Numerical modeling of fluid behavior using Smoothed Particle Hydrodynamics method

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

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

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

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

5th Mykhailo Moroz
Mentor
Zibra
Lviv, Ukraine
michael08840884@gmail.com

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

***Index Terms*—smoothed particle hydrodynamics, computational fluid dynamics, fast fixed-radius nearest neighbors, kernel hydrodynamics**

I. Introduction

The aim of this project is to simulate the fluid behavior. Navier-Stokes equations are commonly used to describe fluids mathematically. 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. These equations have no analytical solutions in general form, so they are often used in computational fluid dynamics [4], [5]. However, these equations can be simplified by using different approximations. In our project, we assume that the fluid is incompressible. We also assume that the fluid is comprised of individual particles, each with its own properties such as mass, position, velocity, and density. Thus, the problem for modeling fluid behavior comes down to calculating the acceleration of each particle due to external forces and updating its position over time. We use the Smoothed-particle hydrodynamics (SPH) method for modeling. SPH is a particle-based, mesh-free Lagrangian method (coordinates move with the fluid in this method). This method is popular in many fields of research, including but not limited to astrophysics, ballistics, volcanology, and oceanography.

II. Possible approaches

A. Methods of fluid modeling

1) Stable Fluids: The Stable Fluids 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 despite 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 dictate careful consideration of simulation parameters and trade-offs in practical applications, which are not suitable for our project.

2) Lattice Boltzmann Method: The Lattice Boltzmann Method (LBM), instead of directly solving the Navier-Stokes equations, 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 multi-physics phenomena, making it suitable for a wide range of applications. However, there are some significant disadvantages that can greatly complicate the simulation work: the method can require substantial computational resources, particularly for high-resolution simulations or scenarios involving turbulent flows. To ensure accurate results, a 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 posesses 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.

4) Conclusion: After weighing the pros and cons, smoothed particle hydrodynamics (SPH) was selected for fluid modeling. Despite the problems with 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 accuracy 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 the SPH framework are done by sticking to the general formulas provided below [1].

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

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

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

The gradient of the scalar field is given by:

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

and the Laplacian is:

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

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 particles. 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 physical parameters (density, viscosity, pressure etc.) based on the surrounding conditions in each frame of the simulation.

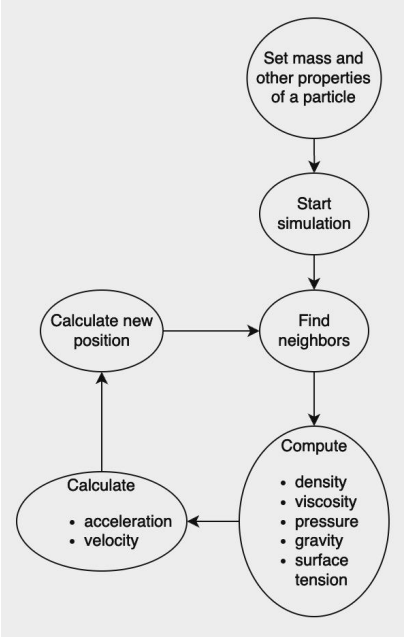

Fig. 1. Finite State Machine of the SPH algorithm.

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 neighboring points, where the weights are determined by a mathematical function – the kernel (also called smoothing kernel, kernel function). Points 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.

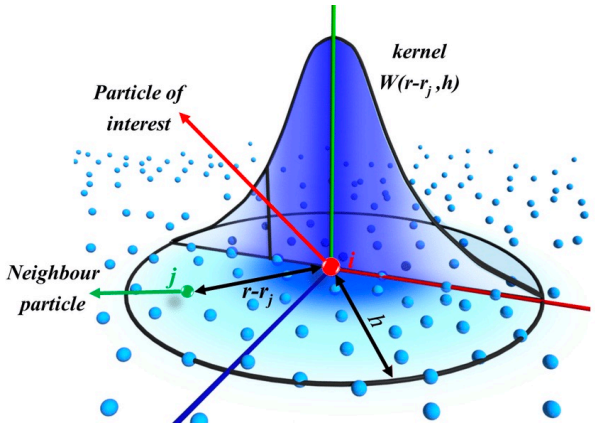

Fig. 2. Schematic illustration of an SPH smoothing kernel function [10].

The algorithm requires three different kernels [1]:

- The Polynomial Kernel.
- The Spiky Kernel.
- 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.

There are also a few general mathematical constraints on smoothing kernels:

$$\int_\Omega W(\vec{r},h)d\Omega = 1,$$
$$W(\vec{r},h) = 0, \quad \|\vec{r}\| \geq h,$$
$$\lim_{h\to 0} W(\vec{r},h) = \delta(\vec{r}),$$

where δ is the Dirac delta function:

$$\delta(\vec{r}) = \begin{cases} \infty & \|\vec{r}\| = 0 \\ 0 & \text{otherwise} \end{cases},$$

and Ω is the computational domain where simulations take place. It includes the whole region where fluid properties are calculated and smoothed using kernel functions. These functions, $W(\vec{r},h)$, are normalized over Ω to ensure accurate and consistent property approximations.

1) The Polynomial Kernel: The polynomial kernel smoothly distributes quantities over space, making it perfect for calculating density and surface tension force. This formula is the default kernel formula suggested by [8].

The Polynomial Kernel:

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

where A can be found out using the condition:

$$\int_\Omega W_{\text{poly}} d\Omega = 1,$$

resulting in:

$$A = \frac{315}{64\pi h^9} \quad \text{for 3D},$$
$$A = \frac{4}{\pi h^8} \quad \text{for 2D}.$$

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:

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

where A can be found out using the condition:

$$\int_\Omega W_{\text{spiky}} d\Omega = 1,$$

resulting in:

$$A = \frac{15}{\pi h^6} \quad \text{for 3D},$$
$$A = \frac{10}{\pi h^5} \quad \text{for 2D}.$$

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:

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

where A can be found out using the condition:

$$\int_\Omega W_{\text{visc}} d\Omega = 1,$$

resulting in:

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

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

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].

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 the divergence of the surface normal that gives the curvature of the surface, σ – the coefficient of surface tension, $\nabla^2 c_s$ – 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. The n is the gradient of the color field c_s that is used to find the surface of the fluid and is given by:

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

3) Viscosity: The viscosity 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 [9].

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 of state*. However, in our implementation, we resort to a more generalized version:

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

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

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

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

IV. Optimization

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

In our initial implementation, we iterated through all particles to compute interactions with just one particle at a time. This resulted in $n(n-1)$ total interactions conducted in each simulation frame. We deemed this method inefficient because particles mainly interact with those that are nearest to them. Therefore, we decided to eliminate unnecessary computations by ignoring particles too far to influence the particle in question significantly.

A. Interaction optimization

A lot of efficient interaction optimization techniques (for example, object bounding boxes in game engines) 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 interacting in the current frame, allowing the more computationally intensive narrow phase to focus only on a small number of such pairs [3].

The main task of the narrow phase involves accurate computation of the intricacies of the interactions and rejection of those not filtered out by the broad phase. Essentially, this phase has been discussed in the earlier sections: a long process of computing individual forces and corresponding kernel values may produce meaningful results for some forces and zeroes for others.

In the case of fluid simulation, it makes sense to split the space into a grid and assign ("sort") particles to individual cells so that later on in the narrow phase interaction, computations are conducted only for particles that are at most n cells away from each other.

B. The broad phase

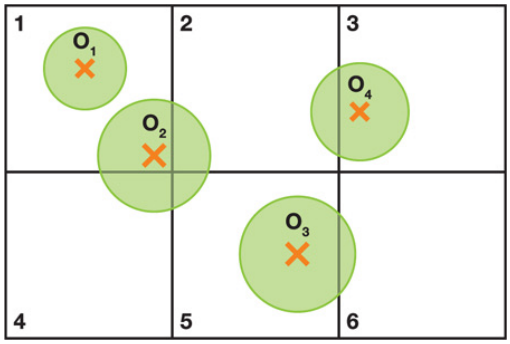

Fig. 3. An Example of 2D Spatial Subdivision of Four Particles [3].

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 Figure 3, Cell 1 includes particles O1, O2; Cell 2 – none; Cell 3 – particle O4; Cell 4 – none; Cell 5 – particle O3; Cell 6 – none.

2) Sorting algorithms: The choice of an efficient sorting algorithm determines how quickly we can build a list of particles belonging to individual cells.

One possible sorting algorithm is the radix sort. Radix sort sorts by the 32-bit cell IDs. An important property of the radix sort is that it sorts the elements iteratively by their digits (or tuples of bits) from the least to most significant, so it takes multiple passes to complete [3], [7].

As it can be seen from Figure 4, we have a set of particles and cells. The first steps are finding the largest cell ID in order to figure out how many passes there will be and determining where the center of each particle lies (which can be done at the end of the previous frame). Then, we iteratively sort the particles by the digits (or tuples of them) of cell IDs associated with them in the following way: first, we calculate the total number of particles sharing the same digit (tuple) at the current iteration's position of cell ID, then for each possible digit value we sum up the number of particles with current digit values smaller than it ("prefix sum" or "partial sum"), thus obtaining an offset for its particles and finally we put the particles' IDs from input buffer to indices in the output buffer based on the prefix sum of the corresponding digit value. The above is repeated as long as we can shift the digit (tuple) position of cell IDs to the left.

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.

3) Implementation: We decided to employ counting sort since radix sort's stability is rather difficult to guarantee in the SIMT (Single instruction, multiple threads, used by GPU) context. Our sorting implementation uses five buffers in total for pairs of type $\langle particleID; cellID \rangle$, number of particles in a cell, prefix sums of the cells, the pairs ordered by cell ID, and specification where in the output array the pairs of a given cell begin.

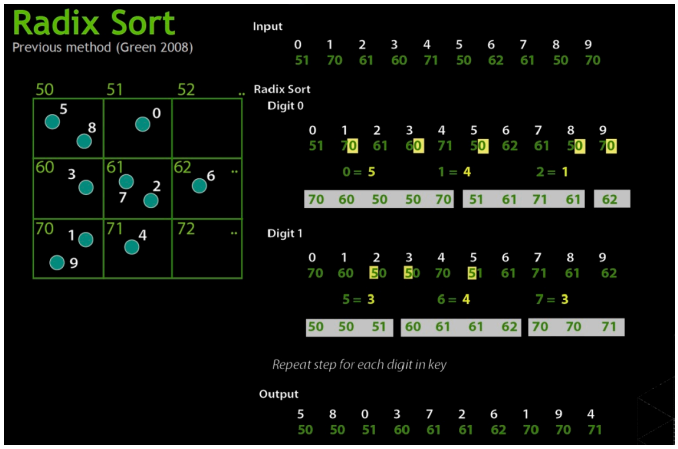

Fig. 4. Example of Radix Sort from [7].

V. Results

The animation of our simulation involving 4800 particles can be seen in the following link to Google Drive. As can be observed from the performance panel, the simulation involving sorting tends to be more stable and exhibits higher maximum values of frame rate than the one without sorting.

VI. Conclusion

In this project, we modelled fluid behavior with the Smoothed Particle Hydrodynamics method. During our work,

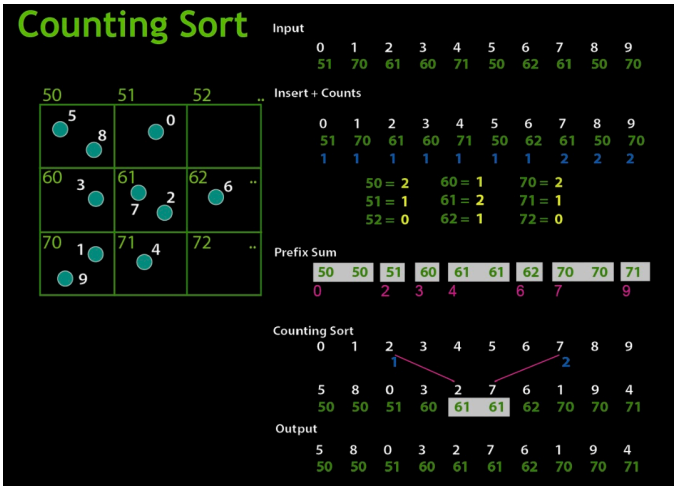

Fig. 5. Example of Counting Sort from [7]

we leveraged the Leap Frog method for integration. However, while working on optimization, we were perplexed when implementing the sorting algorithm. First, we tried to use radix sort, but it turned out too hard to guarantee its stability and efficiency in a parallel environment at the same time. So we implemented the counting sort instead. We also got more familiar with Unity Game Engine, the principles of GPU computations, and shader programming. The code of the project is available at GitHub repository.

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. Link to paper

[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. Link to paper

[3] Scott Le Grand. Broad-Phase Collision Detection with CUDA. GPU Gems 3, Chapter 32. Available online at NVIDIA Developer website: Link to the book

[4] Fluid Dynamics

[5] Sph Basics

[6] Leapfrog Integration Method

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

[8] M. Müller, 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.

[9] Modeling viscous forces in SPH Link to the presentation

[10] Giuseppe Giorgi PhD Thesis, 2018, doi: 10.13140/RG.2.2.17893.96481, Link.