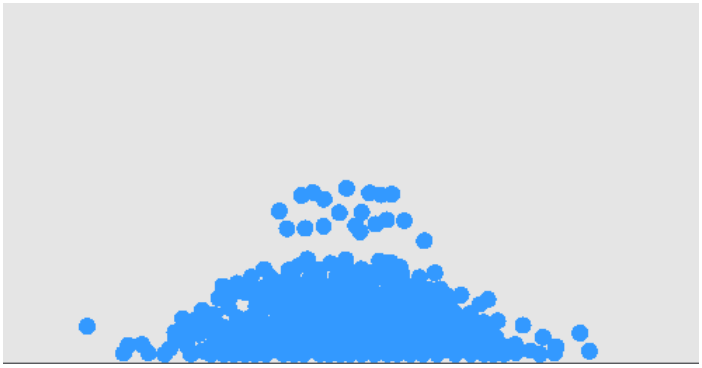

Figure 1: Illustrative SPH simulation of a droplet, represented by a collection of particles, hitting a surface.

2 Algorithm

The simulation consists of a set, P, of N particles, with positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$. Each particle has a mass m and a radius of influence h, beyond which no interaction with other particles is considered. The computational domain is a box $\Omega = [0, 1]^2$. The parameters for the problem are:

Constant	Symbol	Value
Gas constant	k	2000.0
Resting density	ρ_0	1000.0
Viscosity	μ	1.0
Acceleration due to gravity	g	9.81
Radius of influence	h	0.01
Coefficient of restitution	e	0.5

For a single time-step the algorithm proceeds as follows:

Calculate density: We approximate the density of the fluid associated with each particle as

$$\rho_i = \sum_j m\, \phi_d(\mathbf{r}_{ij}, h), \tag{1}$$

where $\mathbf{r}_{ij} = \mathbf{x}_i - \mathbf{x}_j$. Let the distance between particles, normalised by the interaction radius, be $q = \frac{||\mathbf{r}_{ij}||}{h}$, then the kernel density function for density, $\phi_d(\mathbf{r}_i, h)$, is given by

$$\phi_d(\mathbf{r}_{ij}, h) = \frac{4}{\pi h^2} \begin{cases} (1-q^2)^3 & \text{if } q < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Calculate pressure: To calculate the pressure, we use the ideal gas law

$$p_i = k(\rho_i - \rho_0), \tag{3}$$

where ρ_0 is a *resting density* and k is a gas constant.

Calculate pressure force: We calculate the force exerted on the particle due to pressure from neighbouring fluid particles

$$\mathbf{F}_i^p = -\sum_j \frac{m}{\rho_j} \frac{(p_i + p_j)}{2} \nabla\phi_p(\mathbf{r}_{ij}, h), \tag{4}$$

where

$$\nabla\phi_p(\mathbf{r}_{ij}, h) = -\frac{30}{\pi h^3} \mathbf{r}_{ij} \begin{cases} \frac{(1-q)^2}{q} & \text{if } q < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Calculate viscous force: Next we calculate the force acting on each particle due to viscous effects.

$$\mathbf{F}_i^v = -\mu \sum_j \frac{m}{\rho_j} \mathbf{v}_{ij}\, \nabla^2\phi_v(\mathbf{r}_i, h), \tag{6}$$

where $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ and

$$\nabla^2\phi_v(\mathbf{r}_{ij}, h) = \frac{40}{\pi h^4} \begin{cases} (1-q) & \text{if } q < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Calculate gravity force: Finally, the force due to gravity can be calculated as

$$\mathbf{F}_i^g = (0,\, -\rho_i g). \tag{8}$$

Time integration: Advancing the particles in time is performed explicitly. While a simple Forward Euler scheme could be implemented, a leap-frog scheme provides improved stability characteristics. Here we use the superscript in brackets to denote the time-level.

$$\mathbf{a}_i = \frac{\mathbf{F}_i^p + \mathbf{F}_i^v + \mathbf{F}_i^g}{\rho_i}, \tag{9}$$

$$\mathbf{v}_i^{(t+\frac{1}{2})} = \mathbf{v}_i^{(t-\frac{1}{2})} + \mathbf{a}_i \Delta t, \tag{10}$$

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+\frac{1}{2})} \Delta t. \tag{11}$$

However, because the velocity is calculated at half-steps, we need to initialise the scheme on the first time step by instead doing:

$$\mathbf{a}_i = \frac{\mathbf{F}_i^p + \mathbf{F}_i^v + \mathbf{F}_i^g}{\rho_i}, \tag{12}$$

$$\mathbf{v}_i^{(\frac{1}{2})} = \mathbf{v}_i^{(0)} + \mathbf{a}_i \Delta t/2, \tag{13}$$

$$\mathbf{x}_i^{(1)} = \mathbf{x}_i^{(0)} + \mathbf{v}_i^{(\frac{1}{2})} \Delta t. \tag{14}$$

To ensure convergence in time a time-step of $\Delta t = 10^{-4}$ is suggested.

2.1 Initial Condition

To initialise the simulation, one or more particles must be specified. These should be evenly distributed. Some examples are suggested in the test cases.

Once the particles are placed, the particle densities should be evaluated with an assumed mass of $m = 1$ and the mass subsequently scaled so that the density is equal to the reference density:

$$m = \frac{N\rho_0}{\sum_i \rho_i}. \tag{15}$$

2.2 Boundary Conditions

All the boundaries are solid walls with damped reflecting boundary conditions. If particles are within a distance h of the boundary, their velocity and position should be updated to reverse the motion of the particle and keep it within the domain. For example, on the right boundary, the x-component of position and velocity would be modified as

$$x^* = 1 - h, \qquad u^* = -eu. \tag{16}$$

2.3 Energy

The motion of particles within the interior of the domain should be energy conserving. As particles strike the boundaries, their motion is damped and therefore lead to a loss of energy from the system.

The kinetic energy and potential energy of the system can be calculated as:

$$E_k = \frac{1}{2} m \sum_i ||\mathbf{v}_i||^2, \tag{17}$$

$$E_p = mg \sum_i y_i. \tag{18}$$

3 Code Development

3.1 Validation cases

When developing your code, the following test cases may be helpful to assess the correctness of your code:

- A single particle at (0.5, 0.5) to test the correctness of time integration and gravity forcing, as well as the bottom boundary condition.
- Two particles at (0.5, 0.5) and (0.5, h) to assess the pressure force and viscous terms. Try changing μ and k.
- Three particles at (0.5, 0.5), (0.495, h) and (0.505, h) to assess left and right boundary conditions. Try changing e.
- Four particles at: (0.505, 0.5), (0.515, 0.5), (0.51, 0.45) and (0.5, 0.45) to assess multiple particle interaction.

3.2 Test cases

These are more interesting initial conditions and are useful for assessing the performance of your code:

- Dam break: a grid of particles occupying the region $[0, 0.2]^2$.
- Block drop: a grid of particles occupying the region $[0.1, 0.3] \times [0.3, 0.6]$
- Droplet: particles occupying a circle of radius 0.1, centred at the point $[0.5, 0.7]$.

You may need to add small amplitude noise to the particle positions to see more realistic behaviour.

3.3 Command-line input

Your code should accept the following parameters as command-line arguments (do **not** prompt the user to enter input during the execution of your program):

```
--ic-dam-break      Use dam-break initial condition.
--ic-block-drop     Use block-drop initial condition.
--ic-droplet        Use droplet initial condition.
--ic-one-particle   Use one particle validation case initial condition.
--ic-two-particles  Use two particles validation case initial condition.
--ic-four-particles Use four particles validation case initial condition.
--dt arg            Time-step to use.
--T arg             Total integration time.
--h arg             Radius of influence of each particle.
```

3.4 File output

Your code should generate a file called `output.txt` with the position of all particles at the final time, specified as two columns (x- and y-coordinates, separated by space). You may output this file at intermediate times (e.g. every second of simulation time) if you wish.

Your code should also generate a file `energy.txt` with four columns of data: the time, kinetic energy, potential energy and total energy at each time step.

3.5 Optimisation and Parallelisation

You should undertake some basic performance optimisation of your implementation and parallelise your code using MPI. There are several approaches to parallelisation for this problem. You should discuss in the report the approach you have chosen and why.

Tasks

The objective of this coursework is to write a high-performance parallel, object-oriented, C++ code which will solve the SPH problem as described above. In completing this assignment you should write a **single** code which satisfies the following requirements:

- Reads the parameters of the problem **from the command line** (do not prompt the user for input). Please use the **exact** syntax given for the parameters in section 3.3 as these will be used when testing your code. [5%]
- Implements a class called `SPH` to solve the SPH problem using the algorithm described above. You are free to add any additional functions and classes, as required. [30%]
- Generates an appropriate initial set of particles based on the command line arguments given. [5%]
- Is parallelised using MPI. It should be able to run on at least two processes (as specified by the `-np` parameter to `mpiexec`). [30%]
- Uses a build system (Make or CMake) to compile your code and uses git for version control. Provide evidence of the latter by running the command

  ```
  git log --name-status > repository.log
  ```

  and including the file `repository.log` in your submission. [5%]
- Uses good coding practices (code layout, comments, documentation) [5%]
- Is accompanied by a report (max 3 pages) as detailed below. [20%]

Report (3 pages)

Write a short and concise report which includes:

- plots of the potential energy, kinetic energy and total energy of the system (all on the same plot) as a function of time, for the three test cases in section 3.2 (three plots in total) until they reach steady state ;
- a discussion of the serial performance of your code, highlighting the parts of your implementation which take the longest and any code optimisations you have applied to improve the execution speed;
- a discussion of how you chose to parallelise your code (and why) as well as any design decisions you explicitly made to enable this.

Submission and Assessment

When submitting your assignment, make sure you include the following:

- All the files needed to compile and run your C++ code:
  - Source files for a single C++ program which performs all the requirements. i.e. All `.cpp` and `.h` files necessary to compile and run the code.
  - The `Makefile` or `CMakeLists.txt` file (as preferred) used for compiling your code and linking with any necessary libraries.
- Your three-page report (in PDF format only).
- The git log (`repository.log`).

These files should be submitted in a **single `ZIP` archive file** to Blackboard Learn.

It is your responsibility to ensure all necessary files are submitted.

You may make unlimited submissions and the last submission before the deadline will be assessed.

END OF ASSIGNMENT