Institute of Neuroinformatics
University of Zurich and ETH Zurich

# A Smoothed Particly Hydrodynamics Approach to the Interactive Swarm Orchestra

Supervised by
Daniel Bisig from the
Artificial Intelligence Lab, University of Zurich

Engin Bumbacher
03-912-862
Im Loorain 14
8803 Rueschlikon

The Interactive Swarm Orchestra is a library built for both artistical and scientific purposes. The aim of this project was to implement fluid dynamics by use of Smoothed Particle Hydrodynamics as a behavior of the swarm, based on the principle of swarm intelligence. The reason why the particle-based approach has been chosen, and not another method based on grid computation, is that it fits in well with the infrastructure of the ISO a swarm consists of individual agents which themselves can be seen as the particles required for the particle approximation of the fluid dynamical forces. The main condition the implementation has to meet is the claim for real-time processing, as interactivity and intelligent swarm behavior are the fundamental paradigms of the ISO project. Unfortunately, it turnes out that the ISO framework does have some disadvantages for the SPH implementation when considering those requirements, as they only can be fulfilled when the number of used particles is rather low. But then again, this small number of particles clearly reduces the accuracy of the method. While this problem was not significant in the case of free surface fluids without any boundary, it became relevant in the case of boundary conditions which I have implemented by means of virtual particles. Thus, this approach to the boundary conditions was not fruitful, which is why I have suggested a more promising approach using wall weight functions.

## Contents

# 1 Introduction

## 1.1 Interactive Swarm Orchestra

The Interactive Swarm Orchestra is a large library, consisting mainly of a generic swarm simulation library (ISO Flock) which a sound synthesis library (ISO Synth) has been added to [3].

A swarm describes an assembly of individuals or agents, having certain properties such as mass, size, etc. in common, whose local interactions give rise to highly complex emergent global behaviors. Thus, the emergent behavior pattern can be seen as the collective behavior of decentralized, self-organized systems[4]. Natural examples of swarms encompass ant colonies, fish schools and animal flocks, but also neural networks, among others. Due to the complexity of the behavior, one also talks of swarm intelligence, which led to algorithmic approaches to artificial swarms with applications in several different areas, ranging from robotics to economics.

The purpose of ISO was to design a framework exploiting the interactivity and emerging complexity of such swarms for artistic objectives, such as visualisation at runtime or control of other ISO tools like ISO Synth. To this end, the concepts of swarm and swarm behavior have been abstracted as follows.

### 1.1.1 Terminology [3]

**Swarm** A swarm is a collection of entities called agents with a constant basic structure of parameters and behaviors.

**Agent** Agents do not exist in space as such. They rather have to be seen as labeled containers for parameters and behaviors. Interaction among agents, between agents and swarms or among swarms is happening on the level of parameters and regulated via behaviors.

**Parameter** Parameters are labeled vectors of arbitrary dimension and manage the relationship (euclidian distance and direction) with other parameters, representing other agents, organized in neighbor groups of the corresponding spaces. A parameter can exist in several spaces at the same time, interacting differently with other parameters depending on the given properties of the specific space.

**Behavior** A behavior describes the functional relationship among parameters. They normally require the definition of input, internal and output parameters, and the neighbor groups of relevant parameters. These interactions as such vary the values of the parameters without changing their structural properties.

**Neighborgroup** Neighbor groups depend on the definition of the relationship among parameters and on the spaces the parameters are existing in. They incorporate the concept of local interaction.

**Space** Spaces define the environment the parameters are existing in and regulate the calculation of the neighbor groups of the parameters.

### 1.1.2 Structure of the ISO library [3]

This structured approach to swarms manifests itself in the strongly object-oriented architecture of the ISO library. Without going too much into the details: Apart from the frameworks required for the calculations and implementations of the swarms, there are the 4 generic base classes called swarm, agent, behavior and parameter within the ISO flock framework and an entire ISO space framework regulating the spatial properties and neighborhood relations. All specialized spaces and behaviors inherit from these base classes.

The main specialized space classes are the point space class, managing distance calculations among point like spatial objects (i.e. parameters) via a QuadTree, OcTree or higher dimensional space partitioning algorithm, and the shape space class calculating the distances among objects in space that possess a shape (as opposed to point like objects).

An example application transforms the tracked outline of people into a spline that serves as movement guide for agents. For a similar purpose another type of space manages the distribution of vectors on an n-dimensional regular grid. Such grids can serve for example as static or dynamic force fields and propel or slow down agents as they move through space. Another example application updates such a force field based on tracked visitor motion.

Normally, the agents are visually manifested via their position in three dimensional space.

The main notion of the ISO framework is the one of real-time applications. ISO should not be seen as a simulation platform, but rather as an environment in which agents behave and interact on a real timescale.

## 2 Computational Fluid Dynamics

Computational Fluid Dynamics deals with the problem of numerically solving the analytical equations of fluid dynamics. In general, a numerical simulation of a CFD problem involves the following factors [2]:

1. Governing equation
2. Boundary and initial conditions
3. Domain discretization technique
4. Numerical discretization technique
5. Numerical technique to solve the partial differential equations (PDE)

There are two fundamental frames for describing the physical governing equations: the Eulerian and the Lagrangian description. The first one is a spatial description, where the flow is described by specification of the time history of the flow properties at every fixed point of the domain [5] (i.e. a fixed coordinate system is used, with the local time derivative $\frac{\partial}{\partial t}$ as the essential variable), and the latter one a material description, where the flow is described by specification of the physical properties of each material particle as a function of time (i.e. a moving coordinate system, with the total time derivative $\frac{D}{Dt} = \frac{\partial}{\partial t} + v^\alpha \frac{\partial}{\partial v^\alpha}$ as the essential variable).

Concerning the domain discretization technique, there are mainly grid-based methods, which can be subdivided into Eulerian and Lagrangian ones, meshfree methods, particle methods, such as SPH, and combinations of them. Grid-based approaches require mesh-generation for the problem domain, thus consume a significant portion of the computational effort. Meshfree methods on the other hand, are based on the key idea of providing accurate and stable numerical solutions for PDEs with a set of arbitrarily distributed nodes (or particles) without using any mesh connecting these nodes. Finally, particle methods employ a finite number of discrete particles to represent the state of a system and to record the movement of the system. Each particle possesses a set of field variables such as mass, momentum, energy, position, etc. related to the specific problem. The evolution of the physical system is then determined by the conservation of mass, momentum and energy of the particles.

In general, obtaining analytical solutions for a set of such PDEs is not possible, why efforts have been made in seeking for numerical solutions. In doing so, a method is needed to provide an approximation for the values of the field functions and their derivatives at any point. This function approximation is then applied to the PDEs to produce a set of ordinary differential equations (ODE) in a discretized form with respect only to time.

Smoothed Particle Hydrodynamics, as a mesh-free particle-based Lagrangian method, provides such a framework [2].

### 2.0.3 Mathematical Framework of Fluid Dynamics

The behavior of a general fluid dynamical system is governed by the conservation of mass, momentum and energy, and the required equation of state. As this project follows the paper of Mller et al [1], we only look at **incompressible, isothermal fluids of constant viscosity**. Due to the isothermal property, the degree of freedom is reduced by one, why the energy of the system can be completely ignored in this case.

#### Conservation of mass

The Continuity equation is:

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \mathbf{v}^\alpha) = 0, \tag{1}$$

where

$\rho(, \mathbf{x})$ ... Density of the material particle at time $t$ and position $\mathbf{x}$

$\mathbf{v}^\alpha$ ... Velocity of particle $\alpha$.

An incompressible flow is a flow in which the density of each material particle remains the same during the motion:

$$\rho(t, \mathbf{x}(t, \mathbf{y}) = \rho(0, \mathbf{y}) \quad \Rightarrow \quad \frac{D\rho}{Dt} = 0 \quad \Rightarrow \quad \nabla(\mathbf{v}^\alpha) = 0.$$

#### Conservation of momentum

The Navier-Stokes equation for incompressible flow and constant viscosity is:

$$\rho \frac{D\mathbf{v}^\alpha}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{v}^\alpha + \rho \mathbf{f}^b, \tag{2}$$

where $\frac{D}{Dt}$ denotes the total time derivation and

$p$ ... Pressure

$\mu$ ... Viscosity coefficient (constant!)

$\mathbf{f}^b$ ... Body force, acting on a particle, proportional to its mass (such as gravity).

## 2.1 Smoothed Particle Hydrodynamics (SPH)

SPH is a time-discrete interpolation method for particle systems. The fluid in question is represented by a set of discrete (i.e. finite spatial extension) elements called particles, which carry relevant properties such as mass, density, pressure, etc. themselves. At the same time, the particles serve as approximation points. Any physical quantity of any particle or any field quantity at any point in space can be obtained by averaging the relevant properties of all the neighboring particles, weighted according to their distance from the particle or point of interest, and their density. The local neighborhood of a particle is defined as the region within a certain range, referred to as *support domain*. Thus, as the particles carry material properties and are allowed to move in virtue of the internal interactions and external forces, SPH harmonically combines the Lagrangian formulation and particle approximation. [2]

The formulation of SPH is often divided into two key steps. The first step is the *integral representation* or the so-called *kernel approximation* of field functions. The second one is the *particle approximation* [6].

### Integral Representation

The kernel approximation for a field quantity $f(\mathbf{x})$ is

$$< f(\mathbf{x}) >= \int_{\Omega} f(\mathbf{x}')W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}',$$

with

$$< \nabla f(\mathbf{x}) >= \int_{\Omega} f(\mathbf{x}')\nabla W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}',$$

where

$W(\mathbf{x}, h)$ ... Smoothing function or smoothing kernel

h ... Smoothing length defining the influence area of the smoothing function

$\Omega$... Problem domain of the simulation over which is integrated, with $\mathbf{x}' \in \Omega$

and where the support domain is lying within the problem domain.

**Particle approximation**

In this step, the continuous integral representations of the field function and its derivatives are converted to discretized forms of summation. This is done by replacing the infinitesimal volume $d\mathbf{x}'$ by the finite volume $\Delta V_j$ of the particle $j$ that is related to the mass $m_j$ by

$$m_j = \Delta V_j \rho_j,$$

where $\rho_j$ is the density of particle which has to be evaluated at every time step.

The resulting equations are:

$$< f(\mathbf{x}_i) >= \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) W_{ij} \tag{3}$$

$$< \nabla f(\mathbf{x}_i) >= \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) \nabla_i W_{ij}, \tag{4}$$

$$< \Delta f(\mathbf{x}_i) >= \sum_j \frac{m_j}{\rho_j} f(\mathbf{x}_j) \Delta_i W_{ij}, \tag{5}$$

where $\mathbf{x}_i$ is the position of the focal particle $i$ and

$W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h) = W(|\mathbf{x}_i - \mathbf{x}_j|, h)$

$\nabla_i W_{ij} = \dfrac{\partial W_{ij}}{\partial \mathbf{x}_i}$

Thus, the density of particles is a fundamental field quantity underlying the concept of SPH. One of the most popular forms of obtaining density (and also used in this project) in SPH is called the *summation density approach* [6]:

$$\rho_i = \sum_j m_j W_{ij} \tag{6}$$

Accuracy of the density can be increased by normalizing the density $\rho i$ with the factor $\sum_j (\frac{m_j}{\rho_j}) W_{ij}$.

### 2.1.1 Constructing the Kernel Function

The kernel has to be chosen such that:

1. Normalization is fulfilled: $\int W(\mathbf{x}') d\mathbf{x}' = 1$
2. $W(-\mathbf{x}) = W(\mathbf{x})$ for radial symmetry

3. $\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}',h) = \delta(\mathbf{x}-\mathbf{x}') \quad \Rightarrow \quad \lim_{h\to 0} < f(\mathbf{x}) >= f(\mathbf{x})$
4. $W(\mathbf{x}-\mathbf{x}',h) = 0 \quad \text{when} \quad |\mathbf{x}-\mathbf{x}'| > h$
5. $W(\mathbf{x}-\mathbf{x}') \geq 0$ for all $|\mathbf{x}-\mathbf{x}'| \leq h$ to ensure a physically stable representation of physical phenomena
6. Monotonical decay with increasing distance
7. the derivatives vanish at the boundary for stability reasons

Apart from these contraints, one is free to design the kernels for special purposes.

### 2.1.2 Inherent Problems

SPH has some inherent problems concerning the simulation of fluids. First, SPH does not guarantee the physical principle of symmetry. Thus, when applying SPH to physical problems, one has to symmetrize the generated forces, as has been done in [1]. Secondly, the accuracy of the method depends on the interpolation order used. Thirdly, there's the so-called *particle inconsistency problem* [2]: The discretized normalization condition is not always fulfilled in meshfree particle methods. This can be the case for particles on and near the boundary, whose support domains are truncated by the boundary, as well as irregularly distributed particles. In the latter case, there are several possibilities to restore the consistency condition, to obtain a k-th order consistency in discrete form, by setting certain conditions for the kernel function. But these are rather complicated, slow down the computational process and especially conflict with the other above-mentioned conditions the kernel function is subject to. As in this project, physical accuracy is not of the highest priority, I dispense with such means. For detailed explanation see [2]. For particles at the boundary, I have implemented the concept of *Virtual Particles*.

### 2.1.3 Boundary Treatment

The above-mentioned particle inconsistency problem appears near or on the boundaries because only particles inside the boundary contribute to the summation of the agent interaction, reducing not only the velocity but also other field variables such as the density to zero, which does not give the correct solutions. This problem can be met by creating additional artificial particles on or outside the boundaries, which contribute to the particle approximations in a proper way. Such particles are called *Virtual Particles* [2]. They take part in the kernel and particle approximations for

the real particles, but without being treated as real ones themselves. In order to distinguish them from real agents in ISO, I call them particles. They are categorized into two types:

**Virtual Particles Type I** These particles are placed on the solid boundary and carry themselves physical variables, specified by the problem. Normally, in order to correctly calculate the density of the real particles at the boundary, several layers of virtual particles type I have to be initialized. Their position and the other initially assigned physical variables do not evolve in time [6]. Furthermore, they exert a repulsive boundary force to prevent the interior particles from penetrating the boundary. If a virtual particle of type I is a neighbor of a real particle, the following force is applied along the centerline of these particles:

$$\mathbf{f}_{ij}^{boundary} = \begin{cases} D((\frac{r_0}{r_{ij}})^{12} - (\frac{r_0}{r_{ij}})^4)\frac{\mathbf{x}_{ij}}{r_{ij}^2} & \text{if } \frac{r_{ij}}{r_0} \leq 1 \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where

$D$ ... Problem dependent parameter; same scale as the square of the largest velocity

$r_0$ ... Cutoff distance; usually selected approximately close to the initial particle spacing.

This force is using a similar approach employed for calculating the molecular force of Lennard-Jones form [8].

**Virtual Particles Type II** These particles are placed continually within the boundary region and but do not evolve in their parameters. They are "mirroring" real particles near the boundary: As soon as a particle approaches the boundary, the method generates a virtual particle of type II, placed symmetrically outside the boundary, having the same density, pressure and mass, but opposite velocity. The status of the particles is updated every evolution step.

Together, these two particle types prevent real particles from penetrating the boundary. Important to note is that the choice of the parameters is crucial, as each SPH problem requires a separate set of values.

# 3 Application of SPH in this project

The aim of this project was to implement a fluid-like behavior for the swarm by simulating an incompressible, isothermal flow of constant viscosity, and under the influence of gravity. Due to the inherent particle-like nature of the Interactive Swarm Orchestra, SPH seemed to be best suited for this problem. Furthermore, as real-time simulation is a strongly desired feature, one has to lower one's sights at physical accuracy which is easily feasible with SPH due to its strong adaptability to the specific problems. But in order to guarantee at least the main physical principals such as symmetry of forces and conservation of momentum, one has to symmetrize the forces generated by the SPH model. This is done by simply averaging the corresponding physical quantities of an interacting pair of particles.

Thereby, the implementation more or less follows the one suggested by the paper [1]. Omitted have been the sections dealing with the visualization of the fluid and especially the surface of it, as ISO builds upon a different visualization concept, where the individual particles resp. agents have to been visible.

As temperature is assumed to be constant for the whole system, the energy equation resulting from conservation of energy can be omitted completely. Furthermore, as the number of particles is constant and each particle has constant mass, the mass conservation is guaranteed, which is why the continuity equation (1) can be left out too [1]. So, the only equation that needs to be solved, is the Navier-Stokes equation (2). Furthermore, since SPH is a lagrangian description, we work with the total time derivative, and thus do not have to look at the partial derivatives of the velocity field.

Particles carry the three quantities mass, position and velocity.

## 3.1 Calculation of the forces

### 3.1.1 Pressure force

The symmetrized particle approximation of the pressure force $-\nabla p$ is

$$\mathbf{f}_i^{\text{pressure}} = -\nabla p(\mathbf{x}_i) = -\sum_j m_j(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2})\nabla W_{ij}. \tag{8}$$

I have chosen this ansatz instead of the one proposed by Mueller et al. because this approach more consequently applies the concept of symmetrization.

For calculating the pressure, I have implemented two equations of state, the first one suggested by Mueller et al. [1] and the latter one by Liu [2]:

$$p = k(\rho - \rho_0), \tag{9}$$

$$p = B((\frac{\rho}{\rho_0})^\gamma - 1), \tag{10}$$

where

$\gamma$... constant set to 7

$\rho_0$... reference or rest density, chosen according to the problem

$B$... sets the limit for maximum change of density [2].

The introduced reference density makes the simulation numerically more stable, as it guarantees that the pressure is zero as long as the particle distribution is not changed.

### 3.1.2 Viscosity force

The symmetrized particle approximation of the viscosity force ($\nabla^2\mathbf{v}$) is

$$\mathbf{f}_i^{\text{viscosity}} = -\nabla^2\mathbf{v}(\mathbf{x}_i) = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W_{ij}. \tag{11}$$

Looking at the equation, one sees that viscosity is interpreted as a force arising from the nonzero relative velocities of interacting particle pairs, accelerating the particles in the direction of the relative speed.

Having calculated the forces, the acceleration of particle $i$ can be obtained by:

$$\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{f}_i}{\rho_i}. \tag{12}$$

## 3.2 Choice of Smoothing Kernels

Mueller et al. proposed the use of a polynomial kernel for calculating the nonforce field quantities, a spiky kernel for calculation of the pressure force and a special kernel designed for the viscosity. They were designed such that the balance between stability, speed and accuracy of the SPH method is as optimal as possible.

The chosen kernels all fulfill the conditions listed in section 2.1.1.

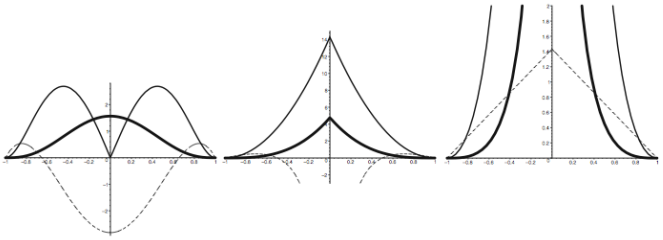


Figure 1: Left: $W_{poly}$, Middle: $W_{spiky}$, Right: $W_{viscosity}$; Thick lines: Kernels, Thin lines: Gradients, Dashed lines: Laplacian; Smoothing length $h = 1$; (Picture from [1])

### 3.2.1 Polynomial Kernel

$$W_{poly}(\mathbf{x}, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}, \text{ where } r = |\mathbf{x}|. \tag{13}$$

This Kernel is fast, as no square root has to be calculated, but due to the lacking repulsion force for close distances of particles, it is not appropriate for calculating the force fields. For this purpose, Mueller et al. [1] applied the following kernel, designed by Delbrun [9]:

### 3.2.2 Spiky Kernel

$$W_{spiky}(\mathbf{x}, h) = \begin{cases} \frac{15}{\pi h^6}(h - r)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}. \tag{14}$$

### 3.2.3 Viscosity Kernel

$$W_{viscosity}(\mathbf{x}, h) = \begin{cases} \frac{15}{2\pi h^3}(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases}. \tag{15}$$

This kernel guarantees that the kernel itself and its second derivative are always larger than zero within the support domain, thus preventing the artifact of increasing the relative velocity of interacting particles, which appears in coarsely sampled velocity fields ([1]) as is the case in our project. A further advantage is that the Laplacian of the kernel can be calculated easily: $\nabla^2 W(\mathbf{r}, h) = \frac{45}{\pi h^6}(h - r)$.

### 3.2.4 Smoothing Length

The smoothing length of the kernel function is a particularly important variable as it directly influences the efficiency of the computation and the accuracy of the solution, negatively affecting the first one when being too large and negatively affecting the latter one when being too small. As a rule of thumb, one says that in three dimensions, the number of neighboring particles should be about 60 if the particles are placed in a lattice with a smoothing length of 2.5 times the particle spacing [2].

In this project, a constant smoothing length is used. The value is chosen depending on the initial average density of the system.

There are many ways to dynamically evolve the smoothing length so that the number of neighboring particles remains relatively constant. But the neighborhood calculating algorithm in the ISO framework is not yet build such that it would allow for a continually adapting neighbor radius.

## 3.3 Simulation

For the integration of equation (12), the second order scheme Leap-Frog Integration is used [1]. In the Leap-Frog Scheme, positions are defined at times $t_i, t_{i+1}, t_{i+2}, t_{i+3}, \ldots$, whereas velocities are defined at times halfway in between, $t_{i-\frac{1}{2}}, t_{i+\frac{1}{2}}, t_{i+\frac{3}{2}}, \ldots$, where $t_{i+1} - t_{i+\frac{1}{2}} = t_{i+\frac{1}{2}} - t_i = \frac{dt}{2}$. But if you want to define the quantities only at integer times, the scheme looks like follows:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i dt + \mathbf{a}_i \frac{(dt)^2}{2}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + (\mathbf{a}_i + \mathbf{a}_{i+1}) \frac{dt}{2}.$$

As the ISO project requires real-time processing, time steps up to about 50 milliseconds are allowed. As in the paper of Mueller et al [1], I used constant time steps, eventhough adaptive ones would probably lead to a better performance. At the same time, there's a lower limit to the size of the time step as very small time steps would result in a larger computational effort and slow down the system eventhough the accuracy of the simulation would gain a lot from such small time steps. The system itself is very sensitive to the time step chosen, and small differences in the time step can lead to a completely different behavior (especially at the boundaries, see section 4.1.3). Further description in section 4.1.4.

# 4 Implementation

## 4.1 Implemented Structure

In order to implement the SPH method in the ISO framework, I have created two behavior classes. One is called *iso_flock_fluid_behavior* and applies the above-mentioned SPH scheme, calculating all the required quantities such as density, pressure, kernel and the relevant forces. The second class is called *iso_flock_virtual_particles_behavior*. It calculates the repulsion force arising from the influence of virtual particles of type I within a neighborhood of a real particle, and updates the status of virtual particles of type II.

### 4.1.1 General Structure

In figure 2, you see the general procedure of an SPH simulation within the ISO framework.

#### Spaces

The two behaviors require three different spaces for calculation and representation of the SPH system:

**fluid_position_space** can be seen as the "classical" space, within which the swarm containing the "real" agents, i.e. the fluid particles, lives in. It visualizes the agent's positions and resulting behavior.

**surface_position_space** is the space within which the positions of the virtual particles of type I live and within which the repulsion forces are calculated. To put it simply, it is the space of solid boundaries, represented by the type I - particles.

**mirror_position_space** is the space within which the positions of the virtual particles of type II live. Depending on the distance of the fluid agents to the solid boundaries, type II - particles are created, updated in their parameters, or destroyed within this space.

Additionally, the virtual particles of type I and II not only live in their original spaces, but also in the *fluid_position_space*, as they have to be considered in the SPH particle approximations of the fluid agents too, as a means of stabilization and accuracy (see section 2.1).

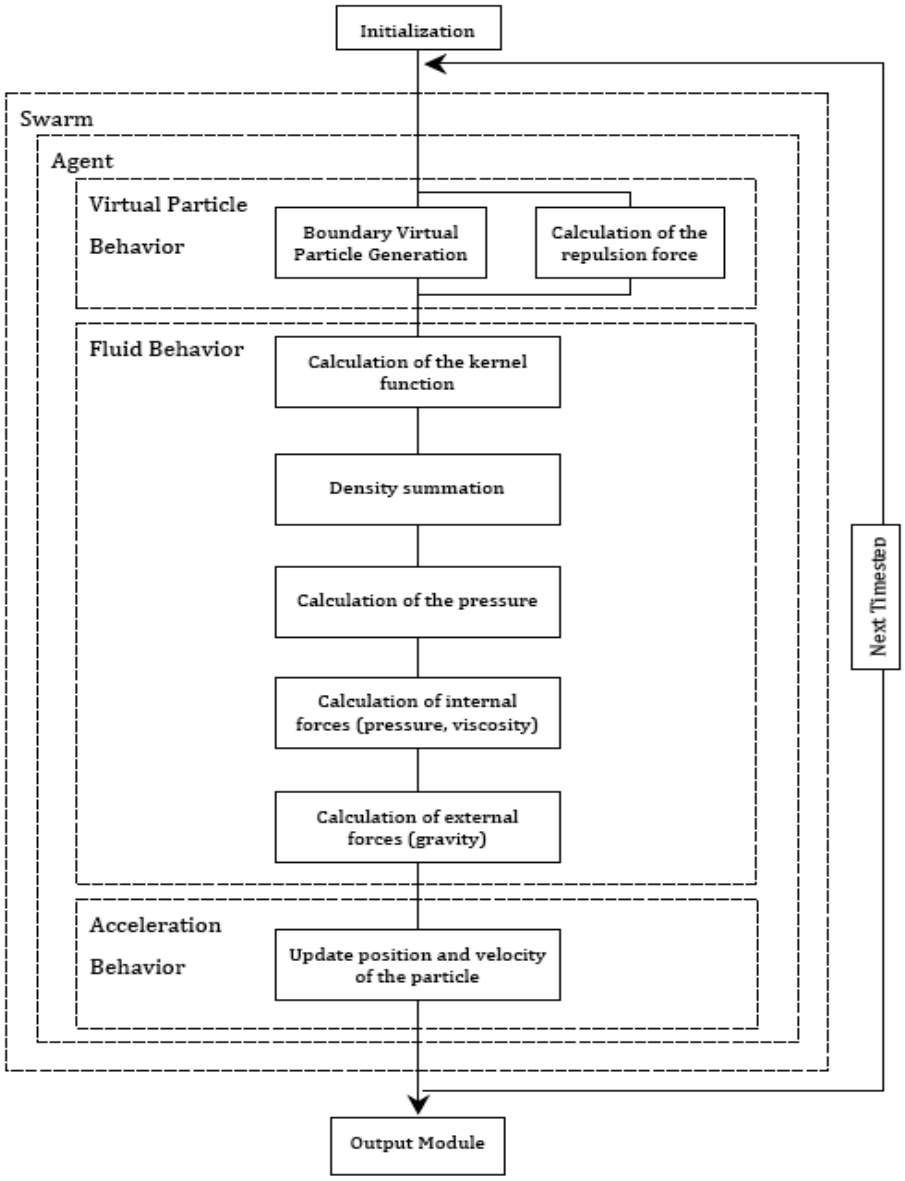



Figure 2: Structure of the implemented code

### Parameters

The main parameters the agents are carrying are *mass*, *position*, *velocity*. Furthermore, as a means of the final integration of behaviors *acceleration* and *force* are added as parameters. Finally, SPH requires the agents to have *density* and *pressure*.

These are the parameters the agents contain. The parameters specific to the behaviors themselves are listed in the following subsections.

### 4.1.2 iso_flock_fluid_behavior

ISO is a highly serially working code. As explained in the introduction, agents in the ISO framework contain a list of parameters and a list of behaviors by which they are defined. When running the simulation, the program goes within one time step from one agent to another, activating every time the behavior list of the actual agent and running the listed behaviors successively, before proceeding with the next agent. This complicates the implementation of a SPH model for fluid dynamics, because SPH requires that agents can access the other agent's simultaneously evolving quantities such as density and pressure through the kernel function. But the trick implemented in ISO to overcome this problem and to simulate parallel behavior is the following: When updating an agent, the newly calculated values are first buffered, invisible to the other agents, and at the end replacing the old values.

As the neighborhood of an agent does not change within one time step, and thus also not within the execution of the behavior for an individual agent, I store copies of the density and pressure of the neighboring agents within arrays (so-called *QVectors*), as well as the values of the desired kernel functions and its derivatives between the actual agent and its neighbors. This reduces the computational cost, speeds up the simulation, but increases memory consumption as well.

#### Behavior-specific parameters

**Kernel** is the continually changing parameter containing all the kernel values of type "poly", "spiky" or "viscosity" the agent shares with its neighbors.

**KernelDerivative** is the analogous for the derivative of the kernels.

**Neighbor Density** stores the densities of the neighboring agents within an array for faster access.

**Neighbor Pressure** is the same for the pressure.

**Reference Density** stores the reference density of the agents and is a global variable, meaning that it has the same value for all agents.

**Initial Pressure** is a fixed constant, chosen accordingly to the problem.

**Viscosity constant**

**Gravity acceleration**

### 4.1.3 iso_flock_virtual_particles_behavior

The above-mentioned inherent property of ISO concerning the serial procedure causes some complications when thinking about how to implement the virtual particle behavior. Unfortunately, I could not find an efficient way of creating or updating virtual particles of type II such that they could still be incorporated within the particle approximation of the real agents within the same time step, so that the current implementation causes some additional approximation errors. To keep those errors as small as possible, the time step of the simulation has to be chosen small enough so that the agents do not make too large steps within one simulation step, as the cutoff distance of the repulsion force exerted by virtual particles of type I is very small. Thus, if the time step was too large, the spatial steps of the agents would be larger than the cutoff distance, leading to the fact, that the boundary particles never would be seen as neighbors of the agents.

The code for administrating the virtual particles of type II has been implemented such that it fits to set-ups which are made of fluids inside a jar. The only condition the jar has to fulfill is that the agent maximally faces 3 different surfaces at once within the jar. This means that the code can maximally generate 3 type II - virtual particles per agent, which is for example the case if the agent is in a corner of a cubic container.

#### Behavior-specific parameters

**Cutoff for mirror particles** is the distance from the agents to surface particles within which type II - particles are generated.

**Cutoff for surface particles** sets the radius of the area around an agent within which they sense the presence of surface particles.

**Jar size** stores the size of the initially defined container for the agents.

**Maximal Velocity** is the parameter storing the maximal veloctiy of the agent, in order to calculate the repulsion force exerted by type I - particles.

### 4.1.4 Time step and smoothing length

The choice of the time step is crucial to the numerical solution of the fluid dynamical equations, as mentioned in section 3.3. Not only, it determines the accuracy of a solution, but also is relevant when seeking for real-time solutions. Adaptive time steps were proposed by Morris et al. [7], considering viscous diffusion: $\Delta t = 0.125 \times \frac{h^2}{\nu}$, where $\nu = \frac{\mu}{\rho}$ is the kinetic viscosity. But here, the time steps are constant.
The smoothing length has to be chosen such that in a regular initial agent distribution entirely with free surfaces, no pressure forces arise (see 3.2.4). As in such a case, the agents at the boundary inherently have a different density than the ones inside the distribution, which would give rise to pressure forces, the smoothing length has to be set such that in the regular distribution, agents do not have neighbors inside their support domain, or by ensuring that the reference density equals the initial density.

## 5 Discussion

### 5.1 General Problems of Implementation

While SPH is a method acting globally, ISO is a code working rather on the individual level. That is, - put in ISO terminology - SPH is a global behavior, which considers the individual particles only as points at which physical quantities are evaluated, but is rather interested in the average of all particles than on the particles themselves. As such, SPH is a scheme which works most efficiently if the computational structure executing it works as parallel as possible. This would require to calculate several quantities such as the kernel values globally, avoiding redundant evaluations. In contrast, ISO focusses on the individual agents (i.e. particles) as entities, interacting with other agents. Therefore, ISO is built such that behaviors are nested within agents, and not vice versa. As mentioned before, this problem could be circumvented due to the strong object-oriented character of ISO, instead of constructing global arrays, etc. . As agents are instances of the agent class, they are containers for all the necessary parameters and field quantities, which can easily be accessed by others by means of the appropriate functions. But the downside to it is that it slows down the program, comprising the required real-time character.

A further problem was the above-mentioned condition of real-time interaction. As SPH is a simulation method for physical phenomena, it has to fulfill certain accuracy and stability conditions which in turn require the use of a minimal amount of agents. The smaller their number, the less the resulting behavior resembles a fluid dynamical behavior and the less stable it is. I tried to ensure interactivity by reducing the number of computations and minimizing the use of computationally expensive functions , but still, the program is not fast enough. For example already in the case of two fluid drops colliding, each consisting of 216 agents, the simulation visibly slows down. Suggestions of how to overcome this problem are mentioned in section 6.

One very subtle problems was to correctly initialize the set-up of fluid agents and boundary particles. It requires to tune the parameters, such as initial and reference density and pressure, smoothing length, cutoff for mirror and for surface particles, etc. Because the SPH method and the virtual particle generator are very sensitive to these parameters, slight changes can lead to large, unwanted effects. As the system itself has a high degree of freedom (number of parameters) and is nonlinear to a large extent, adjusting the parameters caused a huge mess. Especially problematic, hardly feasible, is the calibration of the virtual particle - parameters. This is due to the fact that the system normally gets more stable, the more particles and agents are used, as this smoothes out unwanted effects and allows more homogeneous distributions. But as this demand conflicts with the claim to interactivity, the number of particles has to be reduced, whereby errors of the system do not get compensated by the averaging and thus lead to unwanted artefacts.

### 5.1.1 Boundary Treatment

One of the most challenging issues was the implementation of boundary conditions. Unfortunately, the paper of Mueller et al. did not mention this problem at all [1], so that I had to look for other possibilities, always having in mind the claim to interactivity. But it became apparent that with this approach, not only did I meet some fundamental problems I could not solve within the given time of the project, but also extensive computational power is required, which again are inconsistent with real-time interactions. For this reason, I propose to implement other boundary conditions, which are described in the section 6.

**What were the main problems of the virtual particle implementation?**

Firstly, it is hardly possible to tune the parameters such that the boundary method

really prevents all the agents in all situations from penetrating the boundary. Testing has been done with both single agents and with fluid drops colliding with the bottom of the jar. In the first case, adjustment of parameters is easy as long as the velocities of the agents are not too large (relatively). As soon as velocities become large, the agents moving towards the wall do not feel the virtual particles in their neighborhood anymore and thus penetrate the wall, as the system works with constant smoothing lengths and time steps. In the latter case, things get again more complicated. For example, when the agents collide with the wall, they come very close to each other due to the compression of the drop, giving rise to very large pressure forces acting in the direction of movement. In order to compensate for these large-range fluctuations of forces, the repulsion force generated by the virtual particles, and the contributions of the virtual particles to the SPH approximations would have to adapt dynamically in the same order of magnitude within a very short time interval. Essential is the ratio between smoothing length, cutoff distance, maximal velocity and simulation time step. It has to adapt to the actual situation, which again would require the implemented code to allow for dynamically changing the parameter values.

Secondly, when agents are repulsed at the boundary, the forces generated by the virtual boundary particles effect the agents behavior on a much too large spatial scale, leading to behavioral artefacts. When a fluid drop falls onto the ground of the jar, driven by gravitation, the repulsive force lets the agents bounce on the ground for an infinitely long time without letting them come to rest, showing an oscillatory behavior in a physically impossible manner. A possible reason for that could be that the system is assumed to be isothermal and thus the energy equation has been omitted. But obviously, in a system with boundaries, isothermal conditions are not fulfilled, as boundary conditions imply friction. This could also explain why the bouncing behavior persists, no matter what parameter values have been chosen. Induced friction would damp the system. At the same time, one could argue that this approach of virtual particles still would be compatible with the isothermal condition, as the generated repulsion force is included as an external force into the Navier-Stokes equation and as the concept of virtual particles complies with the SPH approach (considering the fact that Mueller et al. worked with isothermal fluids, and still were able to implement solid boundary conditions).

These problems probably arise partly because the claim to real-time computation does not allow for very small temporal and spatial magnitudes which obviously would be required for accurate simulations. For example, Liu et al use a cutoff distance in the order of $10^{-5}$ and smoothing lengths in the order of $10^{-2}$ [2]. But within the ISO

framework, such magnitudes are not beneficial, as said before.

Finally, due to the above-mentioned lower boundary of parameter magnitude, the type II virtual particles do not show large effects within the SPH approximation of the agents. In order to be a neighbor of an agent, the distance of the virtual particles from the agents must be smaller than the smoothing length, so the cutoff parameter for these type II particles has to be half the smoothing length. But as soon as the agents have a certain velocity, they will hardly feel the presence of these virtual particles because of the constant, relatively large simulation time step.

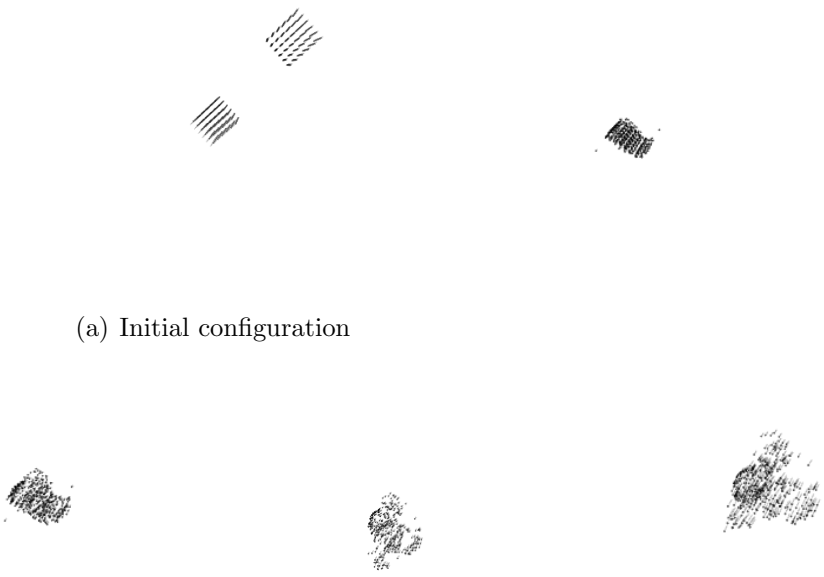
## 5.2 Tuning of Parameters

Due to the previously described problems arising from the implementation of virtual particles in the ISO framework, I only mention the role of parameters of the fluid dynamical behavior.

### 5.2.1 Fluid Behavior

If you only look at the fluid behavior within an "infinite" space - i.e. considered as without any boundaries -, the tuning of the smoothing length and reference density just depend on the initial agent distribution. In order to simulate fluid drops, the initial distribution has been set as a regular, cubic ensemble of agents. Thus, smoothing length is chosen to be slightly smaller than the initial agent spacing, and the reference density is set equal to the initial density. As the smoothing length is set such that no agents fall into the support domain of another agent, the boundary of the free surface drops does not have to be treated specially.

Figure 3 shows a good example of two cubic fluid drops approaching each other in z-direction, such that they have approximately the same velocity when colliding. The initial pressure has been chosen large enough so that the pressure and viscosity force are within the same range ($\mu = 1.5$ [2]). If the initial pressure is too small or the viscosity coefficient too large, the agents within the drops will stick together as soon as they collide with each other, which is due to the fact that the viscosity force becomes so dominant, that the dynamics is entirely driven by the fraction of force showing in the direction of relative distance between two agents. This can clearly be seen when comparing figure 4 with a small initial pressure to figure 5 with standard initial pressure. Additionally, the gravity acceleration has been adjusted such that the gravity force is within the same range as the other forces. Velocities have to be chosen within a "reasonable", problem specific range (here within $[0.0, 1.0]$), because



(a) Initial configuration

Figure 3: Succeeding frames of two cubic fluid drops approaching each other in z-direction. The value of the initial pressure is $1.013 \cdot 10^{-1}$.

Figure 4: Succeeding frames after collision. The initial pressure is $1.013 \cdot 10^{-4}$.

Figure 5: Succeeding frames after collision. The initial pressure is $1.013 \cdot 10^{-1}$.

the faster the agents are, the smaller the time steps have to be chosen for correctly running the simulations.

# 6 Outlook

## 6.1 Wall Weight Functions as an Alternative Implementation of the Boundary Conditions

Unfortunately, as elaborated in the previous section, the implementation of virtual particles led to several unsolved complications. For this reason, the next step of the project should be to try out another possible implementation of the boundary conditions. This implementation would have to comply with the restrictions given by the ISO framework, assuming that the framework itself will not be changed with respect to the neighborhood algorithm, the variability of smoothing length and the one of the time step. Furthermore, the implementation has to overcome the problems inherent to boundary conditions in SPH, i.e. the fact that the boundaries are not

well-defined and the kernel sum deficiencies at the boundary.

In general, most possibilities rely on the representation of the boundary by boundary particles. While some combine this with the creation of virtual particles [2][10], others let the boundary particles exert repulsion forces [7]. Thus, one could think of improving the existing implementation of virtual particles by creating several fixed rows of virtual particles of type II within a smoothing length instead of using virtual particles of type II. But as could be seen with the realization of such a method in ISO, the problem arising from the use of boundary particles within the given framework among others are the computational costs which steeply ascend with the number of generated boundary particles, because particles are based on the complex concept of agents.

Instead, I would like to propose here another possible approach of implementing boundary conditions in a computationally efficient way. This approach, suggested by Takahiro Harada et al., uses a *distance function*, calculated from a polygon model, to represent the boundary, and introduces so-called *wall weight functions* required to handle particles close to the boundary [11]. With respect to the ISO framework, this has two significant advantages: Firstly, it allows to dramatically reduce the total number of particles, to increase the computational efficiency with respect to real-time processing and to use arbitrarily complex shapes of boundaries [11]. Secondly, the ISO framework has already the concept of shapes and with it a set of classes (iso_space_shape, etc.) concerned with the relationship of agents with geometric space shapes and as such contains already functions required for calculating such distances.

It has to be said that it is wrong to claim that this method fully dispenses with boundary particles and in this sense provides a completely novel approach. The method just circumvents the necessity of integrating the boundary particles into the SPH approximation during the simulation, causing this additional computational effort, by doing the calculations of the wall weight and the distance functions involving those boundary particles **in advance**.

The theoretical framework does not differ very much from the previous one. Still, the boundary is represented by boundary particles. But now, the SPH approximations of the density, the viscosity and the pressure are split into two terms. One contains the contributions of the real particles, and the other one consists of contributions from the boundary particles (e.g. density: $\rho_i(\mathbf{r}_i) = \sum_{j \in fluid} m_j W(|\mathbf{r}_i - \mathbf{r}_j|) + \sum_{j \in wall} m_j W(|\mathbf{r}_i - \mathbf{r}_j|)$). As the boundary particles are placed temporarily on the boundary depending on the relative position of particle and wall (see figure 6), the contribution of such wall particles is uniquely determined by the distance to the

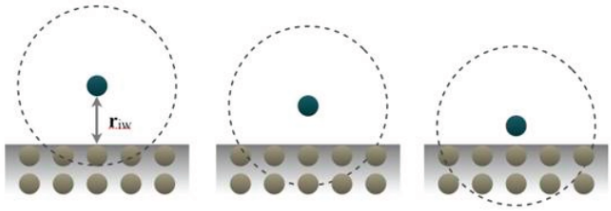



Figure 6: Distribution of wall particles for the computation of wall weight functions

wall boundary as follows:

$$\rho_{i,wall} = \mathbf{Z}^{rho}_{wall}(|\mathbf{r}_{ij}|) \tag{16}$$

$$\mathbf{f}^{visc}_{i,wall} = \mu(\mathbf{v}_i - \mathbf{v}_j)\mathbf{Z}^{visc}_{wall}(|\mathbf{r}_{ij}|) \tag{17}$$

$$\mathbf{f}^{press}_{i,wall} = \mathrm{m}_i \frac{\Delta \mathrm{x}_i}{\Delta t^2} = \mathrm{m}_i \frac{(d - |\mathbf{r}_{ij}|)\mathbf{n}(\mathbf{r}_i)}{\Delta t^2}, \tag{18}$$

where

$$\mathbf{Z}^{rho}_{wall} = \rho_i = \sum_{j \in wall} m_j W(|\mathbf{r}_{ij}|).$$

$$\mathbf{n}(\mathbf{r}_i) \text{is the normal vector of the closest boundary point at} \mathbf{r}_i. \tag{19}$$

$\mathbf{Z}^{rho}_{wall}(|\mathbf{r}_{ij}|)$ and $\mathbf{Z}^{visc}_{wall}(|\mathbf{r}_{ij}|)$ are the so-called wall weight functions.

### 6.1.1 Calculating the Wall Weight Function

Since the wall weight functions solely depend on the distance to the wall boundary, the functions can be computed in advance. Particles are placed on the boundary according to figure 6 before the simulation starts such that the density of the wall particles remains constant. The wall weight functions are only computed at a few points within the smoothing lengths. By interpolation, the function values can be computed at all the other points [11].

For simple polygons, the distance to the boundary is easily calculated, but as soon as more complicated polygons are used, an auxiliary distance function is required [11] Such a function is already provided by the ISO framework. As the boundaries remain stable, the distances from possible particle positions to the boundary can be precomputed too. Additionally, the normal vectors of the boundaries can be obtained by calculating the gradient vectors at different locations, as they have the same direction as the normal vector for the closest boundary [11]. Here, interpolation for the other spatial points is used too.

The interpolation procedure can take place during the simulation, as the main part of boundary computation is done in advance.

## 6.2 Further possibilities of improvement

As already elaborated in section 5, SPH is a type of "globally acting" behaviors. Thus, to improve efficiency and ensure real-time processing, it would be advisable to implement in ISO the possibility of implementing such behaviors encompassing the agents, and not only vice versa.

Furthermore, ISO should provide the possibility of dynamically changing global parameters such as smoothing length and thus the neighbor radius used in the neighborhood algorithm. Additionally, it would be interesting to have the possibility of continually generating fluid agents, simulating a water tap. This again would require a globally acting behavior.

Finally, this fluiddynamical approach could also be extended to smoke behavior in various ways. A very interesting one is suggested by the L.Shi et al. in [**?**] in which smoke is animated such that it resembles moving or still objects.

## References

[1] M.Muller, D.Charypar, M.Gross, Particle-based Fluid Simulations for Interactive Applications, *Proc. of Siggraph Symposium on Computer Animation*, pp. 154 -159, 2003

[2] G.R.Liu, M.B. Liu in *Smoothed Particle Hydrodynamics* (World Scientific Publishing Co. Pte. Ltd.), 2003

[3] D. Bisig, M. Neukom, J. Flury, Interactive Swarm Orchestra - Tutorials, http://www.i-s-o.ch/doc/index.html

[4] G. Beni, J. Wang, Swarm Intelligence in Cellular Robotic Systems, *Proceed. NATO Advanced Workshop on Robots and Biological Systems*, Tuscany, Italy, June 26-30, 1989

[5] P. Wesseling in *Principles of Computational Fluid Dynamics* (Springer Verlag), 2001

[6] J.J. Monaghan, Smoothed Particle Hydrodynamics, *Annu. Rev. Astrophys.*, Vol. 30, pp. 543 - 574, 1992

[7] J.J. Monaghan, Simulating Free Surface Flows with SPH, *J.Comp.Phys.*,Vol. 110, pp. 399 - 406, 1994

[8] J.J. Monaghan, A. Kos, Solitary Waves on a Cretan Beach, *J. Waterway, Port, Coastal, and Ocean Engrg*, ASCE,Vol. 125, 1999.

[9] M. Desbrun, M. P. Cani, Smoothed particles: A new paradigm for animating highly deformable bodies, *Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation)*, pp. 61 - 76, Springer-Verlag, 1996

[10] H. Takeda, S.M. Miyama, M. Sekiya, Numerical Simulation of Viscous Flow by Smoothed Particle Hydrodynamics, *Prog. of Theor. Phys.*, Vol. 92, pp. 939 - 960, 1994

[11] T. Harada, S. Koshizuka, Y. Kawaguchi, Improvement in the Boundary Conditions of Smoothed Particle Hydrodynamics, *Computer Graphics and Geometry*, Vol. 9, pp. 2 - 15, 2007

[12] L. Shi, Y. Yu, Controllable Smoke Animation with Guiding Objects, *ACM Transactions on Graphics*, Vol. 24, No. 1, pp.140 - 164, 2005

# 7 Appendix - A short guide through the code

## 7.1 Iso_flock_fluid_main.cpp

This subsection provides a short overview of the code and gives basic explanations of relevant pieces of the code. I do not go explain the fluid behavior and the virtual particle behavior in detail, as the code itself should be self-explaining, and thus I only list what has to be known in order to run and modify the code correctly.

### 7.1.1 Set the parameters

```
////////////////////////////////////
/// Set the necessary parameters ///
////////////////////////////////////

/*
    Adjust the parameters if required.
*/

//Parameters for the jar
    unsigned int Jar = 0; //1...Generate the jar; 0...Do not generate the jar

    double JarLength = 6; //Sidelength of the ground surface
    //--> HAS TO BE SET IN VIRTUAL PARTICLE BEHAVIOR TOO!

    double JarHeight = 3; //Height of the jar
    double GridSize = 10; //Spacing of virtual particles for unit length
    math::Vector3<float> GroundLeftCorner = math::Vector3<float>(0.0,0.0,0.0); //Position of the jar
    unsigned int visible = 1; //1...Virtual Particles are visible; 0...Virtual Particles are not visible

//Parameters for the fluid drops
    double CubeLength = 2; //Sidelength of the cubeshaped fluid drops
    double GridSpace = 3;  //Spacing of agents for unit length (i.e. number of agents per unit length)

    //The initial positions and velocities of the two fluid drops
    math::Vector3<float> InitialPosition1 = math::Vector3<float>(0.3, 0.0, 3.5);
    math::Vector3<float> InitialVelocity1 = math::Vector3<float>(-0.05, -0.05, -0.2);

    math::Vector3<float> InitialPosition2 = math::Vector3<float>(0.1, -0.5, 0.0);
    math::Vector3<float> InitialVelocity2 = math::Vector3<float>(0.0, 0.08, 0.4);

//Simulation Parameters
    double smoothLength = (1/GridSpace) - 0.01 ; //Smoothing length assigned to all agents

    double cutOff = 0.2;                        //Parameter required for the mirror_particle behavior
    //--> HAS TO BE SET IN VIRTUAL PARTICLE BEHAVIOR TOO!

    double IntegrationTimeStep = 0.1;           //Integration Time Step of the simulation
```

Before running the simulation, one first has to decide, whether or not a jar is required, find out where to place the fluid drops such that they collide and decide about the accuracy of the simulation by defining the resolution of the fluid drops (GridSpace) and the size of time steps. One has to consider that for larger velocities, smaller time steps should be implemented in order to guarantee stability and fluid dynamical behavior. As can be seen, the smoothing length is set such that it fulfills the above-mentioned conditions.

### 7.1.2 Define the correct spaces

```
////////////////////////////
///  Set up the spaces   ///
////////////////////////////

space::SpaceManager::get().addSpace( new space::PointSpace("fluid_position_space", 3) );
//fluid_position_space is used for the interaction among the fluid particles themselves
space::SpaceManager::get().addSpace( new space::PointSpace("surface_position_space", 3) );
//surface_position_space is used for calculating the force the surface particles exert on the fluid particles
space::SpaceManager::get().addSpace( new space::PointSpace("mirror_position_space", 3) );
//mirror_position_space is used for calculating the force the mirror particles exert on the fluid particles
```

The fluid dynamical simulation requires the generation of three types of different spaces, according to the implemented methods. The main space is the point space "fluid_position_space", as this is the one where all the fluid agents live in.

### 7.1.3 Set up the boundaries

```
////////////////////////////
/// Create Surface Agents///
////////////////////////////

if(Jar==1){

double GroundSurface = JarLength*JarLength;
double stepx, stepy, stepz;

double surfaceMass = 0.1;
double volume = GridSize*GridSize*GridSize;
double surfaceDensity = surfaceMass/volume;
float surfacePressure = 0.0;

for (unsigned int i=0; i<GridSize*JarLength; ++i){...}

}
```

The surface of the jar is being occupied by virtual particles of type one (which are assigned to the "fluid_surface_space"), according to the defined granularity. All particles have the same properties, and are generated within the indicated for - loop.

### 7.1.4 Add the fluid behavior to the swarm

```
////////////////////////////
///   Set up the swarm   ///
////////////////////////////


Swarm* swarm = new Swarm("Fluid swarm");

//Calculate the Kernel of the particles, whereby the smoothlength is set such that in the initial distribution, no neighbours exist:
double myKernel =  1.5667 / (smoothLength*smoothLength*smoothLength);
double mass = 0.1;
float myDensity = myKernel*mass;

swarm->addParameter( "position", 3 );
swarm->assignNeighbors( "position", "fluid_position_space", true, new space::NeighborGroupAlg( smoothLength, 100, true ) );
swarm->assignNeighbors( "position", "surface_position_space", false, new space::NeighborGroupAlg( cutOff, 100, true) );
swarm->assignNeighbors( "position", "mirror_position_space", false, new space::NeighborGroupAlg( 2*smoothLength, 100, true) );
swarm->addParameter( "velocity", 3 );
swarm->addParameter( "acceleration", 3 );
swarm->addParameter( "force", 3 );
swarm->addParameter( "mass", 1, mass );
swarm->addParameter( "density", 1, myDensity);
swarm->addParameter( "pressure", 1);

swarm->addBehavior( "resetForce", ResetBehavior( "", "force" ) ); //Wofür?

//swarm->addBehavior( "randomize", RandomizeBehavior( "", "force" ) );
//swarm->set("randomize_range", 0.1);

swarm->addBehavior("VirtualParticle", VirtualParticleBehavior("position:surface_position_space position:mirror_position_space...
... position:fluid_position_space velocity", "force"));
swarm->set("VirtualParticle_cutOff", mirror_cutOff);

swarm->addBehavior("Fluid", FluidBehavior("mass position:fluid_position_space velocity density pressure", "force" ));
swarm->set("Fluid_SmoothLength", smoothLength);
```

Here, you see all the inputs that are required for the creation of the fluid - and the virtual particle - behaviors.

### 7.1.5 Create the fluid drops

```
//////////////////////////////////////////
/// Initial Configuration of two cubes ///
//////////////////////////////////////////
unsigned int CubeNumber = 2;

double fluidcount = GridSpace*CubeLength*GridSpace*CubeLength*GridSpace*CubeLength;
unsigned int agentCount = (int) CubeNumber*fluidcount;
swarm->addAgents(agentCount);

double fluidstepx, fluidstepy, fluidstepz;

//Generate the first cube
for (unsigned int z=0; z<GridSpace*CubeLength;++z){...}

//Generate the second cube
for (unsigned int z=0; z<GridSpace*CubeLength;++z){...}
```

## 7.2 Iso_flock_fluid_behavior.cpp

```
void
FluidBehavior::act()
{

    if(mActivePar->value() <= 0.0) return;

    space::NeighborGroup& positionNeighbors = *mPositionNeighbors;
    math::Vector<real>& force = mForcePar->backupValues();
    math::Vector<real>& tmpForce = mTmpForce;
    force = math::Vector3<real>(0,0,0);
    tmpForce = math::Vector3<real>(0,0,0);

    real& mass = mMassPar->value();
    unsigned int totalNeighborCount = positionNeighbors.neighborCount();

    //Generate the kernels
    FluidBehavior::kernel("Poly");
    FluidBehavior::kernelDer("Poly",0);

    //Calculate the density and pressure
    FluidBehavior::density();
    FluidBehavior::pressure();

    //Calculate the Viscosity and Pressure force
    FluidBehavior::viscosityforce();
    FluidBehavior::pressureforce();

    //Calculate the gravity force, if required
    real& myDensity = mDensityPar->value();
    if(mGravityActivePar->value() == 1){
        const math::Vector<real> gravity =  mGravityPar->values();
        tmpForce+=gravity*myDensity;
    }

    //Calculate the resulting forces
    force += tmpForce;
    force = force*mass/myDensity;
}
```

Here, you see the act() function of this behavior. First, the kernel values for the focal agent and its neighbors are calculated. Then, the code calculates the density and the pressure of the focal agent, storing the respective values of the neighbors in the arrays constructed for this purpose. Only then, the forces generated by pressure and viscosity (and finally gravity) can be calculated.

As the ISO code calculates the acceleration by *acceleration = force / mass*, but as within SPH, the acceleration has to be calculated by *acceleration = force / density*, we have to divide the resulting force by the density of the agent.

### 7.3 Iso_flock_virtual_particle_behavior.cpp

```
void
VirtualParticleBehavior::act(){

        math::Vector<real>& force = mForcePar->values();
        math::Vector<real>& tmpForce = mTmpForce;
        space::NeighborGroup& positionSurfaceNeighbors = *mPositionMirrorNeighbors;
        unsigned int totalNeighborCount = positionSurfaceNeighbors.neighborCount();

        VirtualParticleBehavior::SurfaceParticles();
        VirtualParticleBehavior::MirrorParticles();
        force+=tmpForce;
}
```

The first function call VirtualParticleBehavior::SurfaceParticles() calculates the repulsion force which is generated by the virtual particles of type I within the surface_position_space, as long as the focal agent is close enough to the boundary. The second one, VirtualParticleBehavior::MirrorParticles(), generates the mirror particles, depending on the position of the focal agent relative to the boundaries.