Masters Project

The Simulation of Fluids:

Newtonian and Non-Newtonian Fluid Simulator based on Smoothed Particle Hydrodynamics approaches

Volpicelli Milo
i7630684

MSc Computer Animation and Visual Effects
Bournemouth University
NCCA

August 2018

Bournemouth
University

Index

Equations

Figures

Abstract

Within this paper, it will be firstly presented briefly the history of Computational Fluid Dynamics by exposing previous and current state of the art techniques with particular focus on SPH approaches. Successively, the mathematical theory behind methods and algorithms utilized to achieve this project's implementation will be discussed and explained. The current developed application described in this paper can perform simulations of both Newtonian and Non-Newtonian fluids according to two different SPH algorithms (WCSPH and PCISPH). The details of the developed piece of software will be exposed and the achieved results will be critically reviewed.

1. Introduction

Simulations of fluid, due to their high demand of use within the VFX and gaming industries, are an important and significant topic within Computer Graphics. Even though the modelling of natural phenomena such as fluids has a prolonged history, it still remains a challenge for modern day researchers to achieve improving results. (FxGuide 2011). Moreover, computational efficiency becomes a fundamental topic while dealing with complex physically-based simulations such as liquids. Therefore, various methods, techniques and algorithms have been studied and implemented following different approaches to try and achieve better results.

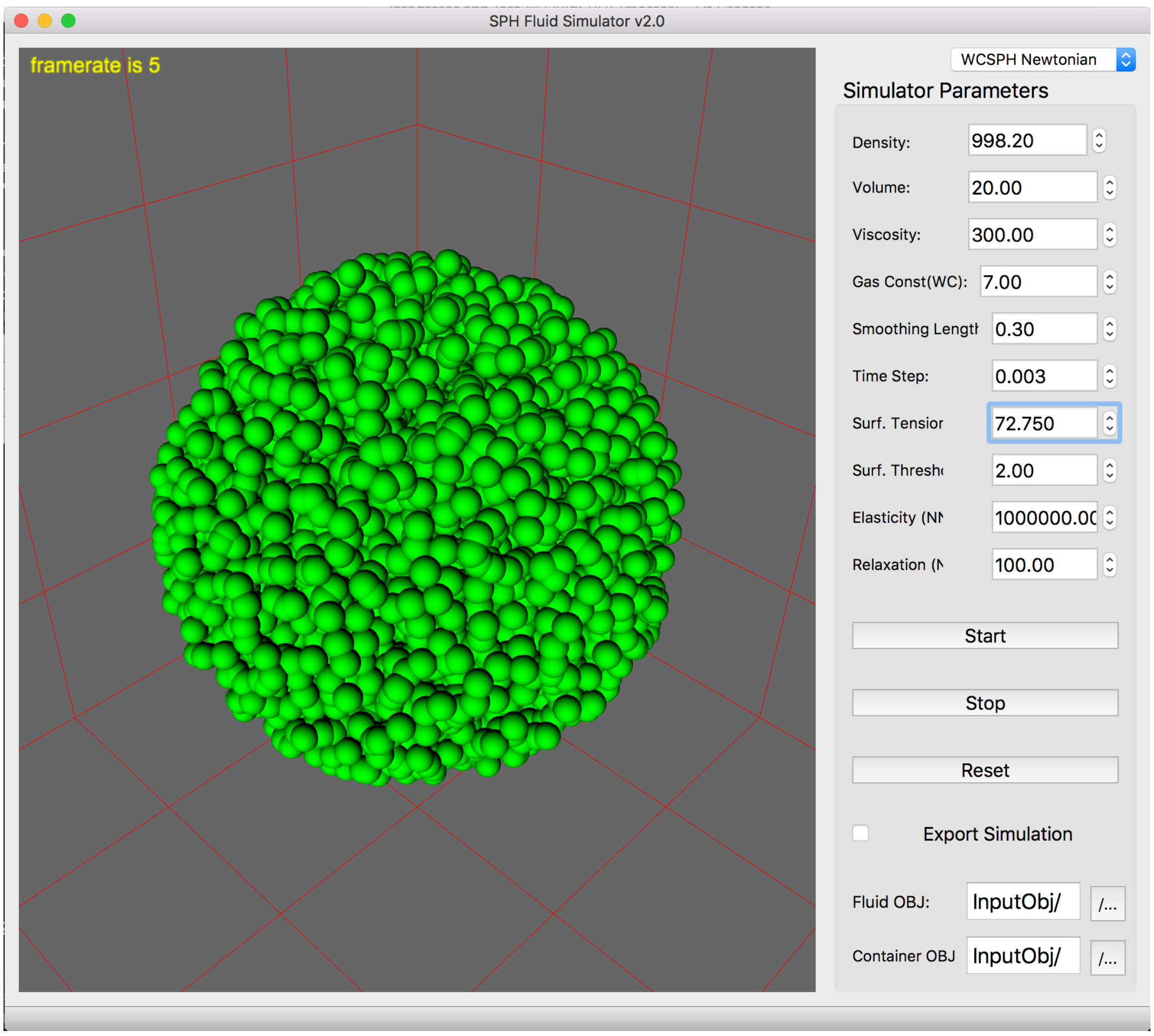
This paper will be structured in the following way: firstly, the most functional and noteworthy methods within the Computer Graphics community will be chronologically cited and outlined in the subsequent section. Additionally, the technical background and documentation required for the developed program will be included.

Furthermore, in order to distinct the different nature of these two diverse methods, a critical comparison between Eulerian and Lagrangian approaches becomes essential and thus it is present at the beginning of the third section of this document.

Successively, fundamental SPH theory and the description of both Newtonian and Non-Newtonian fluid models are described as both models will be present within the simulator. In addition, the program's features and algorithms (WCSPH and PCISPH) on which the current piece of software is based will be exposed and explained in section four.

The description of the program's structure and the overall application's flow will follow: the simulator was implemented in C++ based on OpenGL for representation and Qt framework for the GUI.

The simulated particle data can be exported from the simulator as text files and can be imported in different renderers for a more realistic visualisation. For this project, Houdini's Particle Fluid Surface node has been used to reconstruct the fluid's surface and Mantra to render it out. In the last section, the results are analysed and compared as well as current issues are described.


SPH Fluid Simulator v2.0

framerate is 5

WCSPH Newtonian

Simulator Parameters

Density:	998.20
Volume:	20.00
Viscosity:	300.00
Gas Const(WC):	7.00
Smoothing Lengt	0.30
Time Step:	0.003
Surf. Tensior	72.750
Surf. Thresh	2.00
Elasticity (N	1000000.0
Relaxation (	100.00

Start

Stop

Reset

☐ Export Simulation

Fluid OBJ:	InputObj/	/...
Container OBJ	InputObj/	/...

Figure 1: Developed Application Interface (Personal collection).

2. Previous Work

During the 1960's, a group of researchers known as the T3 group of the Los Alamos National Laboratory discovered the first mathematical models to implement a fluid simulation (Evans and Harlow 1960). This first developed method was known as "particle-in-cell (PIC) method". Few years later, the same group of researchers developed a second method, the Marker and Cell Grid method (MAC), that became widely used and is still used today in different hybrid methods due to its efficiency and functionality (Fxguide 2011). Following Evans and Harlow's footsteps, in 1986 Bracknill and Ruppel developed the Fluid Implicit Particle method (FLIP). Successively, in 1996, the Navier-Stokes equations were solved for three dimensions and used to simulate fluids based on Eulerian fixed grids (Foster and Metaxas 1996).

In 1977, a group of astrophysics invented the Smooth Particle Hydrodynamics method to simulate galactic motion and formation (Lucy et al. 1977). This method was then adapted to simulate liquids (Gingold and Monaghan 1977). In 1996, a new method for the calculation of the pressure and to calculate particle interactions in the SPH method was introduced by Desbrun and Gascuel. This allowed the SPH method to gain popularity in the Computer Graphics community. In 2003, free surface flows were simulated through SPH and the solving of the three-dimensional Navier-Stokes equations (Müller et al. 2003). Additionally, within the same research paper, it is proposed a method for calculating and including surface tension calculations in the simulation using colour fields.

During the same year, an optimized spatial hashing neighbourhood searching algorithm was developed for collision detection of deformable objects (Teschner et al. 2003). In 2007, the

Weakly Compressible SPH solver was introduced and allowed to achieve fluid simulations with low compressibility (Becker and Tescher 2007). Successively, in 2009, Solenthaler and Pajarola proposed the Predictive Corrective Incompressible method in order to achieve incompressibility. However, better and more stable fluid incompressibility was achieved thanks to Raveendran's hybrid method (2011). One year later, a local Poison method was developed for ever more accurate incompressibility (He et al. 2012). In the next few years, diverse cutting-edge researchers such as Solenthaler, Tescher and Ihmsen, worked together on a complete and extensive SPH method implementation (Ihmsen et al. 2014).

In 2018, a hybrid incompressible SPH fluid solver with interface handling for boundary collisions has been proposed (Takahashi et al.). During the same year a FLIP method has been adapted to work on the GPU using Sparse Volumes producing very appealing results (Wu et al. 2018). At the same time, in order to achieve even more computational performance, an alternative has been found in developing a system that automatically distributes grid-based and hybrid simulations across cloud computing nodes, utilizing efficiently the parallelized computational power that cloud services have to offer (Mashayekhi et al. 2018).

Regarding Non-Newtonian fluids in terms of Lagrangian approaches, different researchers have simulated viscous fluids with particle systems (Chang et al 2009), such as Miller and Pierce (1989) and Terzopoulos (1991). In 1999, Stora et al. realistically simulated lava flows according to an SPH approach. A few years later, Mao and Yang (2006) developed an SPH simulator based on co-rotational Maxwell modes. Successively, Paiva et al. (2007; 2009), edited the Navier-Stokes equations in order to simulate viscoelastic fluid and its interactions with solids by using a Generalized Newtonian Liquid mathematical model (Chang et al. 2009).

A few months later in 2009, Chang et al. implemented viscoelastic fluid solver by further modifications of the Navier-Stokes equations, increasing the variety of possible achievable simulation results and performance. Recently, in 2015, a stable and efficient particle-based method that can simulate coiling ad buckling, based on implicit integration formulations, has been developed (Takahashi et al. 2015). Moreover, this method granted more robustness for viscosity integration, it can handle variable viscosity and larger time-steps were allowed.

3. Technical Background

3.1 Eulerian Grid-based VS Lagrangian Particle-based approaches

Two main approaches have been discovered over the course of CFD's history in order to achieve a successful fluid solver: the Eulerian grid-based method and the Lagrangian particle-based method. Currently, the state of the art in fluid dynamic simulations is dictated by hybrid methods which involve parts of both approaches.

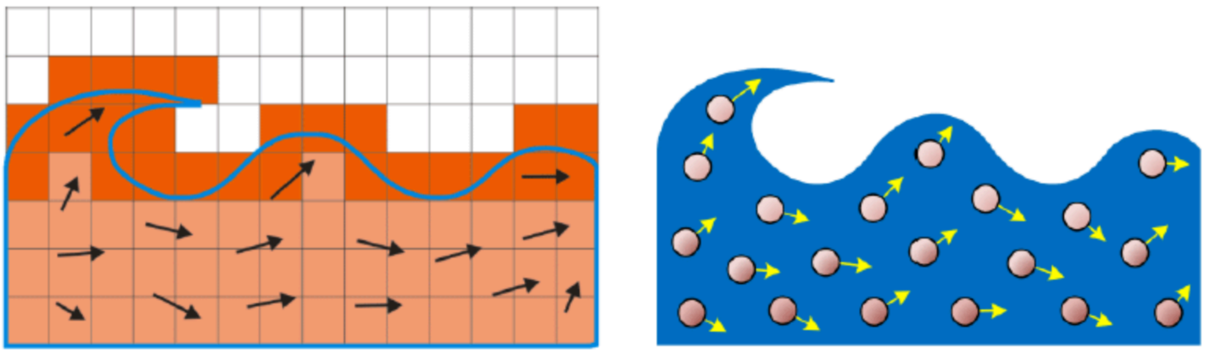


Figure 2: Eulerian vs Lagrangian approaches (Liu et al. 2018).

Following a Eulerian grid-based method, the fluid's quantities are stored within the points in space (the grid). Moreover, this technique represents the fluid flow though the occurring changes in the field of velocity, density and pressure. Furthermore, this kind of approach can output fluid simulations of very high quality and are generally more precise than Lagrangian methods in handling pressure and incompressibility issues (Zhang et al. 1993). However, they are highly dependent on the grid resolution size and solving of spatial PDEs is required.

The Lagrangian approach to simulate liquid dynamics, instead, treats the fluid as a particle system. The quantities (attributes) such as position, velocity, mass, etc., defining the fluid are stored in each particle. Every particle will contribute to the fluid's flow by its reaction to forces. This method does not have any spatial limitations. However, to obtain a realistic fluid's motion, a large number of particles is required, making this method less computationally efficient. The method used within this project's implementation falls within the category of Lagrangian approach.

3.2 SPH Theory

Smoothed particle hydrodynamics (SPH) was invented by Lucy et al. in 1977 to simulate non-axisymmetric phenomena present in astrophysics (Grindold and Monaghan 1977). The SPH method is particle system based. Therefore, it does not require a grid for the calculations of spatial derivatives (Monaghan 1988). According to this method, the equations of momentum and energy become ordinary differential equations. As instance, the pressure gradient will be expressed as a force between particles (Monaghan 1885). Furthermore, an interpolation method that allows any function to be expressed in terms of its values at a set of disordered points is used and defined as "*the heart of SPH*" (Monaghan 1988).

The integral interpolant of any function $A(\boldsymbol{r})$ is defined by:

$$A_I(\boldsymbol{r}) = \int A(\boldsymbol{r}')\, W(\boldsymbol{r} - \boldsymbol{r}', h) \boldsymbol{dr}'$$

Equation 1: Integral interpolant of function $A_I(\boldsymbol{r})$(Monaghan 1992).

The interpolation is over the total space and W is an interpolant kernel provided with two specific proprieties:

$$\int W(\boldsymbol{r}-\boldsymbol{r}', h)\boldsymbol{dr}' = 1$$

Equation 2: Kernel's first condition of existence.

and

$$\lim_{h\to 0} W(\boldsymbol{r}-\boldsymbol{r}', h) = \delta(\boldsymbol{r}-\boldsymbol{r}')$$

Equation 3: Kernel's second condition of existence.

The limit is to be considered as the limit of the corresponding integral interpolants (Monaghan 1988). The integral interpolant is then approximated through a summation interpolant:

$$A_S(\boldsymbol{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\boldsymbol{r}-\boldsymbol{r}_b, h)$$

Equation 4: Summation of the interpolant to approximate the integral interpolant.

Where the summation index b denotes a particle id. The considered particle has mass m_b, position $\boldsymbol{r}_b$ and density ρ_b. The value of any quantity A at $\boldsymbol{r}_b$ is denoted by A_b.

Monaghan (1988, 2) continues stating the fundamental point of the SPH method: "*The essential point is that we can construct a differentiable interpolant of a function from its values at the particles by using differentiable kernels*". Therefore, derivatives of this interpolant are calculated through ordinary differentiation, with no need of finite differences or grids.

Consequentially, ∇A would result as:

$$\nabla A_S(\boldsymbol{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\boldsymbol{r} - \mathbf{r}', h)$$

Equation 5: Integral Interpolant gradient value.

And the Laplacian:

$$\nabla^2 A_S(\boldsymbol{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla^2 W(\boldsymbol{r} - \mathbf{r}', h)$$

Equation 6: Integral Interpolant Laplacian value.

3.3 Navier-Stokes Equations

The Navier-Stokes equations are a set of non-linear equations which describe a fluid's flow at any given point. A three-dimensional version of the Navier-Stokes equations has been firstly derived by Foster and Metaxas (1996). Furthermore, these set of equations are based on the Conservation of Mass and Conservation of Momentum Laws (Müller et al 2003). The mass in an SPH approach implementation is usually defined as a user-set constant. Therefore, the following equation (Equation 7) is used only in Eulerian grid-based methods.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

Equation 7: Conservation of Mass

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \rho \boldsymbol{g} + \mu \nabla^2 \mathbf{v}$$

Equation 8: Navier-Stokes equation derived from the Conservation of Momentum equation

$$\rho \left(\frac{D\mathbf{v}}{Dt}\right) == -\nabla p + \rho \boldsymbol{g} + \mu \nabla^2 \mathbf{v}$$

Equation 9: Navier-Stokes equation reformulated by Müller et al. (2003).

In these equations, $\boldsymbol{\rho}$ is the density, $\boldsymbol{p}$ is the pressure term, $\boldsymbol{g}$ is gravity and/or external forces and $\mu\nabla^2\mathbf{v}$ corresponds to the viscosity term.

Interestingly, this reformulated equation can only be used for simulations based in particle-system techniques due to the absence of the convection term ($\mathbf{v} \cdot \boldsymbol{\nabla}\mathbf{v}$). Since the moving fluid is composed of particles, its resulting velocity field is equal to the time derivative of the velocity of all the particles (Müller et al. 2003). This is only valid for Lagrangian approaches. In addition, this particular reformulation of the Navier-Stokes equation is only functional for simulating fluids with a linear relationship between shear stress and shear rate (Newtonian Fluids).

Consequentially, in order to describe the flow of Non-Newtonian fluids, Mao and Yang (2006) reformulate this equation as:

$$\frac{d\mathrm{v}}{dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \mathrm{T} + \frac{\mu}{\rho}\nabla^2 \mathrm{v} + \frac{1}{\rho}\,\mathbf{g}$$

Equation 10: Navier-Stokes reformulated by Mao and Yang for Non-Newtonian Fluid model (2006).

With T being the stress tensor term. Specifically, Non-Newtonian fluids differ from Newtonian due to their non-linear relationship between shear stress and shear rate (Figure 2). Therefore, the stress term requires to be computed and included in the net force equation.

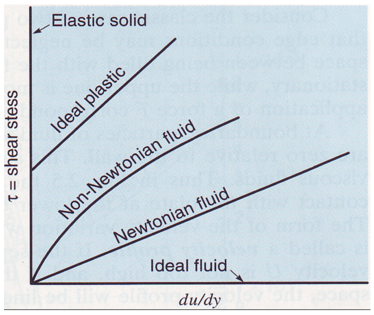


Figure 3: Graph showing different shear stress and shear rate functions for different fluids (Fandom 2018).

4.0 Implementation Details

This project's implementation has been designed for being able to simulate both Newtonian and Non-Newtonian fluids according to two different SPH algorithms. The algorithms that were taken in consideration and as reference for this developed piece of software are respectively the WCSPH algorithm (Müller et al. 2003; Becker and Teschner 2007) and the PCISPH (Solenthaler and Pajarola 2009). Regarding Non-Newtonian Fluids, the methods and algorithm proposed by Mao and Yang in 2006 were adapted and modified with influence from the respective approaches of Paiva et al. (2009) and Takahashi et al. (2015).

In terms of software design, the main solver algorithms are held within the *FluidSolver* class, as well as the fluid quantities parameters. Within the *Particle* class, the attributes of the particle instances are stored. Furthermore, specific classes have been designed for the respective components of the program such as *Forces, Kernels, Spatial Hashing, Integrators* and *Boundaries*.

The application has been written in C++, based on OpenGL and NGL graphic libraries and utilizes Qt framework for the GUI. To support tensors, Eigen C++ library was found the most efficient and complete (Jacob 2015). The visualization of the fluid has been achieved through the use of SideFX's Houdini and it's built in renderer.

In the following pages, the program's UML diagram and main solver algorithms will be exposed:

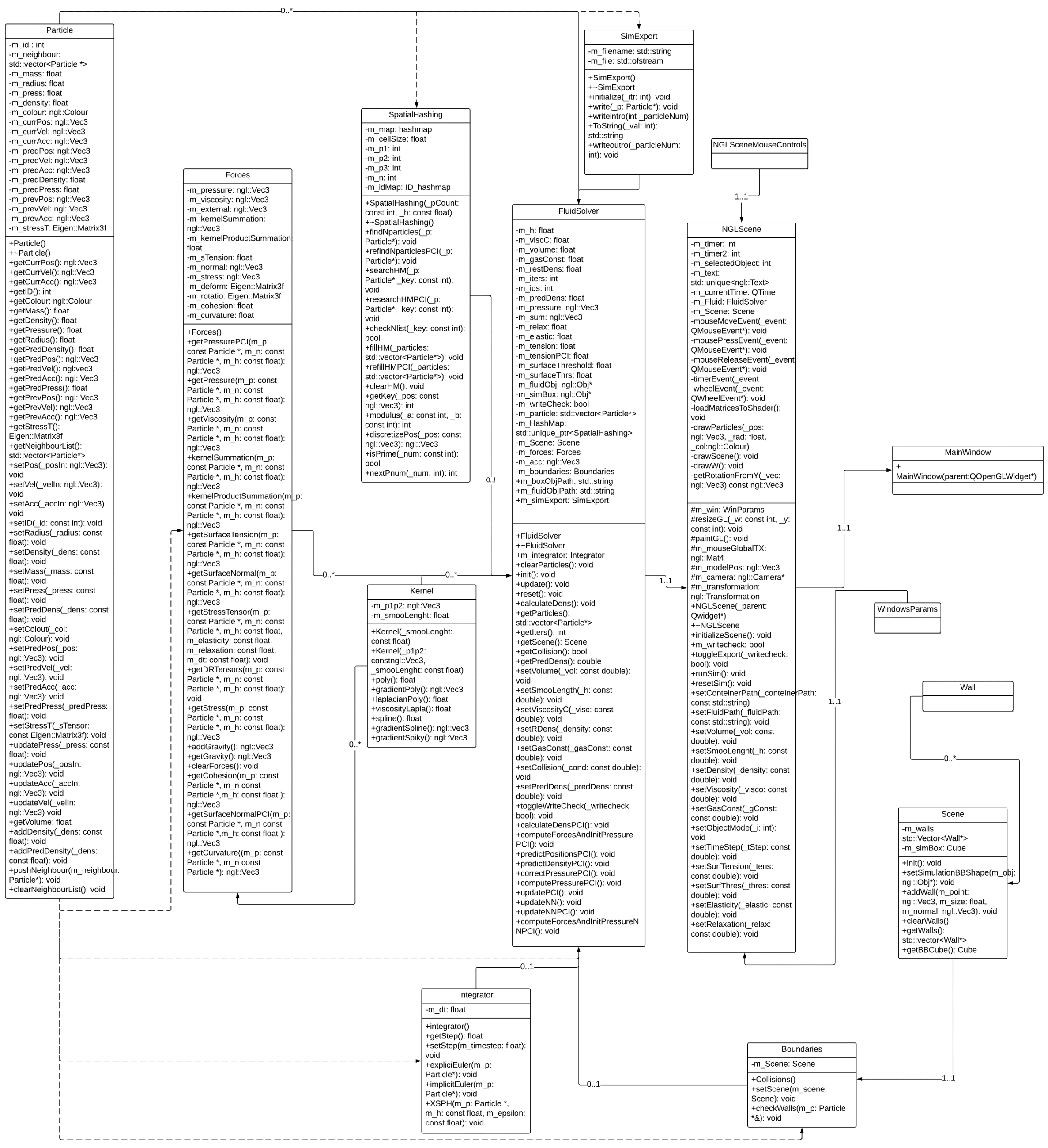

0..*
Particle
-m_id : int
-m_neighbour: std::vector<Particle *>
-m_mass: float
-m_radius: float
-m_press: float
-m_density: float
-m_colour: ngl::Colour
-m_currPos: ngl::Vec3
-m_currVel: ngl::Vec3
-m_currAcc: ngl::Vec3
-m_predPos: ngl::Vec3
-m_predVel: ngl::Vec3
-m_predAcc: ngl::Vec3
-m_predDensity: float
-m_predPress: float
-m_prevPos: ngl::Vec3
-m_prevVel: ngl::Vec3
-m_prevAcc: ngl::Vec3
-m_stressT: Eigen::Matrix3f
+Particle()
+~Particle()
+getCurrPos(): ngl::Vec3
+getCurrVel(): ngl::Vec3
+getCurrAcc(): ngl::Vec3
+getID(): int
+getColour: ngl::Colour
+getMass(): float
+getDensity(): float
+getPressure(): float
+getRadius(): float
+getPredDensity(): float
+getPredPos(): ngl::Vec3
+getPredVel(): ngl:vec3
+getPredAcc(): ngl::Vec3
+getPredPress(): float
+getPrevPos(): ngl::Vec3
+getPrevVel): ngl::Vec3
+getPrevAcc(): ngl::Vec3
+getStressT(): Eigen::Matrix3f
+getNeighbourList(): std::vector<Particle*>
+setPos(_posIn: ngl::Vec3): void
+setVel(_velIn: ngl::Vec3): void
+setAcc(_accIn: ngl::Vec3) void
+setID(_id: const int): void
+setRadius(_radius: const float): void
+setDensity(_dens: const float): void
+setMass(_mass: const float): void
+setPress(_press: const float): void
+setPredDens(_dens: const float): void
+setColout(_col: ngl::Colour): void
+setPredPos(_pos: ngl::Vec3): void
+setPredVel(_vel: ngl::Vec3): void
+setPredAcc(_acc: ngl::Vec3): void
+setPredPress(_predPress: float): void
+setStressT(_sTensor: const Eigen::Matrix3f): void
+updatePress(_press: const float): void
+updatePos(_posIn: ngl::Vec3): void
+updateAcc(_accIn: ngl::Vec3): void
+updateVel(_velIn: ngl::Vec3) void
+getVolume: float
+addDensity(_dens: const float): void
+addPredDensity(_dens: const float): void
+pushNeighbour(m_neighbour: Particle*): void
+clearNeighbourList(): void
Forces
-m_pressure: ngl::Vec3
-m_viscosity: ngl::Vec3
-m_external: ngl::Vec3
-m_kernelSummation: ngl::Vec3
-m_kernelProductSummation: float
-m_sTension: float
-m_normal: ngl::Vec3
-m_stress: ngl::Vec3
-m_deform: Eigen::Matrix3f
-m_rotatio: Eigen::Matrix3f
-m_cohesion: float
-m_curvature: float
+Forces()
+getPressurePCI(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+getPressure(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+getViscosity(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+kernelSummation(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+kernelProductSummation(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+getSurfaceTension(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+getSurfaceNormal(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+getStressTensor(m_p: const Particle *, m_n: const Particle *, m_h: const float, m_elasticity: const float, m_relaxation: const float, m_dt: const float): void
+getDRTensors(m_p: const Particle *, m_n: const Particle *, m_h: const float): void
+getStress(m_p: const Particle *, m_n: const Particle *, m_h: const float): ngl::Vec3
+addGravity(): ngl::Vec3
+getGravity(): ngl::Vec3
+clearForces(): void
+getCohesion(m_p: const Particle *, m_n const Particle *,m_h: const float): ngl::Vec3
+getSurfaceNormalPCI(m_p: const Particle *, m_n const Particle *,m_h: const float): ngl::Vec3
+getCurvature((m_p: const Particle *, m_n const Particle *): ngl::Vec3
SpatialHashing
-m_map: hashmap
-m_cellSize: float
-m_p1: int
-m_p2: int
-m_p3: int
-m_n: int
-m_idMap: ID_hashmap
+SpatialHashing(_pCount: const int, _h: const float)
+~SpatialHashing()
+findNparticles(_p: Particle*): void
+refindNparticlesPCI(_p: Particle*): void
+searchHM(_p: Particle*,_key: const int): void
+researchHMPCI(_p: Particle*,_key: const int): void
+checkNlist(_key: const int): bool
+fillHM(_particles: std::vector<Particle*>): void
+refillHMPCI(_particles: std::vector<Particle*>): void
+clearHM(): void
+getKey(_pos: const ngl::Vec3): int
+modulus(_a: const int, _b: const int): int
+discretizePos(_pos: const ngl::Vec3): ngl::Vec3
+isPrime(_num: const int): bool
+nextPnum(_num: int): int
SimExport
-m_filename: std::string
-m_file: std::ofstream
+SimExport()
+~SimExport
+initialize(_itr: int): void
+write(_p: Particle*): void
+writeintro(int _particleNum)
+ToString(_val: int): std::string
+writeoutro(_particleNum: int): void
NGLSceneMouseControls
1..1
FluidSolver
-m_h: float
-m_viscC: float
-m_volume: float
-m_gasConst: float
-m_restDens: float
-m_iters: int
-m_ids: int
-m_predDens: float
-m_pressure: ngl::Vec3
-m_sum: ngl::Vec3
-m_relax: float
-m_elastic: float
-m_tension: float
-m_tensionPCI: float
-m_surfaceThreshold: float
-m_surfaceThrs: float
-m_fluidObj: ngl::Obj*
-m_simBox: ngl::Obj*
-m_writeCheck: bool
-m_particle: std::vector<Particle*>
-m_HashMap: std::unique_ptr<SpatialHashing>
-m_Scene: Scene
-m_forces: Forces
-m_acc: ngl::Vec3
-m_boundaries: Boundaries
+m_boxObjPath: std::string
+m_fluidObjPath: std::string
+m_simExport: SimExport
+FluidSolver
+~FluidSolver
+m_integrator: Integrator
+clearParticles(): void
+init(): void
+update(): void
+reset(): void
+calculateDens(): void
+getParticles(): std::vector<Particle*>
+getIters(): int
+getScene(): Scene
+getCollision(): bool
+getPredDens(): double
+setVolume(_vol: const double): void
+setSmooLength(_h: const double): void
+setViscosityC(_visc: const double): void
+setRDens(_density: const double): void
+setGasConst(_gasConst: const double): void
+setCollision(_cond: const double): void
+setPredDens(_predDens: const double): void
+toggleWriteCheck(_writecheck: bool): void
+calculateDensPCI(): void
+computeForcesAndInitPressurePCI(): void
+predictPositionsPCI(): void
+predictDensityPCI(): void
+correctPressurePCI(): void
+computePressurePCI(): void
+updatePCI(): void
+updateNN(): void
+updateNNPCI(): void
+computeForcesAndInitPressureNNPCI(): void
NGLScene
-m_timer: int
-m_timer2: int
-m_selectedObject: int
-m_text: std::unique<ngl::Text>
-m_currentTime: QTime
-m_Fluid: FluidSolver
-m_Scene: Scene
-mouseMoveEvent(_event: QMouseEvent*): void
-mousePressEvent(_event: QMouseEvent*): void
-mouseReleaseEvent(_event: QMouseEvent*): void
-timerEvent(_event
-wheelEvent(_event: QWheelEvent*): void
-loadMatricesToShader(): void
-drawParticles(_pos: ngl::Vec3, _rad: float, _col:ngl::Colour)
-drawScene(): void
-drawW(): void
-getRotationFromY(_vec: ngl::Vec3) const ngl::Vec3
#m_win: WinParams
#resizeGL(_w: const int, _y: const int): void
#paintGL(): void
#m_mouseGlobalTX: ngl::Mat4
#m_modelPos: ngl::Vec3
#m_camera: ngl::Camera*
#m_transformation: ngl::Transformation
+NGLScene(_parent: Qwidget*)
+~NGLScene
+initializeScene(): void
+m_writecheck: bool
+toggleExport(_writecheck: bool): void
+runSim(): void
+resetSim(): void
+setConteinerPath(_conteinerPath: const std::string)
+setFluidPath(_fluidPath: const std::string): void
+setVolume(_vol: const double): void
+setSmooLenght(_h: const double): void
+setDensity(_density: const double): void
+setViscosity(_visco: const double): void
+setGasConst(_gConst: const double): void
+setObjectMode(_i: int): void
+setTimeStep(_tStep: const double): void
+setSurfTension(_tens: const double): void
+setSurfThres(_thres: const double): void
+setElasticity(_elastic: const double): void
+setRelaxation(_relax: const double): void
MainWindow
+ MainWindow(parent:QOpenGLWidget*)
1..1
WindowsParams
1..1
1..1
Wall
0..*
Scene
-m_walls: std::Vector<Wall*>
-m_simBox: Cube
+init(): void
+setSimulationBBShape(m_obj: ngl::Obj*): void
+addWall(m_point: ngl::Vec3, m_size: float, m_normal: ngl::Vec3): void
+clearWalls()
+getWalls(): std::vector<Wall*>
+getBBCube(): Cube
0..*
0..*
0..!
Kernel
-m_p1p2: ngl::Vec3
-m_smooLenght: float
+Kernel(_smooLenght: const float)
+Kernel(_p1p2: constngl::Vec3, _smooLenght: const float)
+poly(): float
+gradientPoly(): ngl::Vec3
+laplacianPoly(): float
+viscosityLapla(): float
+spline(): float
+gradientSpline(): ngl::vec3
+gradientSpiky(): ngl::Vec3
0..*
0..1
Integrator
-m_dt: float
+integrator()
+getStep(): float
+setStep(m_timestep: float): void
+expliciEuler(m_p: Particle*): void
+implicitEuler(m_p: Particle*): void
+XSPH(m_p: Particle *, m_h: const float, m_epsilon: const float): void
0..1
Boundaries
-m_Scene: Scene
+Collisions()
+setScene(m_scene: Scene): void
+checkWalls(m_p: Particle *&): void
1..1

Figure 4: UML Diagram of the developed application.

Newtonian Fluids:

```
For every particle i:
    Find Neighbors of i(j);
    Calculate Density $\rho_i$ (Equation 12);
    Calculate Pressure $p_i$ using Density $\rho_i$ according to Tait's EOS (Equation 14);
For every particle i:
    If particle i is not j:
        Compute Pressure Force (Equation 17);
        Compute Viscosity Force (Equation 35);
    Compute Surface Tension (Equation 39);
    Add External Forces;
For every particle i:
    Integrate particle velocity (Equation 62a);
    Integrate particle position (Equation 62b);
    Boundaries detection;
    XSPH velocity correction (Equation 63);
```

Non-Newtonian Fluids:

```
For every particle i:
    Find Neighbors of i(j)
    Calculate Density $\rho_i$   $\rho_i = \sum_j m_j W_{ij}$ (Equation 12);
    Calculate Pressure $p_i$ using Density $\rho_i$ according to Tait's EOS (Equation 14);
For every particle i:
    Calculate $\omega_i$ and $D_i$ (Equation 48 and 51);
For every particle i:
    If particle i is not j:
        Compute Pressure Force (Equation 17);
        Compute Viscosity Force (Equation 35);
    Compute Stress Tensor (Equation 55);
    Add External Forces;
For every particle i:
    Integrate particle velocity (Equation 62a);
    Integrate particle position (Equation 62b);
    Boundaries detection;
    XSPH velocity correction (Equation 63);
```

Newtonian Fluids:

```
For every particle i:
    Find Neighbors of i(j);
    Initialize Density;
For every particle i:
    Compute Viscosity and External forces;
    Initialize local Pressure and Pressure;
While(iter<minIterations) do
    For every particle i:
        Predict velocity of particle i $\boldsymbol{v}_i^*(t+1)$;
        Predict position of particle i $\boldsymbol{x}_i^*(t+1)$;
    For every particle i:
        Predict Density $\rho_i^*(t+1)$ (Equation 13);
        Predict Density Variation $\rho_{error}^*(t+1)$;
        Update Pressure $p_i(t) += f\ (\rho_{error}^*(t+1)$;
    For every particle i:
        Compute Pressure Force $\boldsymbol{F}^p(t)$ (Equation 24);
For every particle i:
    Integrate particle velocity (Equation 62a);
    Integrate particle position (Equation 62b);
    Boundaries detection;
    XSPH velocity correction (Equation 63)
```

Non-Newtonian Fluids:

```
For every *particle i:*
    Find Neighbors of *i(j);*
    Initialize Density;
For every *particle i:*
    Calculate $\omega_i$ and $D_i$ (Equation 48 and 51);
For every *particle i:*
    Compute Viscosity and External forces;
    Compute Stress Tensor (Equation 55);
    Initialize local Pressure and Pressure Force;
**While**(iter<minIterations) **do**
    For every *particle i:*
        Predict velocity of *particle i* $\boldsymbol{v}_i^*(t+1)$;
        Predict position of *particle i* $\boldsymbol{x}_i^*(t+1)$;
    For every *particle i:*
        Predict Density $\rho_i^*(t+1)$ (Equation 13);
        Predict Density Variation $\rho_{error}^*(t+1)$;
        Update Pressure $p_i(t) += f\ (\rho_{error}^*(t+1)$;
    For every *particle i:*
        Recalculate Neighbors of *i (j);*
        Compute Pressure Force $\boldsymbol{F}^p(t)$ (Equation 24);
For every *particle i:*
    Integrate particle velocity (Equation 62a);
    Integrate particle position (Equation 62b);
    Boundaries detection;
    XSPH velocity correction (Equation 63);
```

4.1 Mass

The mass of every single particle is computed during the initialization phase of the program, according to a modified version of the volumetric density equation (Kelager 2006).

$$m_i = \frac{\rho_0 \cdot V}{n}$$

Equation 11: Volumetric density equation (Kelager 2006).

Where ρ_0 is the rest density of the fluid, V is the user set volume and n is the number of vertices (particles) loaded from the .obj file.

The calculated mass will stay the same for the course of the entire simulation process. Moreover, the calculated mass is dependent from the number of particles loaded as .obj file and from both rest density and volume parameter set by the user. This allows the program to perform mass calculations for an arbitrary set of parameters given as input. This process is the same across all the different solvers.

4.2 Density

In order to perform density calculations for every particle, the classic SPH technique known as Density summation is used (Monaghan 2005).

$$\rho_i = \sum_j m_j W(\boldsymbol{x_i} - \boldsymbol{x_j}, h)$$

Equation 12: Density summation equation for WC solver (Monaghan 2005).

Additionally, the PCISPH solver requires a density prediction process to compute density variation in order to update the pressure values of each particle. A density summation analogue to Equation 12 is used for density prediction (Solenthaler and Pajarola 2009).

$$\rho_i^* = \rho_i(t+1) = \sum_j m_j W(\boldsymbol{x_i}(t+1) - \boldsymbol{x_j}(t+1), h)$$

Equation 13: Density summation prediction equation for PCI solver (Solenthaler and Pajarola 2009).

4.3 Pressure (WCSPH)

During the initial phase of the WCSPH solver's algorithm, a local scalar pressure value is calculated and assigned to every particle of the fluid according to a modified version of the Equation of State (EOS) described by Tait (Yu and Turk 2010). This version of Tait's equation is more efficient than the original EOS used in Becker's and Teschner's paper (2007) due to the absence of power calculations. Alternatively, within the WC solver's algorithm is also present a method to calculate scalar pressure values according to Poisson's equation (Müller et al. 2003). Different approaches in order to compute initial pressure value for the particles were taken in consideration in order to employ the most efficient calculations within the developed piece of software.

$$k \cdot \rho_0 \left(\frac{\rho_i}{\rho_0} - 1 \right)$$

Equation 14: Modified Tait's Equation of State (Yu and Turk 2010).

$$k \cdot (\frac{\rho_i}{\rho_0} - 1)^7$$

Equation 15: Modified Tait's Equation of State (Becher and Teschner 2007).

$$k \cdot (\rho_i - \rho_0)$$

Equation 16: Modified Poisson's Equation of State (Müller et al. 2003).

Once the scalar pressure values have been computed, the resulting pressure force acting on the particles is then calculated according to Monaghan's (1992) equation. This formulation yields a symmetric force due to the arithmetic mean of the particle's pressures.

$$\boldsymbol{F}_i^{pressure} = -\sum_j m_i m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W(\boldsymbol{x}_i - \boldsymbol{x}_j, h)$$

Equation 17: Equation used for Pressure Force calculations in WCSPH algorithm (Monaghan 1992).

4.4 Pressure Derivation and Pressure (PCISPH)

According to the PCI algorithm, the scalar pressure values of the particles are not calculated during the initial phase of the algorithm using an EOS. Instead, scalar pressure is calculated through a corrective process: taking in consideration the predicted density of the particle and the respective predicted density error. The aim is to find a pressure *p* that changes the position of the particles in such way that the predicted density corresponds to the reference density (Solenthaler and Pajarola 2009). Moreover, the predicted density at point in time *t+1* is calculated as exposed in Equation 13 therefore:

$$\rho_i(t+1) = m \sum_j W\left(\boldsymbol{x_i}(t+1) - \boldsymbol{x_j}(t+1), h\right)$$

$$= m \textstyle\sum_j \; W\left(\boldsymbol{x}_i(t) + \Delta\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t) + \Delta\boldsymbol{x}_j(t)\right)$$

$$= m \sum_j W\left(\boldsymbol{d}_{ij}(t)\right) + \Delta\left(\boldsymbol{d}_{ij}(t)\right)$$

Equation 18: Pressure derivation (Solenthaler and Pajarola 2009).

Where $\boldsymbol{d}_{ij}(t) = \boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)$ and $\Delta\boldsymbol{d}_{ij}(t) = \Delta\boldsymbol{x}_i(t) - \Delta\boldsymbol{x}_j(t)$. Considering $\Delta\boldsymbol{d}_{ij}$as relatively small, the first Taylor approximation can be applied to the term $W\left(\boldsymbol{d}_{ij}(t)\right) + \Delta\left(\boldsymbol{d}_{ij}(t)\right)$ resulting in:

$$\rho_i(t+1) = m\sum_j W\left(\boldsymbol{d}_{ij}(t)\right) + \nabla W\left(\boldsymbol{d}_{ij}(t)\right)\cdot\Delta\left(\boldsymbol{d}_{ij}(t)\right)$$

$$= m\sum_j W\left(\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)\right) + m\sum_j \nabla W\left(\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)\right)\cdot\left(\Delta\boldsymbol{x}_i(t) - \Delta\boldsymbol{x}_j(t)\right)$$

$$= \rho_i(t) + \Delta\rho_i(t)$$

Equation 19: Reformulation of predicted density formulation (Solenthaler and Pajarola 2009).

Within this equation the term $\Delta\rho_i(t)$ is unknown. After reformulation and using W_{ij}=W($\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)$) it results in:

$$\Delta\rho_i(t) = m\sum_j \nabla W_{ij}\left(\Delta\boldsymbol{x}_i(t) - \Delta\boldsymbol{x}_j(t)\right)$$

$$= m(\sum_j \nabla W_{ij}\Delta\boldsymbol{x}_i(t) - \sum_j \nabla W_{ij}\Delta\boldsymbol{x}_j(t))$$

$$= m(\Delta\boldsymbol{x}_i(t)\sum_j \nabla W_{ij} - \sum_j \nabla W_{ij}\Delta\boldsymbol{x}_j(t))$$

Equation 20: Difference of densities formulation (Solenthaler and Pajarola 2009).

Where $\Delta\boldsymbol{x}$ can be derived through time integration methods (Semi-Implicit Euler). Neglecting all forces except pressure:

$$\Delta\boldsymbol{x}_i = \Delta t^2 \frac{\boldsymbol{F}_i^p}{m}$$

Equation 21: Difference of position derivation (Solenthaler and Pajarola 2009).

If the simplistic assumption is made that the particles carry equal pressures $\tilde{p}_i$ and that the density corresponds to the rest density ρ_0, this results in:

$$\boldsymbol{F}_{\boldsymbol{i}}^{pressure} = -m^{\mathbf{2}}\sum_{\boldsymbol{j}}(\frac{\tilde{p}_i}{\rho_0^2}+\frac{\tilde{p}_i}{\rho_0^2})\nabla W_{ij} = -m^{\mathbf{2}}(2\frac{\tilde{p}_i}{\rho_0^2})\sum_{\boldsymbol{j}}\nabla W_{\boldsymbol{ij}}$$

Equation 22: PCISPH pressure force formulation (Solenthaler and Pajarola 2009).

Inserting Equation 22 in Equation 21 it results in:

$$\Delta\boldsymbol{x}_i = -\Delta t^2 m(2\frac{\tilde{p}_i}{\rho_0^2})\sum_{\boldsymbol{j}}\nabla W_{\boldsymbol{ij}}$$

Equation 23: Difference of positions reformulation (Solenthaler and Pajarola 2009).

and considering the force as symmetric the following derivation is possible:

$$\boldsymbol{F}_{j|i}^{pressure} = m^{\mathbf{2}}(\frac{\tilde{p}_i}{\rho_0^2}+\frac{\tilde{p}_i}{\rho_0^2})\nabla W_{ij} = m^{\mathbf{2}}(2\frac{\tilde{p}_i}{\rho_0^2})\nabla W_{\boldsymbol{ij}}$$

Equation 24: Pressure force formulation (Solenthaler and Pajarola 2009).

The position of particle *j* will change by:

$$\Delta \boldsymbol{x}_{\boldsymbol{j|i}} = \Delta \boldsymbol{t}^{\boldsymbol{2}} \boldsymbol{m}(\boldsymbol{2}\frac{\tilde{\boldsymbol{p}}_{\boldsymbol{i}}}{\boldsymbol{\rho}_{\boldsymbol{0}}^{\boldsymbol{2}}})\, \boldsymbol{\nabla W}_{\boldsymbol{ij}}$$

Equation 25: Rate of change of particle j positions (Solenthaler and Pajarola 2009).

Inserting Equation 23 in Equation 25 and considering $\Delta \boldsymbol{x}_{j|i} = \Delta \boldsymbol{x}_i$will result in:

$$\Delta\rho_i(t) = m(-\Delta t^2 m\left(2\frac{\tilde{p}_i}{\rho_0^2}\right)\sum_{\boldsymbol{j}} \nabla W_{\boldsymbol{ij}} \cdot \sum_{\boldsymbol{j}} \nabla W_{\boldsymbol{ij}} - \sum_{\boldsymbol{j}} (\nabla W_{\boldsymbol{ij}} \cdot \Delta t^2 m(2\frac{\tilde{p}_i}{\rho_0^2})\, \nabla W_{\boldsymbol{ij}}$$

$$\Delta t^2 m(2\frac{\tilde{p}_i}{\rho_0^2})(-\sum_j \nabla W_{ij} \cdot \sum_j \nabla W_{ij} - \sum_j (\nabla W_{ij} \cdot \nabla W_{ij}))$$

Equation 26: Final difference of densities reformulation (Solenthaler and Pajarola 2009).

After solving for $\tilde{p}_i$:

$$\tilde{p}_i = \frac{\Delta\rho_i(t)}{\beta(-\sum_j \nabla W_{ij} \cdot \sum_j \nabla W_{ij} - \sum_j (\nabla W_{ij} \cdot \nabla W_{ij}))}$$

Equation 27: Corrective pressure formulation in terms of density variation (Solenthaler and Pajarola 2009).

where β is corresponds to

$$\beta = \Delta t^2 m^2 \frac{2}{\rho_0^2}$$

Equation 28: Beta value formulation (Solenthaler and Pajarola 2009).

According to Equation 27 a pressure $\tilde{p}_i$ is required to achieve a change in density $\Delta\rho_i(t)$. Taking in consideration Equation 29 and by reversing the error, Equation 30 is formulated.

$$\rho^*_{err_i} = \rho^*_i - \rho_0$$

Equation 29: Predicted Density Error formula (Solenthaler and Pajarola 2009).

$$\tilde{p}_i = \frac{-\rho^*_{err_i}}{\beta(-\sum_j \nabla W_{ij} \cdot \sum_j \nabla W_{ij} - \sum_j (\nabla W_{ij} \cdot \nabla W_{ij}))}$$

Equation 30: Corrective pressure equation (Solenthaler and Pajarola 2009).

Still referring to Solenthaler and Pajarola (2009), Equation 30 does manifest issues when particles suffer from neighborhood deficiency giving falsified values. Therefore, the formula has been modified by inserting a precomputed single scaling factor δ. Resulting in:

$$\delta = \frac{-1}{\beta(-\sum_j \nabla W_{ij} \cdot \sum_j \nabla W_{ij} - \sum_j (\nabla W_{ij} \cdot \nabla W_{ij}))}$$

Equation 31: Single scaling factor (delta) value (Solenthaler and Pajarola 2009).

And:

$$\tilde{p}_i = \delta\rho^*_{err_i}$$

Equation 32: Resulting equation for corrective pressure calculations (Solenthaler and Pajarola 2009).

Since the prediction-correction step is repeated, the corrective pressure is summed at every predictive-corrective iteration (Solenthaler and Pajarola 2009).

$$p_i {+}{=}\ \tilde{p}_i$$

Equation 33: Pressure update equation (Solenthaler and Pajarola 2009).

4.5 Viscosity

Viscosity is used to specify the behaviour and propriety of a fluid in opposing resistance in flowing. Moreover, this propriety can be quantified, and fluids can be differentiated accordingly: Newtonian fluids do have a linear relationship between shear stress and shear rate. On the other hand, Non-Newtonian fluids do not have constant viscosity and therefore the relationship between shear stress and shear rate is not linear.

For both solver algorithms, viscosity calculations have been done according to the solution proposed by Müller et al. (2003). Furthermore, applying the SPH rule for the $\mu\nabla^2\mathbf{v}$ term of Equation 9 will result in:

$$\boldsymbol{F}_{\boldsymbol{i}}^{viscosity} = \mu\nabla^2\boldsymbol{v}(\boldsymbol{r_a}) = \mu\sum_{\boldsymbol{j}} m_j \frac{\boldsymbol{v}_j}{\rho_j}\nabla W_{ij}$$

Equation 34: Viscosity classic SPH formulation (Monaghan 1988).

However, the resulting force is asymmetric due to the velocity field changes from particle to particle. Therefore, the force is symmetrized by using velocity differences, resulting in:

$$\boldsymbol{F}_{\boldsymbol{i}}^{viscosity} = \mu\sum_{\boldsymbol{j}} m_j \frac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_j}\nabla W_{ij}$$

Equation 35: Viscosity force formulation according to Müller et al. (2003).

4.6 Surface Tension

Molecules in fluids do attract each other (Morris 2000). Furthermore, these intramolecular attractive forces are equal in all direction and balance each other. However, the same forces acting on particles disposed on the surface of the fluid are unbalanced (Müller et al. 2003). Moreover, the tension forces have the direction of the surface normal and act toward the fluid. Additionally, surface tension forces do tend to minimize the curvature of the surface. A tension coefficient will contribute in determining the intensity of the fluid's surface tension force.

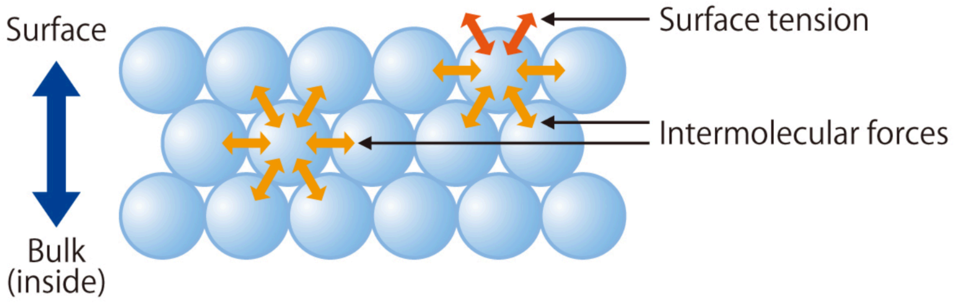


Figure 5: Surface Tension (Kyowa Ltd. 2018).

The surface of a fluids is found by employing a field quantity which correspond to 1 at particle locations and 0 everywhere else (Müller et al. 2003). In literature, this field quantity is identified as color field. Applying the SPH rule will result in:

$$c_S(\boldsymbol{i}) = \sum_j m_j \frac{1}{\rho_j} W_{ij}$$

Equation 36: Color field fomulation according to SPH approach (Müller et al 2003).

The gradient field of the smoothed color field is equal to the surface normal pointing into the fluid.

$$\mathbf{n} = \nabla c_S$$

Equation 37: Gradient of the smoothed color field yielding the surface normal (Müller et al. 2003).

Moreover, the divergence of **n** measures the curvature of the fluid's surface (Müller et al 2003).

$$\kappa = \frac{-\nabla^2 c_S}{|\mathbf{n}|}$$

Equation 38: Curvature formulation according to Müller et al. (2003).

Consequentially, the surface tension force acting near the surface will result in:

$$\boldsymbol{F}_i^{sTension} = \sigma\kappa\mathbf{n} = -\sigma\nabla^2 c_S \frac{\mathbf{n}}{|\mathbf{n}|}$$

Equation 39: Surface Tension Equation (Müller et al. 2003).

Where σ is the surface tension coefficient of the fluid. Additionally, the force is only evaluated if |**n**| exceeds a set threshold due to the numerical problems caused in computing **n**/|**n**|.

Alternatively, another method to simulate surface tension has been developed for this project's implementation. Furthermore, this approach is based on reconstructing the surface tension as a sum of cohesion and curvature forces. However, in this piece of software the adhesion calculations are not included. Cohesion force is calculated as follows:

$$\boldsymbol{F}_{i\leftarrow j}^{cohesion} = -\sigma m_i m_j C(|\boldsymbol{x}_i - \boldsymbol{x}_j|)\frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|}$$

Equation 40: Cohesion force formulation (Akinci et al 2013).

Where *C* is a 3D spline function with the following conditions:

$$C(r) = \frac{32}{\pi h^9} \begin{cases} (h-r)^3 r^3 & 2r > h \wedge r \leq h \\ 2(h-r)^3 r^3 - \frac{h^6}{64} & r > 0 \wedge 2r \leq h \\ 0 & otherwise \end{cases}$$

Equation 41: 3D Spline function created by Akinci et al. (2013).

Where *r* is the distance between the pair of particles, *h* is the smoothing length and h^9 being a normalization factor.

The smoothed gradient of the color field in this approach is calculated as

$$\mathbf{n}_{\boldsymbol{i}} = h \sum_{\boldsymbol{j}} \frac{m_j}{\rho_j} \nabla W_{ij}$$

Equation 42: Gradient of the smoothed color field according to Akinci el al. (2013).

With the *h* factor in order to achieve a computed normal which is scale independent.

Successively, the force acting on the curvature of the fluid's surface is calculated as

$$\boldsymbol{F}_{i \leftarrow j}^{curvature} = -\sigma m_i (\boldsymbol{n}_i - \boldsymbol{n}_j)$$

Equation 43: Curvature force formulation (Akinci et al. 2013).

According to this method a symmetrized correction factor is computed:

$$K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j}$$

Equation 44: Symmetrized correction factor (Akinci et al. 2013).

Then, it is multiplied by the sum of the curvature and the cohesion forces:

$$\boldsymbol{F}_i^{sTension} = K_{ij}(\boldsymbol{F}_{i\leftarrow j}^{curvature} + \boldsymbol{F}_{i\leftarrow j}^{cohesion})$$

Equation 45: Surface Tension formulation as sum of curvature and cohesion force.

4.7 Stress

For Non-Newtonian fluids, the stress tensor is a nonlinear function of the velocity gradient (Mao and Yang, 2006). According to literature, a common method to calculate the stress tensor is to integrate the tensor-rate as in Gotekin et al. (2004) and in Owens and Phillips (2002). Both of these methods are based on a nonlinear co-rotational Maxwell model (Ellero et al. 2002).

$$\frac{dT}{dt} = \Omega + \mu_e D' - \frac{1}{\lambda} T$$

Equation 46: Tensor rate description according to corotational Maxwell model (Ellero et al. 2002).

Where T is the stress Tensor, μ_e the shear modulus or elasticity, λ the relaxation time or intensity of the fluid's behaviour in returning at its previous shape and Ω being equal to:

$$\Omega = \frac{1}{2}(T \cdot \omega - \omega \cdot T)$$

Equation 47: Rotation tensor formulation (Mao and Yang 2006).

Furthermore, Ω, the rotational tensor, can be intended as a coordinate transformation change between the inertial frame and the frame rotating with the fluid's angular velocity. Moreover, ω will result in:

$$\omega = \nabla v - (\nabla v)^T$$

Equation 48: Difference of gradients of velocity vector and it's transpose (Mao and Yang 2006).

where $\alpha\beta$ are 3D spatial coordinate therefore each element of $\omega^{\alpha\beta}$ corresponds to:

$$\omega^{\alpha\beta} = \frac{\partial v^{\beta}}{\partial r^{\alpha}} - \frac{\partial v^{\alpha}}{\partial r^{\beta}}$$

Equation 49: Formulation for calculating every element of ω (Mao and Yang 2006).

D' is the traceless strain tensor and is equal to:

$$D' = \frac{1}{2} D - \frac{Trace(D)}{3} I$$

Equation 50: Equation for the traceless strain tensor according to Mao and Yang (2006).

With:

$$D = \nabla v + (\nabla v)^{T}$$

Equation 51: Formulation for the strain-rate tensor (Mao and Yang 2006).

where each element of $D^{\alpha\beta}$ is equal to:

$$D^{\alpha\beta} = \frac{\partial v^{\beta}}{\partial r^{\alpha}} + \frac{\partial v^{\alpha}}{\partial r^{\beta}}$$

Equation 52: Formulation for calculating every component of strain-rate tensor D (Mao and Yang 2006).

Consequentially, the partial velocity derivative on every particle according to SPH formulism corresponds to:

$$\frac{\partial v^{a}}{\partial r^{b}} = \sum_{j} \frac{m_j}{\rho_j} (v_j^{a} - v^{a}) \frac{\partial W(r - r_j, h)}{\partial r^{b}}$$

Equation 53: Partial velocity derivative calculation in SPH formulism (Mao and Yang 2006).

Therefore:

$$\nabla v_i = \sum_j \frac{m_j}{\rho_j}\left(v_j^a - v^a\right) \otimes \nabla W_{ij}$$

Equation 54: Formulation of velocity gradient (Paiva et al. 2006).

Once the stress tensor has been computed for every particle, the stress term present in Equation 10 can be formulated as:

$$\frac{1}{\rho_i} \nabla \cdot T_i = \sum_j \frac{m_j}{\rho_i \rho_j}\left(\boldsymbol{S}_i + \boldsymbol{S}_j\right) \nabla W_{ij}$$

Equation 55: Calculations for the stress term (Paiva et al. 2006).

This approach has been developed in such way in order to be temperature independent (Erktin 2015).

4.8 Kernels

Smoothing kernels are part of the heart of SPH methods as explained in section 3.2. Accordingly, different kernels are employed for the calculations of the different fluid quantities. Specifically, the kernel functions take two parameters: the distance between the neighbouring particles $\boldsymbol{r}$ and the smoothing length h. The smoothing length is a user set parameter and it determines the results of kernel functions values and neighbourhood determination.

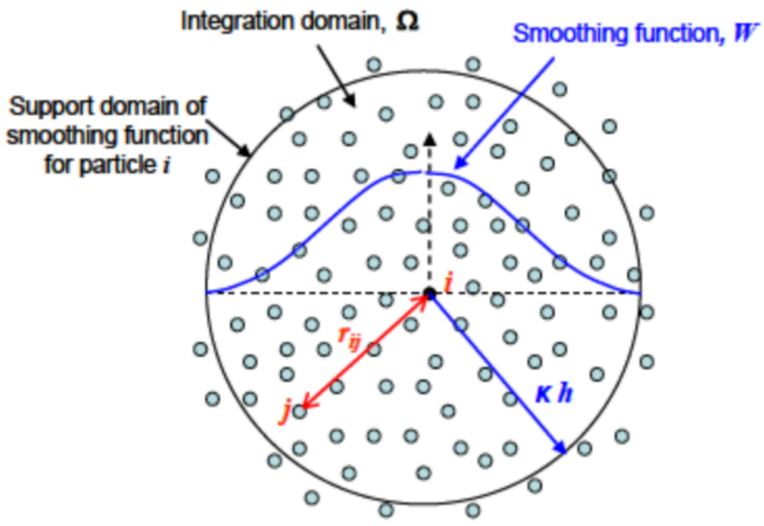


Figure 6: Example of smoothing kernel function (Acin 2018).

For density calculations, a classical sixth order polynomial kernel has been used (Müller et al. 2003).

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |r|^2)^3 & 0 \leq |\boldsymbol{r}| \leq h \\ 0 & otherwise \end{cases}$$

Equation 56: Sixth order Polynomial Kernel (Müller et al. 2003).

The gradient of the sixth order polynomial kernel has been used instead for surface normal calculations and for the pressure predictive-corrective step of the PCI solver:

$$\nabla W_{poly6}(\boldsymbol{r},h) = -\frac{945}{32\pi h^9}\begin{cases}\boldsymbol{r}(h^2-|\boldsymbol{r}|^2)^2 & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise\end{cases}$$

Equation 57: Gradient of the sixth order Polynomial kernel (Müller et al. 2003).

The Laplacian of this function is used for calculating surface tension force according to Müller et al. (2003).

$$\nabla^2 W_{poly6}(\boldsymbol{r},h) = -\frac{945}{32\pi h^9}\begin{cases}(h^2-|\boldsymbol{r}|^2)\;\;(3h^2-7|\boldsymbol{r}|^2) & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise\end{cases}$$

Equation 58: Laplacian value of the sixth order polynomial function (Müller et al. 2003).

Instead, for cohesion forces calculations the 3D Spline function described in Equation 33 is used. Furthermore, for the surface normal calculations it was required to compute the curvature force, using the gradient of the sixth order polynomial kernel.

For calculating pressure forces in the WCSPH solver, a Spiky kernel based on the work of Desbrun and Gascuel (1996) was developed and used in this implementation. Specifically, since this function no longer tends toward zero, it contributes in solving particle cluster problems.

$$\nabla W_{spiky}(\boldsymbol{r},h) = -\frac{45}{\pi h^6}\begin{cases}\frac{\boldsymbol{r}}{|\boldsymbol{r}|}(h-|\boldsymbol{r}|)^2 & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise\end{cases}$$

Equation 59: Gradient Spiky kernel (Desbrun and Gascuel 1996).

According to the literature, a kernel of which the Laplacian value is always greater than or equal to zero is required for achieving correct viscosity calculations. Therefore, the viscosity kernel function described by Müller et al. (2003) has been implemented and used.

$$\nabla^2 W_{viscoLapla}(\boldsymbol{r}, h) = \frac{45}{\pi h^6} \begin{cases} (h - |\boldsymbol{r}|) & 0 \leq |\boldsymbol{r}| \leq h \\ 0 & otherwise \end{cases}$$

Equation 60: Viscosity kernel (Müller et al. 2003).

The gradient kernel value implemented for the stress tensor calculations of this implementation is analogous to the function described by Monaghan's Spline kernel (1992) or Müller et al. polynomial kernel (2003). Mao's and Yang's gradient of the spline kernel was also implemented, but did not give functional results (Erktin 2015).

$$\nabla W_{spline}(\boldsymbol{r}, h) = -\frac{9}{4\pi h^5} \begin{cases} \left(\frac{|\boldsymbol{r}|}{h} - 1.33\right)\boldsymbol{r} & 0 \leq \frac{|\boldsymbol{r}|}{h} \leq 1 \\ -\frac{1}{3}(2 - \frac{|\boldsymbol{r}|}{h})^2 \frac{h}{|\boldsymbol{r}|}\boldsymbol{r} & 1 \leq \frac{|\boldsymbol{r}|}{h} \leq 2 \\ 0 & otherwise \end{cases}$$

Equation 61: Gradient Spline Kernel (Mao and Yang 2006).

4.8 Integration and Prediction

Once the total forces acting on every particle has been summed, a method to compute the new velocity and position is required. Therefore, within this project's implementation, a Semi Implicit integration technique is also present. The same method is used for the prediction step of the PCI solver.

$$\text{a)} \quad \boldsymbol{v}_i^{t+\Delta t} = \boldsymbol{v}_i^t + \boldsymbol{F}_i^t \cdot \Delta t$$

$$\text{b)} \quad \boldsymbol{x}_i^{t+\Delta t} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+\Delta t} \cdot \Delta t$$

Equation 62: Semi-Implicit Euler Integration Scheme a) Velocity b) Position (Kelager 2006).

4.9 XSPH

A correction method was introduced in 1989 by Monaghan in order to prevent particle penetration by adjusting the velocity of the particles. The correction is performed after the integration step and is defined by Monaghan (1989) as:

$$\widehat{\boldsymbol{v}_i} = \boldsymbol{v}_i + \varepsilon \sum_j m_j [\frac{\boldsymbol{v}_j - \boldsymbol{v}_i}{(\rho_i + \rho_j)/2}] W_{ij}$$

Equation 63: XSPH velocity correction formulation (Monaghan 1989).

where ε is $0 \leq \varepsilon \leq 1$ and W_{ij} is the sixth polynomial kernel.

4.10 Neighbor Search (Spatial Hashing)

Neighbourhood determination of a particle is a fundamental process within an SPH based solver. Furthermore, basic neighbour searching algorithms generally have high complexity, which become even more inefficient with bigger particles count. With an efficient neighbour approximation technique in an SPH solver, it is possible to achieve a substantial increase in terms of performance and computation time. Therefore, a Fast Nearest Neighbour Searching (FNNS) algorithm has been developed for this project's implementation based upon the Standard C++ multi-map container. The algorithm chosen for the current piece of software is the Spatial Hashing Algorithm described by Teschner et al. (2003).

Spatial hashing is a process by which a 3D or 2D space is projected onto a 1D hash-table (Hastings et al. 2005). In order to implement a spatial hashing algorithm for an 3D SPH solver, three features are fundamental: a 3D grid, a hash function and a hash-table. Moreover, the grid is defined by the cell size and two points that set the cell in space. Specifically, the hash function takes positional data and returns a respective grid cell that corresponds to a 1D bucket in the created hash table (Hastings et al. 2005).

During the first step of the algorithm, the position of the considered particle *i* is discretized in with respect to a user-defined cell size *l* (Tescher et al. 2003). Still referring to Kelager's work, the cell size has been set equal to the smoothing length of the kernels (Ihmsen et al. 2011).

According to the discretization process, the coordinates of the particle are divided by the cell size and rounded to the closest smaller integer:

$$(i, j, k): i = \lfloor x/l \rfloor, j = \lfloor y/l \rfloor, j = \lfloor z/l \rfloor$$

Equation 64: Discretization formula (Kelager 2006).

Successively, the hash function maps the obtained discretized positions to a 1D index h, so that the information is store in a hash table with index equal to h: $h = hash(i, j, k)$ (Kelager 2006). The hash function is defined in the current implementation as:

$$hash(x, y, z) = (\, x\, p1 \,\mathbf{xor}\, y\, p2 \,\mathbf{xor}\, z\, p3)\,\mathbf{mod}\, n$$

Equation 65: Hash function (Kelager 2006).

where *p1, p2, p3* are big prime numbers with the respective value of 73856093, 19349663 and 83482791 and *n* corresponds to the hash table size. Furthermore, the size of the hash table is set to be the double of the particle count (Teschner et al. 2003) and is determined according to Kelager's (2006) definition:

$$n = prime(2 \cdot particleCount)$$

Equation 66: Hash table size calculations (Kelager 2006).

where *prime(x)*, $x \in \mathbb{Z}$ is a function which will return the next prime number $\geq x$ (Kelager 2006).

The particle query within this piece of software is based on a bounding box method according to the work of Kelager (2006). The bounding box is defined around the particle taken in consideration according to the cell size and the position of the particle.

$$BB_{min=x_i-(h,h,h)^T}$$

$$BB_{max=x_i+(h,h,h)^T}$$

Equation 67: Calculations for the bounding box determination (Kelager 2006).

Once the bounding box is defined, an iteration over three dimensions is performed so that the possible neighbors can be found (Priscott 2010). The existence of neighbors is controlled through a normal hash map due to the requirement of returning unique results. If the candidate particle is not an existing neighbor, the distance between that particle and the current particle is calculated. If the distance results less than the grid size, then it is stored as a dynamic neighbor on the normal hash map. Specifically, in terms of complexity this algorithm takes *O(n)* (Kelager 2006).

5.0 Results, Issues and Suggestions for further studies

This program's implementation has been designed in order to simulate Newtonian and Non-Newtonian fluids according to two different Smoothed Particle Hydrodynamics approaches. The achieved results for the diverse solver algorithms are exposed and analyzed in the following section.

As shown in Figure 6 and 7, the Weakly Compressible based solver works correctly, and the output of these solvers is quite realistic. Moreover, the computation time of the WCSPH for Newtonian fluids is acceptable for low particle count. Once the particle count increases, also does the computation time. Specifically, the simulation remains stable with over 100k particles allowing an adequate realistic simulation of fluids.

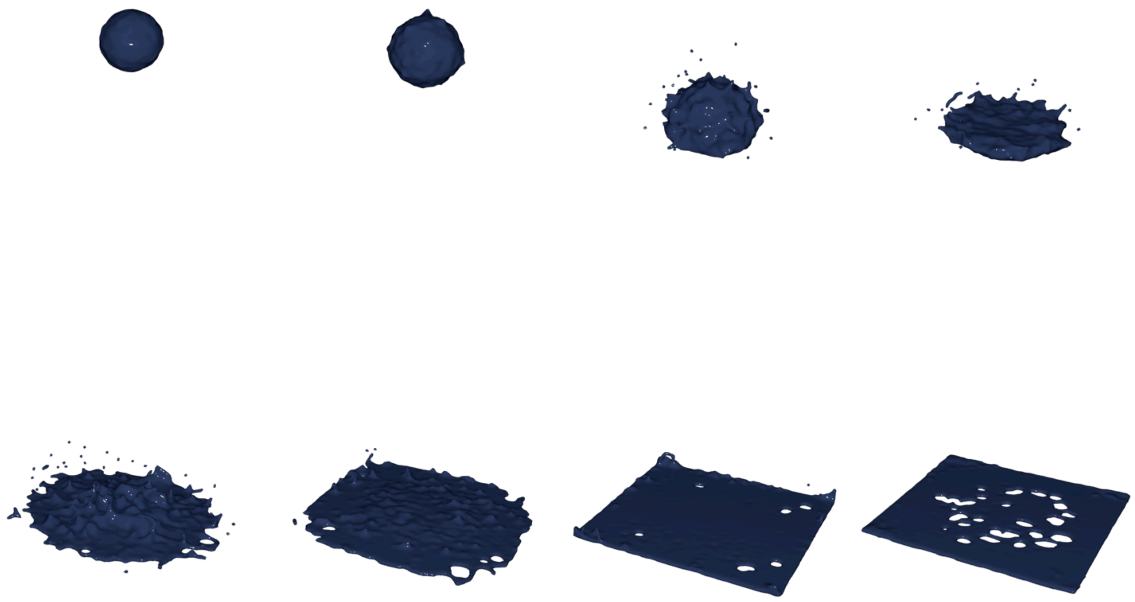


Figure 7: WCSPH solver simulating water with 10k particles (Personal collection).

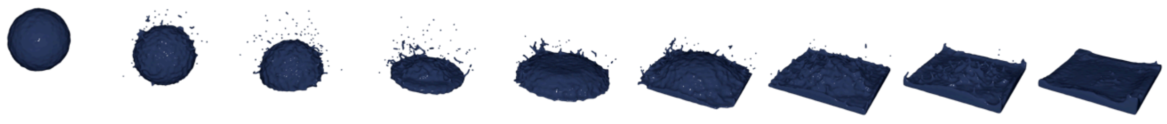


Figure 8: WCSPH simulating water with 100k particles (Personal collection).

The WCSPH Non-Newtonian solver's simulations do manifest viscoelastic behavior (Figure 9). Furthermore, higher smoothing length and smaller time-step are required for this solver to work efficiently. Therefore, the computation time increases drastically when simulating Non-Newtonian fluids. The simulations are stable, usable and the results are quite appealing within a limited number of iterations.

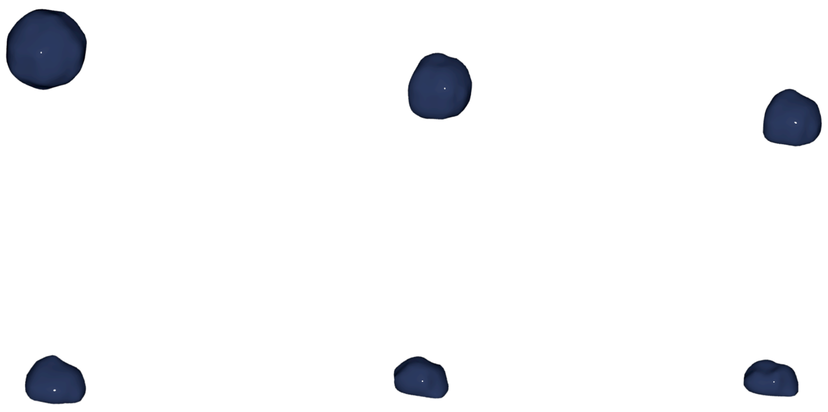


Figure 9:WCSPH Non-Newtonian Fluid with high elasticity and relaxation parameters (Personal collection).

Moreover, it can be confirmed that the possible achievable results according to this method are restricted (Chang et al. 2009). Dropping spheres simulations does give similar results to the reference paper of Mao and Yang (2006). Additionally, according to the obtained analyzed results, the XSPH velocity correction (Monaghan 1989) covers a fundamental role in the Non-Newtonian solvers. In addition, it has been noted that the fluid tends to shrink in order to gain a viscoelastic consistency subject to the computed exceeding stress force and the disharmony between the rest density of the fluid and the particle mass. In some cases, the excessive stress force modifies the fluid flow in an unrealistic way.

The incompressibility of the fluids simulated with the WCSPH algorithms is still an issue. Even though, changing the EOS from a Poisson to a modified Tait's allowed to achieve low level of compressibility particularly in Newtonian Fluids (Figure 8).

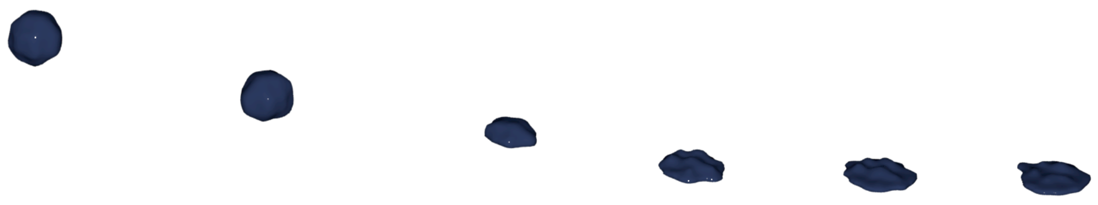


Figure 10: WCSPH Non-Newtonian fluid with low elasticity and relaxation parameters (Personal collection).

The PCI algorithm-based solver for both types of fluids is not complete. This algorithm was developed in order to deal with incompressibility of the fluids. Still, the incompressibility of fluids was not achieved. In particular, only the pressure calculations are working and therefore included in the implementation, generating simulations that resemble a fluid, but cannot be defined realistic (Figure 11). In addition, viscosity calculations were also included but high viscosity coefficients lead toward an unstable simulation. Similarly, for surface tension, two different approaches were implemented in order to give the fluid a more realistic and appealing aesthetics.

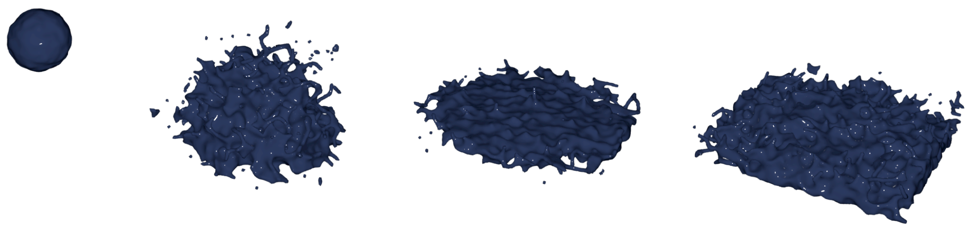


Figure 11: PCISPH simulating water with 10k particles (Personal collection).

Nevertheless, both approaches did not give the expected results. Therefore, viscosity and surface tension calculations are included in the implementation only as proof of concept (Figure 12). Furthermore, the resulting density prediction calculations have been overestimated in order to achieve a density variation that would allow a scalar pressure value to enable the fluid to flow.

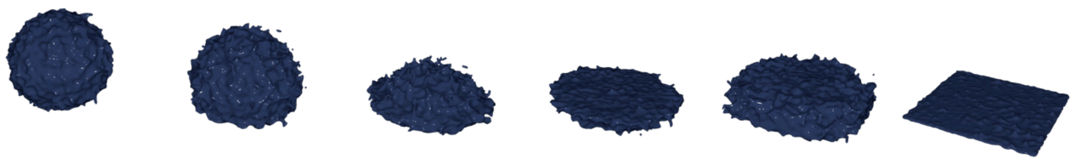


Figure 12: PCISPH simulating water with 100k particles (Personal collection).

However, in terms of computational time, the PCISPH solver for Newtonian Fluids is slightly more efficient than the WCSPH, confirming that further refinements and improvements could be significant in achieving an overall better solver. Simulations performed with this algorithm are stable with over 100k particles.

Regarding the PCI solver for Non-Newtonian fluids, the achieved results do present some viscoelastic behavior (Figure 13). Yet, the solver is not functional nor efficient. Furthermore, the volume is not preserved correctly especially with low elasticity and relaxation parameters. Moreover, computation time is way too high due to the high smoothing length and small time-step required for the simulator to work. Additionally, a supplementary neighbor search is performed before calculating the pressure force, in order to achieve the obtained results, which causes a drastic decrease in terms of performance.

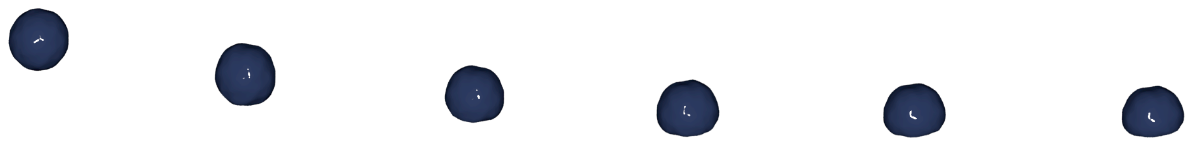


Figure 13: PCISPH Non-Newtonian Fluid demonstrating some viscoelastic behaviour with extremely high elasticity and relaxation parameters (Personal collection).

After profiling with Instruments the current piece of software several times, it was found that the main issue for all solvers was neighbor approximation. Specifically, in the case of the PCI based solvers which suffers more from this type of problem, there was deficiency in the neighborhood determination which resulted in falsified values (Solenthaler and Pajarola 2009).

This had to be taken in consideration when performing calculations for the resulting forces acting on the fluid’s particles, including pressure and stress. Accordingly, an improvement in the neighbor search algorithm would result in an increase in terms of performance for all solvers due to the high influence of the particle count on the FNNS algorithm. Therefore, different neighborhood determination techniques could be implemented and tested to circumvent this problem such as a dynamic Verlet list or a plane sweep algorithm.

6.0 Conclusion and Future Work

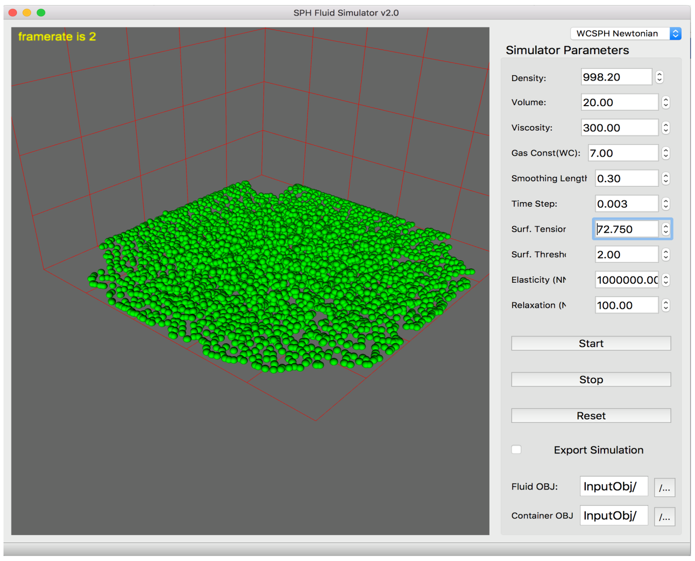
The current implementation presents a Weakly Compressible and Predictive-Corrective based SPH solvers for Newtonian and Non-Newtonian fluids. The WCSPH solver for Newtonian fluids can be defined complete of the fundamental basic SPH simulator features. On the other hand, the Non-Newtonian WC solver does require some optimization in terms of performance in order to be defined completely functional. However, appealing results can be achieved through the use of this algorithm. Instead, the PCI based solver for Newtonian fluids presents different issues and cannot be defined complete of SPH fundamental features due to the absence of viscosity and surface tension calculations. Moreover, the PCI Non-Newtonian solver cannot be defined neither functional nor efficient due to the high computation time required to successfully simulate fluids.

Different improvements could be made in the future for all solvers starting from a more efficient and accurate neighbor determination algorithm. Specifically, this improvement could result in a significant increase of efficiency and accuracy in the calculations for the PCI based solvers which suffer from neighborhood deficiency (Solenthaler and Pajarola 2009). Furthermore, adaptive time-stepping could be a solution in order to achieve an increase in performance (Akinci 2010). In addition, in order to prevent particle scattering due to sharp velocity field variations, Ghost Particles SPH methods (Schechter and Bridson 2012) could be included in the future.

In order to deal with incompressibility of the fluid, the Position Based Fluid method proposed by Macklin and Müller (2013) could offer instead a more optimal solution. Additionally, an arbitrary object boundary and collision technique could also be included, increasing as a consequence the number of different possible scenarios creatable from the simulator.

For Non-Newtonian fluid solvers, an optimization in terms of computation time would allow the respective solver to be defined efficient. Additionally, a dependency from temperature can be included as proposed (Paiva et al. 2006). Therefore, allowing more control over the elasticity and viscosity of the fluid over time. Furthermore, different approaches in describing Non-Newtonian fluids flow could have probably resulted more efficient and might be tested in the future (Peer et al. 2015).

Finally, a parser could be included within this project's implementation so that fluids parameters could be loaded to the simulator UI from a text file. This would allow a faster generation of different simulations and scenarios. In addition, a functionality that would allow the user to save the fluid parameters of an appealing simulation onto a properly formatted text file, could also be implemented.



WCSPH Newtonian

Simulator Parameters

Density:	998.20
Volume:	20.00
Viscosity:	300.00
Gas Const(WC):	7.00
Smoothing Lengtl	0.30
Time Step:	0.003
Surf. Tensior	72.750
Surf. Thresh	2.00
Elasticity (NI	1000000.0C
Relaxation (N	100.00

☐ Export Simulation

Fluid OBJ: InputObj/ /...

Container OBJ InputObj/ /...

Figure 14: Frame from the simulation process (Personal collection).

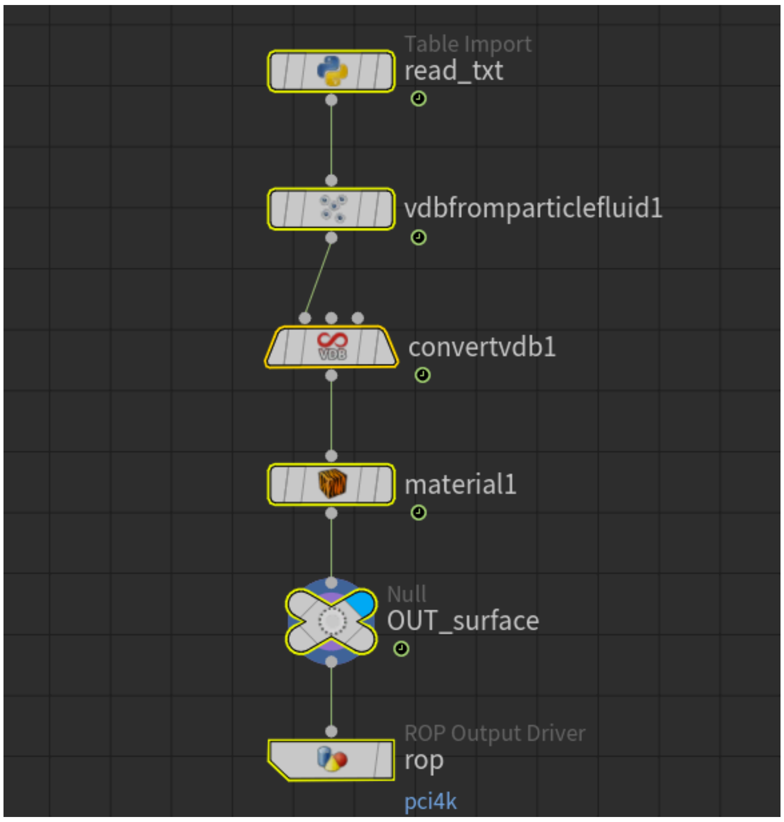


Figure 15: SideFx's Houdini Network used to render out the simulation frames (Personal collection).

References

[1] Akinci, N. *Boudary Handling and Adaptive Time-Stepping for Incompressible SPH*. Thesis (MSc). University of Freiburg.

[2] Akinci, N., Akinci, G. and Teschner, M., 2013. Versatile Surface Tension and Adhesion for SPH Fluids. *ACM Transaction on Graphics*. 32 (6), 1-8.

[3] Becker, M. and Teschner, M., 2007. Weakly compressible SPH for free surface flows. *SCA '07 Proceedings of the 2007 ACM SIGGRAPH Eurographics symposium on Computer animation*. 2017, 209–217.

[4] Brackbill, J. U. and Ruppel, H. M., 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65 (2), 314–343.

[5] Chang, Y., Bao, K., Lju, Y, Zhu, J. and Wu, E., 2009. A particle-based method for viscoelastic fluids animation. *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*. VRST '09, 111-117.

[6] Desbrun, M. and Gascuel, M. P., 1995. Animating soft substances with implicit surfaces. *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95 (44), 287–290.

[7] Desbrun, M. and Gascuel, M. P., 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*. 61–76.

[8] Ellero, M., Kroger, M. and Hess, S., 2002. Viscoelastic flows studied by smoothed particle dynamics. *Journal of Non-Newtonian Fluid Mechanics*. 105, 35–51.

[9] Ertekin, B., 2015. *Fluid Simulation Using Smoothed Particle Hydrodynamics*. Thesis (MSc). Bournemouth University.

[10] Evans, M. and Harlow, F., 1957. *The particle-in-cell method for hydrodynamics calculations*. Los Alamos, New Mexico: The University of California, Los Alamos Scientific Laboratory.

[11] Foster, N. and Fedkiw, R., 2001. Practical animation of liquids. *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01, 23–30.

[12] Foster, N. and Metaxas, D., 1996. Realistic animation of liquids. *Graphical Models and Image Processing*. 58(5), 471 – 483.

[13] Fxguide, 2011. *The science of Fluid Sims* [online]. California, USA: Fxguide. Available from: https://www.fxguide.com/featured/the-science-of-fluid-sims/ [Accessed 10 May 2018].

[14] Gingold, R. A. and Monaghan, J. J., 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*. 181 (3), 375–389.

[15] Goktekin T. G., Bargteil A. W. and O'Brien J. F., 2004. A method for animating viscoelastic fluids. *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04, 463–468.

[16] Harlow, F. H., 1962. *Computer Physics Communications*. Los Alamos, New Mexico: The University of California, Los Alamos Scientific Laboratory.

[17] Hastings, E. J., Mesit, J. and Guha, R. K., 2005. *Optimization of Large-Scale, real-Time Simulations by Spatial Hashing*. Orlando Florida: University of Central Florida.

[18] He, X., Liu, N., Li, S., Wang, H. and Wang, G., 2012. Local poisson sph for viscous incompressible fluids. *Computer Graphics Forum*. 31 (6), 1948–1958.

[19] Ihmsen, M., Akinci, N., Becker, M. and Teschner, M., 2011. A parallel SPH Implementation on Multi-Core CPUs. *Computer Graphics Forum*

[20] Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A. and Teschner, M., 2014. *SPH Fluids in Computer Graphics* [online]. Freibourg: The Eurographics Association. Eurographics 2014 - State of the Art Reports.

[21] Jacob, G. and Jacob, B, 2010. *Eigen C++*.

[22] Kelager, M., 2006. *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Thesis (BSc). Department of Computer Science, University of Copenhagen.

[23] Lucy, L. B., 1977. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*. 82 (12), 1013–1024.

[24] Mao, H. and Yang, Y., 2006. *Particle-Based Non-Newtonian Fluid Animation with Heating Effects*. Alberta: University of Alberta.

[25] Mashayekhi, O., Shah, C., Qu, H., Lim, A. and Levis, P., 2018. Automatically Distributing Eulerian and Hybrid Fluid Simulations in the Cloud. *ACM Transactions on Graphics*. 37 (2), 24.

[26] Miller, G. and Pearce, A., 1989. Globular dynamics: A connected particle system for animating viscous fluids. *Computer and Graphics*. 12 (3), 305-309.

[27] Monaghan, J. J., 1992. Smoothed particle Hydrodynamics. *Annual Review of Astronomy and Astrophysics*. 30 (1), 543–574.

[28] Morris, J., P., 2000. Simulating surface tension with smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*. 33, 333-353.

[29] Müller, M., Charypar, D. and Gross, M., 2003. Particle-Based Fluid Simulation for Interactive Applications. *SCA '03 Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 154–159.

[30] Owens, R. G. and Phillips, T. N., 2002. Computational Rheology. United Kingdom: Imperial College Press.

[31] Peer, A., Ihmsen, M., Cornelis, J. and Teschner, M., 2015. An implicit viscosity formulation for SPH fluids. *ACM Transactions on Graphics(TOG), 34 (4), 114*.

[32] Priscott, C., 2010. *3D Lagrangian Fluid Solver using SPH approximations*. Thesis (MSc). Bournemouth University.

[33] Raveendran, K., Wojtan, C. and Turk, G., 2011. Hybrid smoothed particle hydrodynamics. *Proceedings of the 2011 ACM SIG- GRAPH/Eurographics Symposium on Computer Animation*. SCA '11, 33–42.

[34] Reeves, W.T., 1983. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects [online]. *ACM Transactions on Graphics (TOG)*. 2 (2), 91-108.

[35] Schechter, H. and Bridson. R., 2012. Ghost SPH for animating water. *Proceedings of Transaction on Graphics*.

[36] Solenthaler, B. and Pajarola, R., 2009. Predictive-corrective incompressible SPH. *ACM Transactions on Graphics*. 28, 3.

[37] Solenthaler, B., Schilafli, J. and Pajarola, R., 2007. A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*. 15 (34), 183-192.

[38] Stam, J., 1999. Stable fluids. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques SIGGRAPH '99*. 121–128.

[39] Stora, D., Agliati, P., O., Paule Cani, M., Neyeret, F. and Gascuel, J. 1999. Animating lava flows. *Graphics Interface*. 203-210.

[40] Takahashi, T., Yoshinori, D., Tomoyuki, N., and Ming C., L., 2018. An Efficient Hybrid Incompressible SPH Solver with Interface handling for Boundary Conditions. *Computer Graphics Forum*. 1-13.

[41] Terzopoulos, D., Platt, J. and Fleischer, K., 1991. Heating and melting deformable models. *The journal of Visualisation and Computer Animation. 2, 2.*

[42] Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D. and Gross, M., 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Proceedings of Vision, Modeling, Visualization*. VMV'03. 47–54.

[43] Wu, K., Truong, N., Yuksel, C. and Hoetzlein, R., 2018. Fast Fluid Simulation with Sparse Volumes on the GPU. In *Proceedeings of Eurographics '18*. 37 (2), 157-167.

[44] Zhang, R., He, X., Shiyi, C., 2000. Interface and surface tension in incompressible lattice Boltzman multiphase model. *Computer Physics Communications*. 129, 121-130.