Université de Liège

MATH0471-A-a : Multiphysics integrated computational project

Deadline 2 : OpenMP implementation of 3D SPH and Explicit Euler scheme.

KOLLASCH Killian s202793
SANTORO Luca s201807

Professor: C. Geuzaine
Professor: R. BOMAN

Academic year : 2023-2024

I Implementation of 3D SPH for Navier-Stockes

Several crucial functions are implemented thanks to the Navier-Stokes equations: the continuity equation, which describes the variation of density over time, and the momentum equation, which accounts for the variation of velocity. These equations use a function called artificial viscosity. Both equations require the use of the function gradW, which computes the gradient of the chosen kernel function. To determine the neighbouring particles of each particle, the function employs the linked-list algorithm.

continuityEquation

The continuity equation is based on the following formula:

$$\frac{D\rho_a}{Dt} = \sum_{b=1}^{N} m_b \vec{u_{ab}} \cdot \vec{\nabla} W_{ab}, \tag{I.1}$$

where indices a and b refer to the particle a and its neighbours b, and N denotes the total number of neighbours of a. Here, $\vec{u_{ab}}$ represents the relative velocity between a and b, and $\vec{\nabla} W_{ab}$ denotes the gradient of the kernel function W, computed by the function "gradW".

Remark: cubic spline is used for the following simulations.

momentumEquation

The momentum Equation is given by:

$$\frac{D\vec{u_a}}{Dt} = -\sum_{n=1}^{N} m_b \left(\frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} + \Pi_{ab} \right) \vec{\nabla} W_{ab} + \vec{F}, \tag{I.2}$$

where p represents pressure, which is determined by the nature of the fluid, whether it is a perfect gas or a quasi-incompressible fluid. $\vec{F}$ typically represents a body force, such as gravity. The term Π_{ab} stands for artificial viscosity, which is calculated using the formula:

$$\Pi_{ab} = \begin{cases} \frac{\alpha \; c_{ab} \;\; \bar{\mu_{ab}} + \beta \;\; \mu_{ab}^2}{\bar{\rho_{ab}}} & \text{for } \vec{u_{ab}} \cdot (\vec{x_a} - \vec{x_b}) < 0 \\ 0 & \text{for } \vec{u_{ab}} \cdot (\vec{x_a} - \vec{x_b}) \geq 0. \end{cases} \tag{I.3}$$

with $\bar{\rho_{ab}}$ and $\bar{c_{ab}}$ are the mean density en speed of sound between particle a and b. α and β are parameters given by the user. And μ_{ab} has as formula:

$$\mu_{ab} = \frac{h\vec{u_{ab}} \cdot (\vec{x_a} - \vec{x_b})}{(\vec{x_a} - \vec{x_b})^2 + \eta^2} \tag{I.4}$$

With $\eta^2 = 0.01 \; h^2$. This artificial viscosity is used in order to reproduce friction and to avoid some numerical instabilities when particles are moving away from each other as explained in L.Goffin master's thesis.

gradW

The last important function, up to this point, is gradW which gives the gradient used in continuityEquation and momentumEquation. Its equation is:

$$\vec{\nabla} W_{ab} = \frac{\vec{x_a} - \vec{x_b}}{r_{ab}} \frac{dW}{dr}|_{r=r_{ab}}. \tag{I.5}$$

Remark: all these functions are implemented in the cpp file "gradient.ccp".

The results obtain by continuity and momentum are used for the Euler explicit scheme. The density, the velocity and the position are updated as follow:

$$\rho_{n+1} = \rho_n + \Delta t \frac{D\rho_n}{Dt}, \tag{I.6}$$

$$\vec{x_{n+1}} = \vec{x_n} + \Delta t \ \vec{u_n}, \tag{I.7}$$

$$\vec{u_{n+1}} = \vec{u_n} + \Delta t \frac{D\vec{u_n}}{Dt}. \tag{I.8}$$

Where Δt is given by the user up to now, but this will be changed in order to incorporate the condition giving in the master's thesis.

II Simulations of the falling water cube

The parameters used for the falling cube simulation are the same as the one used by L.Goffin master's thesis : $\kappa = 2, \rho_0 = \rho_{MP} = \rho_{FP} = 1000 kg/m^3, T = 298.15K, c_0 = 30m/s, \gamma = 7, \alpha = 0.5, \beta = 0, M = 18.10^{-3} kg/mol$. Also, we chose a timestep $\Delta t = 5.10^{-5} s$:

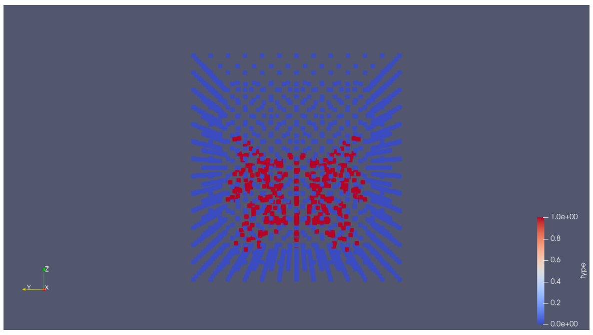

Figure 1: Side view ($\Pi_{ab} = 0$), iteration 72/199.

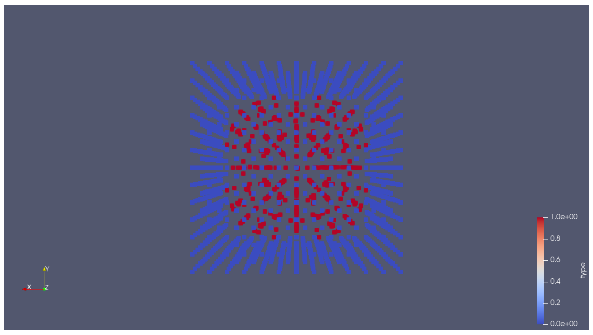

Figure 2: Top view ($\Pi_{ab} = 0$), iteration 72/199.

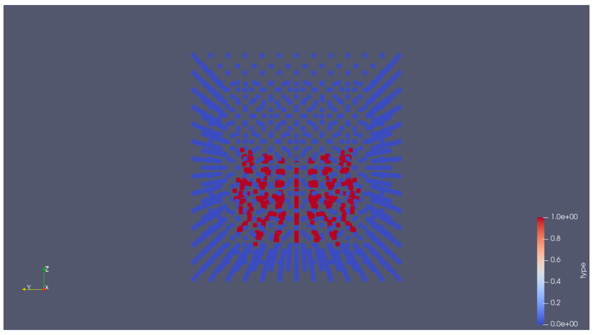

Figure 3: Side view ($\Pi_{ab} \neq 0$), iteration 72/199.

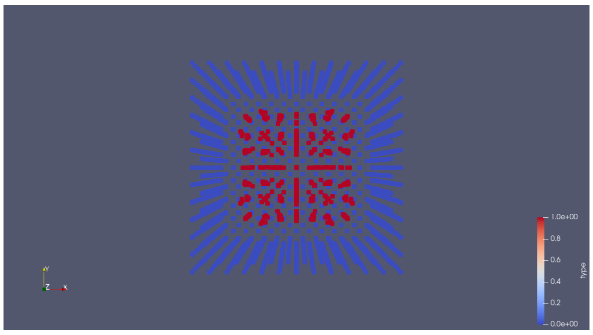

Figure 4: Top view ($\Pi_{ab} \neq 0$), iteration 72/199.

Figure 5: Simulation of a falling water cube (particles in red) in a box (particles in blue).

Figure 5 depicts two simulations of a falling cube. In the two first simulations, artificial viscosity remains consistently zero as the function is not used. Initially, the solution appears symmetric. However, after several time steps following the collision with the bottom of the box, some particles pass through, indicating an issue that needs resolution.

In the two last simulations, artificial viscosity is employed. Despite this addition, some problems persists. When fluid particles make contact with the bottom, they experience repulsion.

However, they stabilise at "equilibrium points," ceasing to move for the remainder of the simulation. This final state is evidently non-physical, presenting another issues that has to be solved.

At present, it remains unclear whether these issues are interconnected or distinct cases. Regardless, resolving them is imperative to obtain physically plausible solutions and to facilitate the simulation of more complex problems. The equation of state *quasi incompressible fluid* leads to slightly better results (smaller oscillations of the particles and lesser moving particles going through the boundaries) but does not seem to be accurate.

Remark: on a single plate (with no extra walls), a simulation (with $\Pi_{ab} = 0$) seems to be completely symmetric but some moving particles keep going through the boundaries while simulation with (with $\Pi_{ab} \neq 0$) leads to a very asymmetric solution.

III Comparison with OpenMP

Because many loops iterate over all particles, parallelisation appears to be a promising approach to enhance speed. In theory, almost all functions that involve loops iterating over particles could be parallelised. So far, we've only use OpenMP for the continuity and momentum equations, as we don't use push_back() or encounter any issues with parallelisation. However, in other loops, such as those in sorted_list, we could include pragma directives to indicate parallelisation. For instance:

```
            #pragma parallel critical.
```

However, we have not tried this possibility yet.

Spacing s	0.3	0.2	0.1
number of part	168	393	1808
Serial [s]	52.7	148.4	667.9
With OpenMP [s]	71.7	147.0	494.2

Table 1: Difference between the time needed between serial and OpenMP.

Table 1 demonstrates the impact of OpenMP on reducing the runtime of the code, particularly noticeable with a large number of particles. While the serial implementation may appear more efficient for smaller particle counts, it's important to note that simulations often require a substantial number of particles to accurately model physical phenomena.

IV What is next ?

Many steps have to be taken to ensure a good simulation and to improve the speed of the code for later use when more particles will be involved. A few steps are described below:

- Correction of the code to prevent moving particles from passing through the box.
- Correction of artificial viscosity if this problem is not linked with the previous one.
- Improving the OpenMP implementation to make the code faster by parallelising the loop containing "pushback()" instruction.
- Implementing Runge Kutta scheme and making a comparison with the Euler one.
- Using profilers to analyse the time spent in each routine and trying to optimise them.

- Reviewing literature to understand how to handle viscosity, surface tension, and adhesion.
- Conducting simulations on microfluidics applications.