Numerical modeling of fluid behavior using Smoothed Particle Hydrodynamics method

1st Olena Azarova
Research and Development
Kapitoshki
Lviv, Ukraine
azarova.pn@ucu.edu.ua

2nd Anna Monastyrska
Research and Development
Kapitoshki
Lviv, Ukraine
monastyrska.pn@ucu.edu.ua

3rd Oles Yarish
Research and Development
Kapitoshki
Lviv, Ukraine
yarish.pn@ucu.edu.ua

4th Elizaveta Arnauta
Research and Development
Kapitoshki
Lviv, Ukraine
arnauta.pn@ucu.edu.ua

5th Mykhailo Moroz
Research and Development
Zibra
Lviv, Ukraine
michael08840884@gmail.com

***Abstract*—This document concludes our work on the Numerical modelling of fluid behaviour using the Smoothed Particle Hydrodynamics (SPH) method. In this project, we discuss the theory of SPH and its implementation to simulate fluid behaviour. and its implementation.**

***Index Terms*—smoothed particle hydrodynamics, computational fluid dynamics, numerical analysis, Fast fixed-radius nearest neighbours, kernel hydrodynamics**

I. Introduction

The aim of this project is to simulate the fluid behavior. To describe fluids mathematically, the Navier-Stokes equations are commonly used. This is a nonlinear system of differential equations that describes the flow of a fluid whose stress depends linearly on flow velocity gradients and pressure. Without simplifications, these equations have no analytical solutions, so they are primarily of use in computational fluid dynamics [5]. However, these equations can be simplified by using different approximations. In our project, we will assume that the fluid is incompressible. We will also assume that the fluid is a set of individual particles, each with its own properties such as mass, position, velocity, and density. Thus, the main task for modeling fluid behavior will be to calculate the acceleration for each particle and update its position over time. We will use the Smoothed-particle hydrodynamics (SPH) method for modeling. SPH is a particle-based, mesh-free Lagrangian method (in this method coordinates move with the fluid). This method has also been used in many fields of research, including astrophysics, ballistics, volcanology, and oceanography.

II. Possible approaches

A. Methods of fluid modelling

1. Stable Fluids

This approach enables the simulation of fluid behaviors with numerical stability and efficiency. The method employs discretization of the Navier-Stokes equations, which govern fluid motion, into a grid-based framework. Key components include the representation of velocity and pressure fields, along with semi-Lagrangian advection for fluid quantity transport. Notably, the term ”stable” underscores its robustness against numerical instabilities, ensuring accurate results even in the presence of small perturbations or computational errors. Implicit time integration techniques further enhance stability, particularly for challenging scenarios. However, this method has notable limitations. One such drawback is its computational cost, especially for high-resolution simulations or complex fluid behaviors. Additionally, the method struggles with accurately capturing turbulent flows, which are prevalent in many real-world scenarios. These limitations necessitate careful consideration of simulation parameters and trade-offs in practical applications, which is not really good for our project.

2. Lattice Boltzmann Method (LBM)

The Boltzmann lattice method (LBM) is based on the fact that instead of directly solving the Navier-Stokes equations, the LBM works at the mesoscopic level, using the lattice structure to model fluid flow. In general, LBM simulates the motion and collisions of fictitious particles in a lattice grid. Through iterative collision and propagation steps, fluid properties such as density and velocity change over time, allowing for the understanding of complex fluid phenomena. One of the “pros” of LBM is its ability to handle complex geometries and boundary conditions with relative ease. In addition, LBM naturally accounts for multiphase and multiphysics phenomena, making it suitable for a wide range of applications. But there some significant disadvantages that can greatly complicate the simulation work: the method can require significant computations, especially for high-resolution simulations or scenarios involving turbulent flows. To ensure accurate results, careful selection of lattice structures and collision models is required, which adds to the complexity of

the implementation.

3. Smoothed-particle hydrodynamics (SPH)

Unlike grid-based approaches, SPH represents the fluid as a collection of particles that interact with each other based on a smoothed kernel function. In SPH, each particle carries properties such as mass, velocity, and density. Interactions between particles are determined by evaluating the kernel function, which quantifies the influence of neighboring particles on each particle's properties. This allows SPH to accurately model complex fluid behaviors, including free surfaces, fluid mixing, and fluid-solid interactions. One of SPH's notable "pros" is its ability to handle dynamic and deformable boundaries without requiring complex mesh generation. Additionally, SPH naturally accommodates simulations with irregular geometries and moving boundaries, making it well-suited for applications in astrophysics, engineering, and computer graphics. However, SPH also has its limitations. One challenge is accurately resolving discontinuities and sharp gradients in fluid properties, which can lead to numerical instabilities or inaccuracies, particularly in high-speed or turbulent flows.

Conclusion

After weighing the pros and cons, smoothed particle hydrodynamics (SPH) seems to be the best choice for fluid modeling. Despite the problem swith numerical stability and computational requirements, SPH gives us unprecedented flexibility in handling complex scenarios with dynamic boundaries and deformable bodies. Its ability to accurately model fluid-solid interactions and irregular geometries outweighs its limitations, making it a very good choice for our project.

B. Methods of integration

The choice of a time integration scheme is an important aspect of any transient fluid simulation. Several possible methods of time integration can be used, such as Euler integration Runge-Kutta, and Leap Frog.

The simplest is the Euler integration scheme, which updates the state of each particle at each time step based on its current velocity and acceleration. This integration can suffer from numerical instability in simulations where forces change very fast or the behavior of particles is complicated (highly non-linear).

Runge-Kutta methods use weighted averages of multiple function evaluations to approximate the solution at each time step. These methods come in various orders, with higher-order methods offering better accuracy, however, they require more computational resources. Thus Runge-Kutta integration is used in some SPH simulations, only when accuracy is critical.

For our simulation, we chose Leap Frog integration [6], which is quite popular in such problems because it offers better accuracy and stability, than Euler's and also requires less computational resources, than Runge-Kutta. Leap Frog is a second-order accurate scheme that calculates the positions and velocities of particles at interleaved time points. Here are the formulas for Leap Frog integration:

$$v_{i+1/2} = v_{i-1/2} + a_i * \Delta t,$$
$$r_{i+1} = r_i + v_{i+1/2} * \Delta t,$$

Where v_i is velocity of a particle at i^{th} step, Δt is time step, and v_i is a position of a particle at i^{th} step.

III. Theory of the SPH method

A. Main idea

Calculations of most forces at play in SPH framework are done by sticking to the general formulas provided below.

To find the value of the scalar field $F(\mathbf{r})$ at any point, the following formula is used:

$$F(\mathbf{r}) = \sum_j F_j V_j W(\mathbf{r_i} - \mathbf{h_j}, h)$$

where the subscript j iterates over all particles, $\mathbf{h}_j$ is the current position of particle j, V_j is the volume of the particle and r is the support radius of the Kernel Function, see below.

The gradient of the scalar field is given by:

$$\nabla F(\mathbf{h}) = \sum_j F_j V_j \nabla W(\mathbf{r_i} - \mathbf{r_j}_j, h)$$

And the Laplacian is:

$$\nabla^2 F(\mathbf{h}) = \sum_j F_j V_j \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h)$$

The main idea of this project is to simulate fluid behaviour in the 2-dimensional space using SPH method. This method works by dividing the fluid into a set of discrete moving elements, referred to as particles. Each particle has such parameters: mass, position, velocity, density, viscosity and pressure. Firstly we have to set the mass of our particles, which will remain unchanged till the end of the simulation. Then we calculate and update their parameters based on the conditions at the moment using smoothing kernels.

B. Smoothing Kernels

As previously discussed, the SPH method represents a fluid as a collection of particles with certain parameters. To compute these parameters we use smoothing kernels.

A kernel smoother is a statistical method used to estimate a continuous function based on observed data. It works by calculating a weighted average of neighbouring points, where the weights are determined by a mathematical function – the kernel (also called smoothing kernel, kernel function). Points that are closer to the point of interest have higher weights in the calculation. Figure 2 is a schematic illustration of the smoothing kernel function. The resulting estimated function is smooth, and the degree of smoothness can be adjusted by a single parameter. The algorithm requires three different kernels:

- The Polynomial Kernel
- The Spiky Kernel

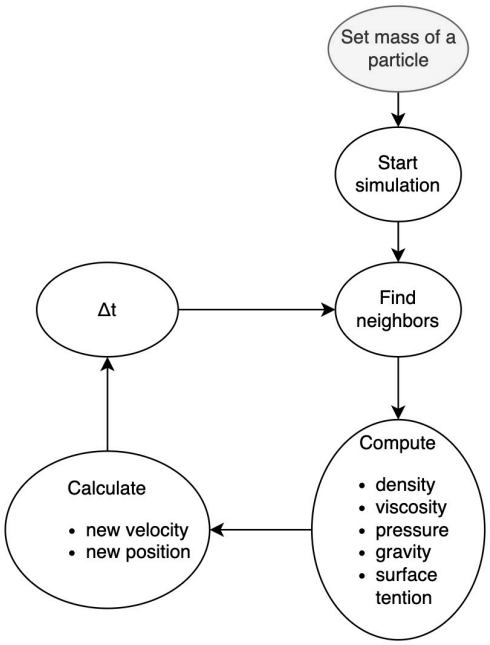

Fig. 1. Finite State Machine of the SPH algorithm

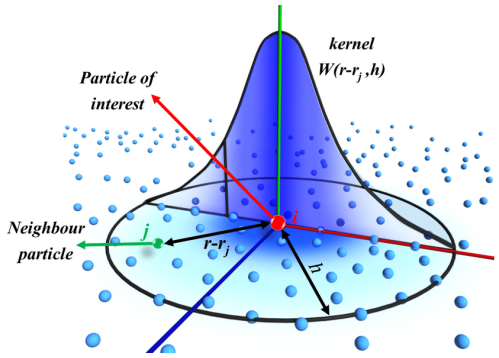

Fig. 2. Schematic illustration of a SPH smoothing kernel function

- The Viscosity Kernel

Each kernel has a specific mathematical formulation that makes it suitable for calculating a particular type of force or interaction between particles in the SPH simulation.

1) The Polynomial Kernel: The polynomial kernel smoothly distributes quantities over space making it perfect to calculate density and surface tension force. This formula is the default kernel formula suggested by [4].

The Polynomial Kernel:

$$W_{\text{poly}}(\vec{r},h) = A\begin{cases}(h^2-\|\vec{r}\|^2)^3 & 0\le\|\vec{r}\|\le h\\ 0 & \|\vec{r}\|>h\end{cases}$$

$$A = \frac{315}{64\pi h^9} \quad \text{for 3D}$$

$$A = \frac{4}{\pi h^8} \quad \text{for 2D}$$

The Polynomial Gradient:

$$\nabla W_{\text{poly}} = -B\vec{r}(h^2-\|\vec{r}\|^2)^2$$

$$B = \frac{945}{32\pi h^9} \quad \text{for 3D}$$

$$B = \frac{24}{\pi h^8} \quad \text{for 2D}$$

The Polynomial Laplacian:

$$\nabla^2 W_{\text{poly}} = -C\cdot(h^2-\|\vec{r}\|^2)(3h^2-7\|\vec{r}\|^2)$$

$$C = \frac{945}{32\pi h^9} \quad \text{for 3D}$$

$$C = \frac{24}{\pi h^8} \quad \text{for 2D}$$

2) The Spiky Kernel: We will use the spiky kernel to calculate the pressure force. It is needed to keep particles from bunching up too much.

The Spiky Kernel:

$$W_{\text{spiky}}(\vec{r},h) = A\begin{cases}(h-\|\vec{r}\|)^3 & 0\le\|\vec{r}\|\le h\\ 0 & \|\vec{r}\|>h\end{cases}$$

$$A = \frac{15}{\pi h^6} \quad \text{for 3D}$$

$$A = \frac{10}{\pi h^5} \quad \text{for 2D}$$

The Spiky Gradient:

$$\nabla W_{\text{spiky}} = -B\cdot\frac{\vec{r}}{\|\vec{r}\|}(h-\|\vec{r}\|)^2$$

$$B = \frac{45}{\pi h^6} \quad \text{for 3D}$$

$$B = \frac{30}{\pi h^5} \quad \text{for 2D}$$

The Spiky Laplacian:

$$\nabla^2 W_{\text{spiky}} = -C\cdot(h-\|\vec{r}\|)(h-2\|\vec{r}\|)/\|\vec{r}\|$$

$$C = \frac{90}{\pi h^6} \quad \text{for 3D}$$

$$C = \frac{60}{\pi h^5} \quad \text{for 2D}$$

3) The Viscosity Kernel: The viscosity kernel simulates the internal frictional forces making it perfect for calculating the viscous force.

The Viscosity Kernel:

$$W_{\text{visc}}(\vec{h}, r) = A \begin{cases} -\frac{\|\vec{r}\|^3}{2h^3} + \frac{\|\vec{r}\|^2}{h^2} + \frac{h}{2\|\vec{r}\|} - 1 & 0 \leq \|\vec{r}\| \leq h \\ 0 & \|\vec{r}\| > h \end{cases}$$

$$A = \frac{15}{2\pi h^3} \quad \text{for 3D}$$

$$A = \frac{10}{3\pi h^2} \quad \text{for 2D}$$

The Viscosity Gradient:

$$\nabla W_{\text{visc}} = -B\vec{r}\left(-\frac{3\|\vec{r}\|}{2h^3} + \frac{2}{h^2} - \frac{h}{2\|\vec{r}\|^3}\right)$$

$$B = \frac{15}{2\pi h^3} \quad \text{for 3D}$$

$$B = \frac{10}{\pi h^2} \quad \text{for 2D}$$

The Viscosity Laplacian:

$$\nabla^2 W_{\text{visc}} = -C \cdot (h - \|\vec{r}\|)$$

$$C = \frac{45}{2\pi h^6} \quad \text{for 3D}$$

$$C = \frac{20}{\pi h^5} \quad \text{for 2D}$$

C. Forces in play

So as to bring the simulation to life, the forces acting between particles must be accurately represented in mathematical notions and then implemented in code. In particular, the forces featured in our simulation are described by the following formulae:

1) Gravity: The formula of gravity is quite straightforward:

$$f_i = \rho_i \cdot \vec{g}$$

where $\vec{g}$ is the gravitational acceleration, and ρ_i represents the density of particle i, expressing the mass per unit volume of the particle or the medium in which it resides.

2) Surface Tension: The formula of surface tension, on the contrary, is quite complicated:

$$f_i = -\sigma * \nabla^2 c_s * \frac{n}{|n|}$$

where $\frac{-\nabla^2 c_s}{|n|}$ is divergence of the surface normal that gives the curvature of the surface. σ — the coefficient of surface tension, which characterizes the energy required to increase the interface area between two phases. $\nabla^2 c_s$ — the Laplacian of the color field c_s, indicating the change in intensity of the color field in space, corresponding to changes in the curvature of the surface. In its turn, n is the gradient of the color field c_s that is given by:

$$c_s(\vec{h}) = \sum_j \frac{m_j}{\rho_j} * W_{poly}(\vec{r_i} - \vec{r_j}, h)$$

3) Viscosity: The viscosity formula is given by:

$$f_i = -\alpha * \sum_j * m_j * \frac{\vec{v_{ij}} \cdot \vec{r_{ij}}}{|\vec{r_{ij}}|} * \nabla W_{visc}(\vec{r_i} - \vec{r_j}, h)$$

where α is the viscosity coefficient.

4) Pressure: The pressure is calculated in the following way:

$$f_i = B[(\frac{\rho}{\rho_0})^\gamma - 1]$$

which is known as *Cole equation state*. However, in our implementation the generalized version is used:

$$f_i = B\left(\frac{p - p_0}{\rho_0}\right)$$

where γ is taken to be zero and $\frac{B}{\rho_0}$ can be considered as a single pressure coefficient. Also, the pressure is later multiplied by appropriate kernel value.

5) Density: Density, that is used in other formulas, is obtained with the following:

$$\rho_i = \sum_j m_j * W_{poly}(\vec{r_i} - \vec{r_j}, h)$$

IV. Optimization

The idea mentioned in previous sections is great for small simulations. One problem that will arise when running a more massive simulation with a greater number of particles is the time needed to update the states of the particles in play.

In the current implementation, we iterate through all particles to assess interactions for just one particle at a time. This results in $n(n-1)$ tests conducted each simulation frame. This method is inefficient because particles mainly interact with those that are nearest to them. Therefore, we can eliminate unnecessary tests by ignoring particles that are too far to significantly influence the particle in question.

A. New idea

A lot of the efficient implementations use a two-phase approach: a broad phase followed by a narrow phase. The output of the broad phase is a set of pairs of objects that potentially interact. This phase aims to quickly identify and eliminate pairs of objects that are definitely not colliding, allowing the more computationally intensive narrow phase to focus only on a small number of such pairs.

The main tasks of the narrow phase involve accurately determining if these pairs actually interact and, if so, calculating the specifics of that interaction. This phase has been discussed in earlier sections.

Such an approach is very suitable in our case when a lot of the objects do not interact, rejecting many pairs of such objects during the broad phase.

B. The broad phase

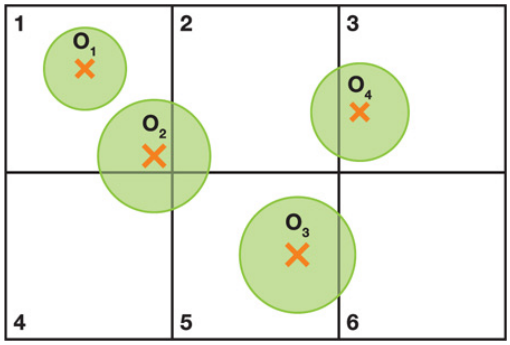

Fig. 3. An Example of 2D Spatial Subdivision of Four Particles

1) Spatial subdivision: Spatial subdivision partitions space into a grid, such that a grid cell is at least as large as the largest object (in our case, all particles are equal). Then we assign to each cell the "list" of all particles whose centers are within that cell.

For example, in Figure3 Cell 1 includes particles O1, O2; Cell 2 - none; Cell 3 - particle O4; Cell 4 - none; Cell 5 - particle O3; Cell 6 - none.

2) Basic conditions for conducting a test: A test between two objects is performed only if:

- they both appear in the same cell
- at least one of them has its centroid in this cell.

For example, in Figure 3, a test is performed between objects O1 and O2 because both have their centroids in cell 1; same with O2 and O3. After all, both appear in cell O5, and O3 has its centroid in cell 5. But no test is performed between O2 and O4 because although both appear in cell 2, neither of their centroids is in cell 2.

3) Sorting algorithms: Since One possible sorting algorithm is radix sort. Radix sort sorts by the 32-bit cell IDs. An important fact about the Radix sort is that it sorts the elements by their digits (or tuples of bits) from the least to most significant, so it takes multiple passes to complete.

As you can see from Figure 4 we have a set of particles with IDs and corresponding cells. The first step is to find the largest cell ID in the array and determine how many passes there will be (70 in our case, which means that there will be 2 passes). Then, we iteratively sort the pairs by the current digit by first calculating the number of particles in a particular cell, then the number of particles in cells preceding the each given cell ("prefix sum"), thus obtaining an offset for its particles, and finally putting the pairs from input buffer to indices in the output buffer based on the prefix sum of the corresponding cell.

In Figure 5 we have an example of counting sort. It works in a way similar to that of radix sort, but takes only one pass to complete, since instead of radices it uses entire cell IDs, thus eliminating the need to guarantee stability between passes.

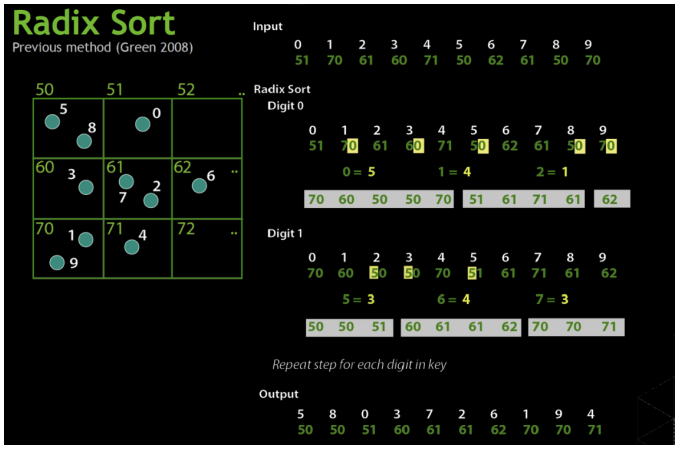

Fig. 4. Example of Radix Sort from [7]

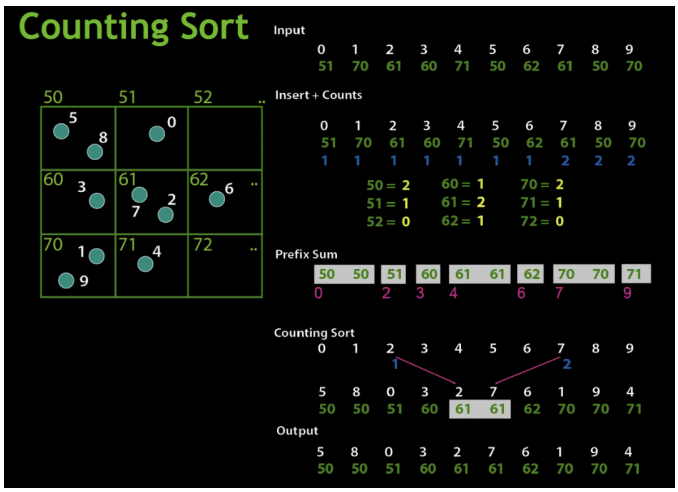

Fig. 5. Example of Counting Sort from [7]

4) Implementation: Eventually we decided to employ counting sort to sort particles, since radix sort's stability is rather difficult to guarantee in SIMT context. Also, our implementation uses five buffers: one that holds pairs of type $\langle cellID; particleID \rangle$, one that stores the number of particles in a cell, one that stores prefix sums of the cells, one where the the pairs ordered by cell ID are stored and one that specifies where in the output array the pairs of a given cell begin.

V. Results

You can see the video of our simulation in the drive. The simulation involved 4800 particles. The link to Google Drive

VI. Conclusion

In this project, we managed to model fluid behaviour with the Smoothed Particle Hydrodynamics method by untangling the theory behind it. During our work we leveraged Euler method for integration. However, while working on optimization we had perplexities with implementing the sorting algorithm. Firstly we tried to use Radix sort, but it was hard to guarantee its stability in parallel environment. So we

implemented Counting sort instead. We also got more familiar with Unity Game Engine, principles of GPU computations and shader programming.

REFERENCES

[1] Adithya Vijaykumar. *Smoothed Particle Hydrodynamics Simulation for Continuous Casting*. Master's Thesis in Scientific Computing, Master Programme in Scientific Computing, Royal Institute of Technology, 2012. Supervisor: Jesper Oppelstrup. Examiner: Michael Hanke. TRITA-MAT-E 2012:11. ISRN-KTH/MAT/E–12/11–SE. Royal Institute of Technology, School of Engineering Sciences, KTH SCI, SE-100 44 Stockholm, Sweden.

[2] Philip Mocz. *Smoothed Particle Hydrodynamics: Theory, Implementation, and Application to Toy Stars*. Mon. Not. R. Astron. Soc. **000**, 1–9 (2011). Printed 13 December 2011 (MN LATEX style file v2.2). Applied Math 205 Final Project, Harvard University, Fall 2011. Prof. Knezevic.

[3] $https://www.dive-solutions.de/blog/sph-basics$

[4] M. M¨uller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation, pages 154–159, 2003. The link to the paper.

[5] $https://en.wikipedia.org/wiki/Fluid_dynamics$

[6] $https://en.wikipedia.org/wiki/Leapfrog_integration$

[7] FAST FIXED-RADIUS NEAREST NEIGHBORS: INTERACTIVE MILLION-PARTICLE FLUIDS Rama C. Hoetzlein, Graphics Devtech, NVIDIA