NTNU

Norwegian University of Science and Technology

Particle-based Powder-snow Avalanche Simulation Using GPU

Leif Kåre Hornnes Yndestad

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Torbjørn Hallgren, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal of this thesis is to develop a numerical simulation of powder-snow avalanches. Recent research in mathematical modeling of snow avalanche flow has provided sets of governing equations that describe the dynamics of such flow.

The computational method will be based on Smoothed Particle Hydrodynamics (SPH), a particle-based simulation technique that provide numerical solutions to fluid motion. Additionally, the flow simulation has been accelerated by an implementation on the GPU.

Assignment given: January 17th 2011
Supervisor: Torbjørn Hallgren, IDI

Abstract

The increased attention given to the determination of the complex behavior of powder-snow avalanche flow has resulted in several mathematical models being developed to describe the flow of such phenomena. In this thesis, a particle-based numerical solution of a mathematical model of powder-snow avalanche flow has been developed, in order to simulate and visualize the dynamics of such flow.

These physics-based fluid simulations requires a lot of computational power in order to calculate the governing equations within reasonable time. The numerical solution has therefore been implemented on the GPU, in order to take advantage of its highly parallel architecture, and provide the necessary computational power to accelerate the calculations of the governing dynamics.

The resulting simulation is shown to procude an evolving flow, indicative of complex behavior, that is dependent on the physical parameters provided to the system.

Preface

This report is the result of the master's thesis leading to the degree of Master of Science in Computer Science at the Norwegian University of Science and Technology.

I would like to thank my supervisor, Torbjørn Hallgren, for his guidance throughout the duration of this project.

Trondheim, Norway
June 2011

Leif Kåre Hornnes Yndestad

Contents

List of Figures

List of Algorithms

Notations

∇	gradient operator: $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}]^T$
$\nabla\cdot$	divergence operator: $\nabla\cdot = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z}$
∇^2	laplacian operator: $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$
$\mathbf{x}_{ij}$	distance vector between particle i and j.
$W(\mathbf{x}_{ij}, h)$	kernel function
$\mathbf{f}$	force
$\mathbf{g}$	gravity
ρ	density
ρ_0	rest density
ρ_r	reference density
h	support radius or smoothing length
k	gas constant or stiffness value
t	time
Δt	simulation time step
$\mathbf{x}$	position vector
$\mathbf{v}$	velocity vector
m	mass
p	pressure
$\langle f(\mathbf{x}) \rangle$	SPH approximation

CHAPTER 1

Introduction

1.1 Motivation and Aim

Fluid animation is considered one of the hardest problems within the field of computer graphics, due to the complex physical behavior a fluid possess. Examples of such behavior include deformations, turbulence, vortex formations, and interface dynamics. Because of this complex dynamic of fluid flow, visually pleasing results are very hard to produce by hand, even for the most talented of artist. As a consequence of this, tools and applications have been produced to aid the creation of realistic fluid animation. The method employed within these tools all have the goal of describing the motion of fluid. This motion is governed by the Navier-Stokes equations, which is a set of constitutive mathematical equations relating the fluid quantities.

A wide range of natural phenomena have been successfully modeled by applying these governing equations to describe their flow. Some examples include water [29], smoke [27], and fire [72]. Powder-snow avalanche is a phenomenon that has gained a lot of attention over the years, as researchers are seeking to best describe the dynamics of its flow. This focus have been motivated by the need for avalanche hazard-zoning, as well as being able to provide realistic animations of snow avalanches for feature films and games. Additionally, new instruments have been developed, making is possible to measure physical properties of real world snow avalanches. As a result of this focus, mathematical models based on the Navier-Stokes equations have been developed to describe the complex dynamic behavior of powder-snow avalanches.

The calculations of the governing equations requires a lot of computational power. This requirement can potentially be devastating for applications that have an essential interest in interactivity and real-time simulations. This time constraint introduce additional challenges to physics-based fluid simulation, and has been the main reason for its non-existence in computer games. The highly parallel architecture of the GPU, in addition to its evolution into a programmable unit, has changed the fluid simulation landscape in modern games, by making physics-based animation a reasonable alternative for numerical simulation.

The goal of this thesis is to develop a physics-based simulation of powder-snow avalanches. A mathematical model describing the complex dynamics of such a flow have recently been developed by Dutykh et.al [23], which is an extension of the Navier-Stokes equation for fluid flow. This model serve as the basis of the implemented simulation. In addition to providing a numerical solution to the governing equations, the simulation has been implemented on the GPU to introduce an acceleration of the computationally intensive tasks in the simulation.

1.2 Outline

The remainder of this thesis is divided into five chapters. The contents of these are as follows.

Chapter 2 provides a discussion of the work that are related to the problem of developing a simulation of powder-snow avalanches. It presents experimental work that have been done to create a better understanding of the snow avalanche flow, in addition to the mathematical models that have been developed to describe its motion. Some previous efforts to implement general fluid flow simulation on the GPU is also presented.

Chapter 3 describes the method applied for solving the problem of snow simulation. This chapter will introduce the Navier-Stokes equations, and how it relates to the problem of physics-based fluid simulation. The mathematical model governing powder-snow avalanche flow is then presented, before an SPH formulation is outlined to numerically solve this model. How the GPU has been integrated into the implementation is presented at the end of this chapter.

Chapter 4 and 5 provides a description of the SPH implementation of the numerical solution, as well as the results of the simulation.

Chapter 6 concludes this thesis with a discussion of the results obtained, and how the implementation may further be improved and extended in order to construct a solid and robust simulation framework for snow avalanche flow.

CHAPTER 2

Related Work

2.1 Snow Avalanche Modeling

An avalanche is a rapid gravity-driven mass of snow moving down slopes. It is a natural phenomenon that typically occurs in mountainous terrain. The reason for an avalanche release is the difference between the gravitational force acting on the top of the slope and the binding force holding the snow together. When fresh snow accumulates on older snow, the layered structure resulting from this is susceptible to internal slides between the layers, leading to av avalanche occurring. There are several factors involved in determining the cohesion between the layers, and thereby the risk of an avalanche happening, e.g temperature and wind.

When snow begins sliding down a mountain slope, its initial state is a dense flow. As air gets entrained within the snow particles of the dense flow, the avalanche will start to evolve into a powder-snow avalanche. In its intermediate state, the avalanche may consist of a dense underlying core of snow, above which a cloud of entrained snow and air particles reside. If this upper

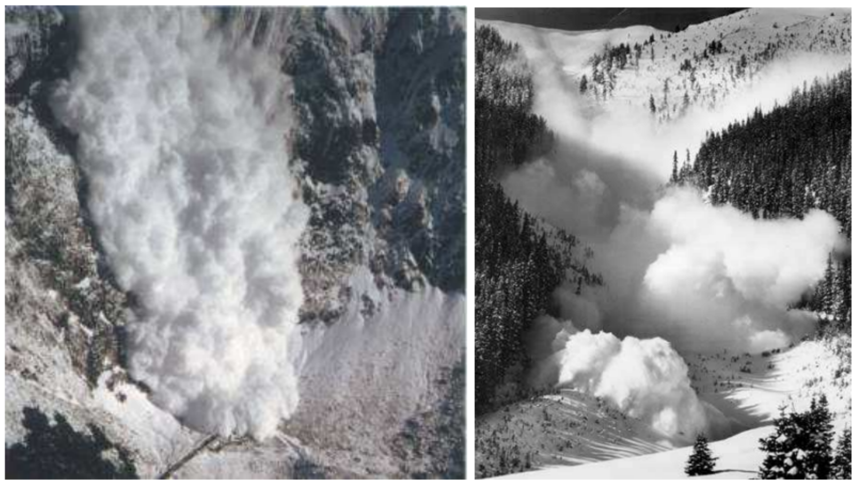


Figure 2.1.1 – Illustrations of powder snow avalanches.

cloud layer reaches a velocity higher than that of the underlying dense layer, the cloud will break away from the core, thereby creating a pure fully developed powder-snow avalanche.

In classifying avalanches, Ancey [4] proposed to only consider the form of motion of the avalanche, and not the quality of the snow. This leads to two limiting cases of avalanches; flowing avalanches and airborne avalanches.

Flowing avalanches are often referred to as dense-flow avalanches due to the high density core at the bottom of the flow. The depth is fairly small, and typical mean velocities ranges from 5 m/s to 25 m/s.

The airborne avalanche is a cloud of snow moving rapidly down slopes. The depth can become very large, and the mean velocity can exceed 100 m/s. Such avalanches are sometimes referred to as a powder-snow avalanche.

Several research experiments have been performed in order to determine the flowing properties of snow avalanches. These experiments have provided statistical and visual measurements that serves as a fundamental physical background for the mathematical models that have later been developed for describing snow avalanches. Two approaches have been used to investigate snow flow properties. One is artificial triggering of full-scale avalanches, while the other are small-scale experiments performed in laboratories.

For dense flow avalanches, laboratory experiments have been performed by [20, 73, 52], while full-scale experiments were performed by [42, 19], both with the goal of analyzing internal velocity profiles in rheological terms. Despite the different experimental approaches, the results obtained from these experiments were found to be very consistent. The velocity profiles gathered suggested the behavior of a yield stress fluid, which tended to lock up on regions where the shear stress in the snow is less than a given threshold value. This result prompted dense-flow avalanche models based on non-Newtonian fluid mechanics. As a consequence, models were presented representing dense-flow behavior using the Bingham [73], Herschel-Bulkley and Cross models [52], and the biviscous model [20]. Using these models, dense-flow avalanches can be described using simple constitutive equations.

Experiments on powder-snow avalanches are more difficult to perform than those regarding dense-flow. They are rare events, making full-scale approaches less viable. Additionally, even in laboratory settings, it is difficult to reproduce the actual physical powder-snow phenomenon. This is why most of the experimental setups consist of studying the dynamics of general turbidity currents, that only give pointers and approximations to the behavior of powder-snow avalanches. Such turbidity currents can be reproduced by dispersing a heavy fluid into a lighter fluid, e.g. salt suspension dispersed in water. The density ratio of such currents are in the range 1 to 2, whereas for powder-snow avalanches this ratio is in the order of 10. The experiments performed on powder-snow avalanches are not only concerned about the velocity profile, but also the advanced dynamics of the powder-cloud, including its flow behavior and internal structure.

Hopfinger et.al [45] and later Beghin [10] performed measurements of velocity and density profiles in two and three dimensions, respectively, of gravity

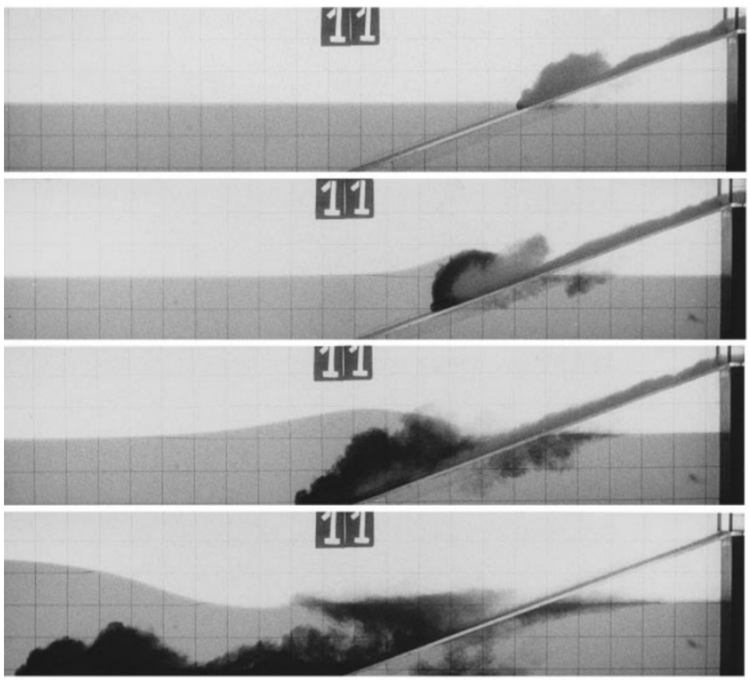


Figure 2.1.2 – Experimental setup on gravity currents performed by [65]. The tank is stratified with a lower layer of salt water, and an upper layer of fresh water. Image from [65].

currents using salt suspensions dispersed in water. They showed that it is possible to obtain laws about velocity and density of the flowing cloud, and how the height, length, and width growth-rates of the clouds were linear function of the slope.

Hermann and Hutter [44] simulated avalanches using polystyrene particles in still water to experiment with powder flow behavior at run-out zones, while Bozhinkiy [13] created a material using a mixture of ferromagnetic sawdust and aluminum dust, seeking better approximations to a natural powder-snow avalanche.

McElwaine [59] carried out ping-pong ball avalanche experiments to study three-dimensional granular flows. They measured individual ball velocities and air pressure using video cameras. They observed development of a complicated three-dimensional structure with a distinct head and tail. Additionally, they deduced the structure of the air flow around the avalanche using the data from air pressure measurements. The experiments provided detailed data to elucidate the dynamics of general two-phase granular flows of matter, and provided insights on the physically significant dynamical processes controlling avalanches.

2.1.1 Snow Avalanche Simulation

Simulating a snow avalanche is a challenging task, because of the complex nature of the phenomenon. The mathematical models that best describes the dynamics of an avalanche are based on the models describing regular fluid flow mechanics [2, 5]. These fluid flow models are essentially expansions of the Navier-Stokes equations, which will be described in the next section. This approach approximates the avalanche as a continuum, when it in fact consists of a composition of a wide range of particle sizes that may change with time and position. Despite this impediment to use a full fluid-mechanics approach to avalanche modeling, most of the models applying this approximation do make for some good simulations of snow avalanche flow.

Using this approach of modeling avalanche dynamics by relying on analogies with other physical phenomena, has resulted in avalanche models being analogous to granular flows [82, 88, 18], Newtonian fluids [47], power-law fluids [74], and viscoplastic flows [20, 3].

The complexity of different modeling schemes is determined by considering the different spatial scale of the avalanche. The simplest models consider the avalanche as a single unit of snow without deformation, and their goal is to calculate the most basic properties of an avalanche, including velocity and run-out distance. Considering the entire avalanche as a single unit will result in not being able to determine the intrinsic parameters that governs the behavior of an avalanche flow. The only parameter involved in these models is the friction coefficient between the snow pack and the ground. This friction

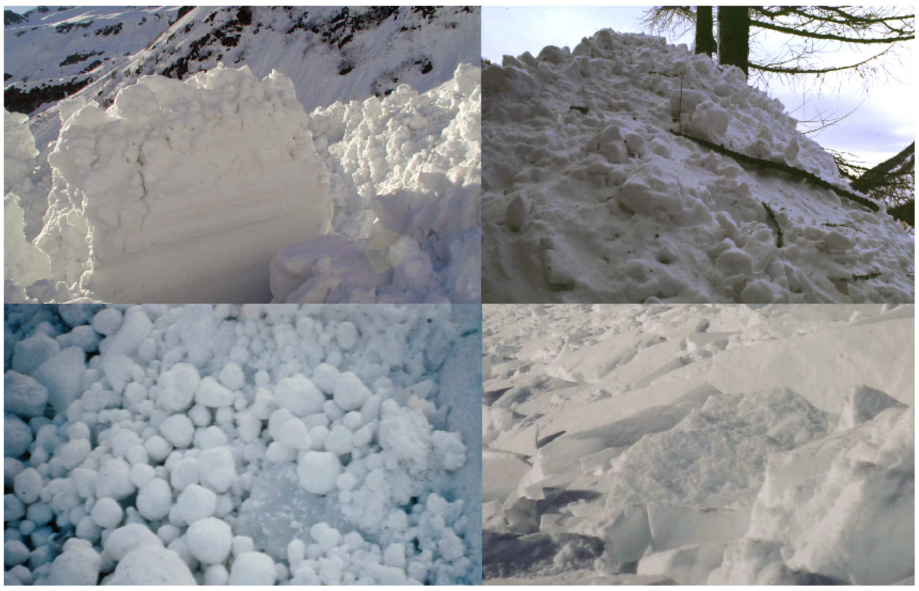


Figure 2.1.3 – Different types of snow observed in avalanche deposits. Even though a snow avalanche may consists of a composition of a wide range of particle sizes, the currently best numerical models for simulating the flow of an avalanche approximates the snow avalanche as a continuum substance. Image from [4]

coefficient is more a conceptual parameter than a physical one, since it is supposed to approximate the effects of more physical intrinsic parameters. The simplest models for dense-flow avalanches are derived from the original formulation by Voellmy [92], which belong the class of sliding-block models, leading to simple ordinary differential equations. The obvious drawbacks of using this model is the oversimplification of the physics of avalanches. Several models based on the Voellmy-model has been developed that seek to relax these simplifications [81, 79].

The simple models for powder-snow avalanches also consider the snow pack to be a single unit, but the amount of parameters is larger than that of dense-flow avalanches due to the more advanced dynamics of these flows. In addition to the friction coefficient, there are coefficients considering the air and snow entrainment and the shape of the avalanche. Kulikovskiy et.al [54] related the problem of modeling powder-snow avalanches to the modeling of

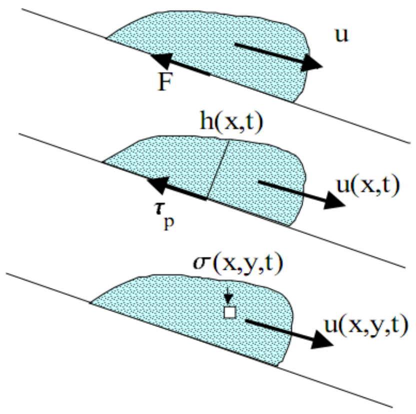


Figure 2.1.4 – Different spatial scales used for describing avalanches. Top: The simplest models consider the avalanche to be a non-deforming unit of snow with friction as the only opposing force. Middle: The intermediate models perform a depth-averaging procedure to accomplish better approximation of flow. Bottom: The complex numerical models consider the avalanche to be made up of particles, and simulate advanced dynamics based on the properties of them. Image from [4].

general particle clouds, and determined a simple theoretical model regarding the cloud to be a semi-elliptic body, whose volume vary with time. They obtained a set of four equations describing the mass-volume, momentum and kinetic energy balance. This theory is referred to as thermal theory, and has been redeveloped in subsequent papers, including [9, 8, 31, 11, 30, 3, 91].

On the opposite side of the spatial scale is the avalanche models considering snow avalanche behavior at the particle level. These models leads to complicated rheological and numerical problems, as the flow characteristics are computed at any point of the occupied space. Despite the computational complexity of these models, they are popular due to their detailed description of the flow dynamics. These models are highly influenced by the models described if hydraulics, which are based on the Navier-Stokes equations, as mentioned earlier.

The experiments presented earlier made for a consensus in the literature, that dense flowing avalanches can be described using non-Newtonian fluid models. Some of the models that were found to best capture the behavior of dense-flow avalanches are the Bingham model [73], the Herschley-Bulkley and Cross model [52], and the biviscous model [20]. What separates the flow behavior of snow avalanches from that of regular non-Newtonian fluid is the entrainment of snow from the base of the avalanche. This entrainment has been shown to strongly influence the dynamics of flowing avalanches [33, 86, 87], and recent literature deals with new models that take this entrainment into account [24, 12].

While the dense-flow avalanche models consists of approximating its behavior to that of simpler fluid flow, the dynamics of powder-snow avalanche are far too complex to make a direct analogy to phenomena within the field of hydraulics. As consequence of this, novel models have been determined to best capture the dynamics of powder-snow avalanches [26, 90, 23]. These models describe the behavior of turbulence flow by adding Fick's law for the diffusion process between air and snow.

As a compromise between the simple and the complex models, intermediate models have also been developed. They benefit from being less computationally expensive than three-dimensional numerical models and yet more accurate than simple ones. These intermediate models are sometimes referred to as depth-averaged models, because they are obtained by integrating the motion equations across the flow depth in a way similar to what is done in hydraulics for shallow water equations. This depth-averaging can be justified by making several assumptions about the flow of matter, for example incompressibility and small depth to length ratio of the flow.

For dense-flow, the most common model was developed by Savage and Hutter [82] and is referred to as the Savage-Hutter model. This model has subsequently been extended by [48, 39, 49]. For powder-snow flow, Parker [78] developed a complete depth-averaged model for general turbidity currents, that were later extended to take into account the effects of air entrainment in snow avalanches by Fukushima et.al [31].

2.2 Fluid Simulation Using GPU

There has been some research in recent years in order to develop fluid simulation algorithms that takes full advantage of the computational potential provided by the GPU.

Harada et.al [43] implemented Smoothed Particle Hydrodynamics on the GPU using OpenGL and the Cg shader language. This was done before the introduction of CUDA and the ability to program scientific calculations in a simple manner. During the time of their implementation, a solid knowledge of computer graphics and details of the graphics pipeline was necessary in order to utilize the computational ability of the GPU. Their implementation was one of the first to process every calculation of SPH on the GPU, removing any CPU-GPU memory transfer overhead.

Zhang et.al [93] presented an SPH implementation that supported adaptive sampling and rendering, all performed on the GPU. Both this algorithm and the one presented by Harada et.al [43] used a grid-based spatial subdivision structure to simplify the nearest neighbor search. Goswami et.al [37] achieved significant speedup and low memory consumption by applying a data structure based on Z-indexing instead of a spatial hashing method on a grid-based subdivision.

CHAPTER 3

Method

The process of implementing a physics-based simulation of a powder-snow avalanche begins with establishing a mathematical model describing the dynamics of the flow. As discussed in Chapter 2, there has been various approaches for establishing such mathematical models, the most detailed of which is based on 3D computational fluid dynamics. The model of choice for this thesis was presented by Dutykh et.al [23].

After a mathematical model has been provided, a numerical simulation is performed to provide a visual representation of the powder-snow avalanche. The numerical solver implemented is based on Smoothed Particle Hydrodynamics (SPH), a particle-based numerical method for approximating values and derivatives of continuous field quantities.

In order to achieve high performance simulation of the powder-snow avalanche, the incorporation of the GPU in the implementation is highly advantageous. The process of integrating the GPU in the simulation to deal with computations of resource-intensive tasks will conclude this chapter of the applied methods.

Before delving into the complex mathematical model describing the flow of a powder-snow avalanche, a discussion of general physics-based fluid simulation and its relevance in a wide range of areas is given. This discussion will present the Navier-Stokes equations, which are the basis for many mathematical models describing fluid flow, including the aforementioned model governing the flow of powder-snow avalanches. The Lagrangian formulation of the Navier-Stokes equations are then presented, which is the necessary formulation of the governing equations when applying a particle-based numerical solver like SPH.

3.1 Physics-based Fluid Simulation

A fluid is a substance that do not resist deformation, meaning it will flow when external forces are applied to it. The two most common fluid matters are gas and liquid. Gases expand to fill the container in which it is released, while liquids flow under the forces of gravity, eventually occupying the lowest regions of the container. Unlike gases, liquid will create free surfaces, the dynamics of which is not affected by the boundaries of the container. There are some physical phenomena that share the characteristics of gas and liquids. Smoke is a substance that is comprised of gas in combination with liquid particulates, a combination referred to as an aerosol. The flow of such phenomena is dependent on the surrounding container, while also having a distinct free surface. In physics-based fluid simulation, the goal is to best capture these physical characteristics , and thereby be able to create a realistic visualization that resembles the behavior of fluids.

There is a wide range of different areas where physics-based fluid simulation is applicable. These include feature films, commercial work, computer games, virtual environments, and medical simulation. The methods applied within these areas differ in the amount of realism that is required to provide a satisfactory result. In the feature film industry, the goal is to have the best match to the behavior of fluids, making the simulation indistinguishable from the real world. Within these fields, there exists a need to simulate at high resolutions, so as to capture all the intrinsic small-scale features of fluids, like splashes, droplets, etc. [57]. In its current state, the technology is such that detailed simulation can not be done in real-time, so areas like computer games has a different methodology when it comes to simulating fluids. Within this field the goal is real-time simulation and stability. For virtual reality applications and computer games, there is therefore a need for a reduction in the complexity of fluid dynamics, so as to be able to compute the behavior in real-time. These reductions include visual degradations by simulating at lower resolution, as well as coarser approximation of physical behavior to reduce computational complexity [69].

3.1.1 Navier-Stokes Equations

A flowing fluid has some basic physical quantities associated with it, namely velocity, density, and pressure. These quantities are considered as continuous fields in the fluid. How the velocity of the fluid changes as a function of time is governed by the Navier-Stokes equations, which describe the relationship between these quantities.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{3.1.1}$$

$$\rho(\frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v}) = -\nabla p + \nabla \cdot (2\mu \mathbb{D}(\mathbf{v})) + \mathbf{f} \tag{3.1.2}$$

where $\mathbf{v}$ is the velocity, ρ is the density, and p is the pressure of the fluid. μ is a measurement of the viscosity in the fluid, and $\mathbf{f}$ represents the sum of external forces acting on the fluid, e.g. gravity.

These two equations represents the classical form of the mass and momentum conservation equations for continuum mechanics, and they describe the motion of a regular fluid, e.g. air or water. Equation (3.1.2) is the momentum balance equation, and it describes how the fluid accelerates due to the forces acting on it. It is essentially Newton's second law stated in continuum mechanics. The two terms $-\nabla p$ and $\nabla \cdot (2\mu \mathbb{D}(\mathbf{v}))$ are internal forces, and are contributions to the total force that only arise from within the fluid. $-\nabla p$ represents the force induced by the differences in pressure in the fluid, and describes how the fluid flows from areas with high pressure to areas with low pressure. $\mathbb{D}(\mathbf{v})$ is the strain rate tensor and is defined as:

$$\mathbb{D}(\mathbf{v}) = \frac{1}{2}(\nabla\mathbf{v} + (\nabla\mathbf{v})^T)$$

Strain is a measure of deformation, and $\mathbb{D}(\mathbf{v})$ represents the rate at which deformation occurs when stress is applied. This rate may be linear, as is the case with regular Newtonian fluids, or non-linear, which would describe a non-Newtonian fluid. Additionally, viscosity μ is a measure of how much internal friction the fluid exhibits, so the entire term $\nabla \cdot (2\mu\mathbb{D}(\mathbf{v}))$ gives a description of the possibly complex behavior of the fluid when it is deformed.

Equation (3.1.1) is the mass continuity equation, and is a statement of the conservation of mass. The mass conservation is necessary in order to fully describe fluid flow. The general formulation as presented in equation (3.1.1) is specified for compressible fluids, and describes how the fluid may compress or expand in order to retain its total mass. By assuming that the fluid may not compress or expand, we state that the density of the fluid is constant. The fluid is then considered incompressible, and the mass continuity equation is expressed as a condition of the divergence of the velocity field, $\nabla \cdot \mathbf{v} = 0$. additionally assuming that the fluid is Newtonian, the strain rate tensor gets reduced to a simple laplacian operation. The complete Navier-Stokes equations of an incompressible, Newtonian fluid is then

$$\nabla \cdot \mathbf{v} = 0 \tag{3.1.3}$$

$$\rho(\frac{\partial\mathbf{v}}{\partial t} + \nabla\mathbf{v}\cdot\mathbf{v}) = -\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{f} \tag{3.1.4}$$

This formulation of the Navier-Stokes equations is based upon the entire fluid being composed of several fluid cells, aligned in a grid structure. Each of these cells has a fixed position in space, in addition to being associated with all the fluid properties. Figure (3.1.1) depicts how the fluid properties are connected to the fluid cells. This formulation of the Navier-Stokes is referred to as the Eulerian formulation, and expresses the problem of measuring the flow of fluid by observing the fluid at fixed points in space. To realize how

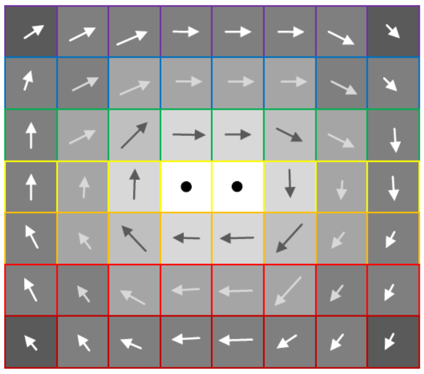


Figure 3.1.1 – In the grid-based Eulerian view, each cell in the grid has fluid properties associated with it, like velocity (arrows), density (cell fill), pressure (arrow color), and temperature (cell outline). Image from [38].

the formulation presented in equations (3.1.3) and (3.1.4) is based on this grid-based spatial subdivision, imagine a particle flowing with the fluid. If we want to measure the rate of change of velocity for this particle as it is flowing through the spatial domain, we have to first predict its position based on previously observed positional data, and then observe the velocity of this particle at its new cell position. Additionally, if the fluid is not considered stable, extra velocity of the particle needs to be predicted based on its acceleration. The velocity of a particle then depends on both time t and position $\mathbf{x}(t)$, which again depends on time. The full derivative of the velocity field then yields

$$\begin{aligned} \frac{d}{dt}\mathbf{v}(t, \mathbf{x}(t)) &= \frac{\partial \mathbf{v}}{\partial t} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}} \frac{d\mathbf{x}}{dt} \\ &= \frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v} \end{aligned}$$

This expression is present in the momentum conservation equation presented earlier.

3.1.2 Lagrangian Fluid Dynamics

The Eulerian formulation of the fluid flow is a good way of measuring dynamics of fluids which occupies the entire spatial domain, e.g. wind. It is, however, less robust when dealing with fluids that have a free surface, the flow of which do not occupy the entire computational domain. By representing the fluid using particles instead of grid-aligned cells, this free surface is handled naturally by the movement of the particles. This particle-based representation of fluid flow is called a Lagrangian representation. Unlike the grid-based Eulerian representation, the properties of the fluid is carried with the particles that make up the fluid. There is therefore not a need to predict a particles position, as it is associated with the particle, and calculated based on its velocity. The problem of finding the rate of change of velocity for a particle flowing through the computational domain, is then made easier, as the velocity now only depends on time. This leads to the Lagrangian formulation of the Navier-Stokes equations.

$$\nabla \cdot \mathbf{v} = 0$$

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

The denotation $\frac{D\mathbf{v}}{Dt}$ is referred to in the literature as the material derivative, and is defined as

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v} \tag{3.1.5}$$

To further the understanding of the two different formulations, one should realize that the velocity variable in the material derivative $\frac{D\mathbf{v}}{Dt}$ is referencing the velocity of some matter flowing with the fluid, whereas the velocity variable on the right hand side of the equations (3.1.5) is referencing the velocity field of the fluid domain. Stating that $\frac{D\mathbf{v}}{Dt} = 0$, specifies that any matter flowing with the flow is either at rest or moving with constant velocity. Stating that $\frac{\partial \mathbf{v}}{\partial t} = 0$, however, defines a constant velocity field, which may still determine a complex flow of matter within the fluid. The movement of the matter in this case is governed by the second term on the right hand side of equation (3.1.5). This term is called the advective term, and sometimes denoted $(\mathbf{v}\cdot\nabla)\mathbf{v}$, where $\mathbf{v}\cdot\nabla$ is known as the advection operator. $(\nabla\cdot\mathbf{v})\mathbf{v}$ then specifies that the velocity of any matter in the flow is advected by the velocity field, which may change over time depending on the value of $\frac{\partial \mathbf{v}}{\partial t}$. The material derivative is a way of linking the Eulerian and the Lagrangian formulations of the Navier-Stokes equations. The Lagrangian particle-based formulation of the governing equations for fluid flow is necessary when simulating fluids using Smoothed Particle Hydrodynamics, which will be presented later.

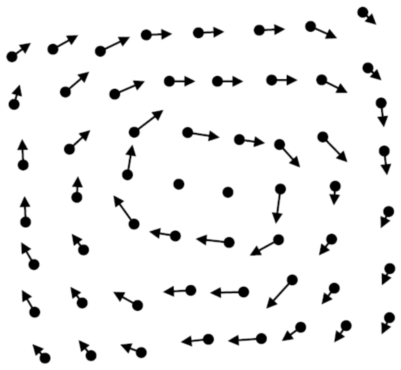


Figure 3.1.2 – In the Lagrangian view, each particle has a position and velocity, in addition to fluid properties following them. Image from [38]

3.2 Mathematical Model for Powder-snow Avalanches

During the discussion of powder-snow avalanche simulation in Chapter 2, it was realized that the evolution from a dense snow avalanche flow to a powder-snow avalanche is governed by the presence of air, and its entrainment in the snow flow. Such flows, whose behavior is governed by the presence of two different fluids is referred to as two-phase flows in fluid mechanics. The mathematical models applying the two-phase modeling approach for the simulation of powder-snow avalanches are the ones that provide the most complete information of the flow structure. However, by including the effects of a second fluid in the governing equations, they become computationally expensive, as the computational domain grows larger [71, 26, 90].

The introduction to the Navier-Stokes equations in the previous section dealt with the problem of finding a solution to a single vector field, namely the velocity field of a single-phase flow. The additional fluid in a two-phase flow would require a solution to two velocity fields, one for each of the fluids, leading to an increase in computation during simulation. This requirement may be removed by assuming that both phases are constrained to be governed by the same velocity. This assumption allows for the solution of a single velocity field, as in the case of a regular single-phase flow [70, 61, 60]. These single-velocity two-phase models have been successfully applied to tsunami waves [22] and breaking waves[14].

Dutykh et.al [23] presented such a single-velocity two-phase model for the simulation of avalanches in the aerosol regime. Their model allows for the two phases of the flow to interpenetrate, forming a mixing zone in the vicinity of the interface. This mixing process will introduce a stratification in the flow, allowing for the evolution of a powder-cloud. The derivation of the mathematical model begins with the Navier-Stokes equations in classical form, as presented in the previous section.

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

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \nabla \mathbf{u} \cdot \mathbf{u}) = -\nabla p + \nabla \cdot (2\mu \mathbb{D}(\mathbf{u})) + \mathbf{f}$$

The mixing of the two fluids is taken into account by Fick's type law [28], resulting in the following quasi-compressible equation

$$\nabla \cdot \mathbf{u} = -\kappa \nabla^2 \log \rho$$

where κ is the diffusion coefficient. These three equations represent the governing equations of the two phases making of the entire computational domain. This set of equations describes the flow of a compressible fluid. By rewriting these in terms of the single-velocity two-phase model, the system of equations will describe an incompressible fluid, which is simpler to solve numerically.

The single velocity variable governing the entire two-phase system is defined as

$$\mathbf{v} \equiv \mathbf{u} + \kappa \nabla \log \rho$$

and is referred to in the literature as the fluid volume velocity [15]. Using this new velocity variable, the system of governing equations are rewritten in terms of it. From Fick's law it is determined that the fluid in incompressible within the fluid volume velocity $\mathbf{v}$.

$$\nabla \cdot \mathbf{v} = 0$$

From this incompressibility condition, along with the assumption of the flow being a Newtonian fluid, the mass and momentum conservation equations in terms of the new velocity variable $\mathbf{v}$ becomes

$$\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho = \kappa \nabla^2 \rho \tag{3.2.1}$$

$$\begin{aligned} \rho(\frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v}) + \nabla p + \kappa \nabla(\mathbf{v} \cdot \nabla \rho) - \kappa(\mathbf{v} \cdot \nabla \rho)\frac{\nabla \rho}{\rho} \\ +\kappa^2 \frac{\nabla \rho}{\rho}\nabla^2 \rho - \kappa^2 \nabla^2 \nabla \rho - \kappa \rho (\nabla \log \rho \cdot \nabla)\mathbf{v} \\ -\kappa \rho (\mathbf{v} \cdot \nabla)\nabla \log \rho + \kappa^2 \rho (\nabla \log \rho \cdot \nabla) \nabla \log \rho \\ = \rho \mathbf{g} + \nabla \cdot (2\mu \mathbb{D}(\mathbf{v})) - \kappa \nabla \cdot (2\mu \nabla \nabla \log \rho) \end{aligned} \tag{3.2.2}$$

The diffuse term in the mass conservation equation (3.2.1) comes from the Fick's law governing the mixing process between two fluids. The momentum conservation equation (3.2.2) is substantially simplified in the model presented by [23], and their final set of governing equations are as follows, expressed in the Lagrangian formulation by using the material derivative (3.1.5)

$$\nabla \cdot \mathbf{v} = 0$$

$$\frac{D\rho}{Dt} = 2\bar{\nu}\nabla^2\rho$$

$$\rho\frac{D\mathbf{v}}{Dt} + \nabla\pi + 2\bar{\nu}(\nabla\mathbf{v})^T\nabla\rho - 2\bar{\nu}\nabla\mathbf{v}\nabla\rho = \rho\mathbf{g} + \nabla\cdot(2\mu\mathbb{D}(\mathbf{v}))$$

where the diffusion coefficient has been set to $\kappa = 2\bar{\nu}$. π is the a new definition of the pressure in the flow

$$\pi = p + 4\bar{\nu}\mu_0\nabla^2\log\rho$$

where p is calculated by an equation of state from the mixture density ρ. The remaining variables are mixture quantities, representing the physical properties of the mixture fluid originating from the interpenetration of snow and air in the avalanche flow.

The derivation of the mixture quantities, as presented by Dutykh [23], begins with the notion of a volume fraction. When considering two-phase flows, the volume fraction of a fluid is the fraction of the entire volume occupied by this fluid. In the case of powder-snow avalanches, the two fluids in question are snow and air. If the fraction of the volume occupied by the snow is $\phi \in [0, 1]$, then the volume fraction of the air is $1 - \phi$. By assuming constant densities and kinematic viscosity for both the snow and air, the mixture density ρ and the mixture dynamic viscosity μ are defined as

$$\rho = \phi\rho^+ + (1-\phi)\rho^-$$

$$\mu = \phi\rho^+\nu^+ + (1-\phi)\rho^-\nu^-$$

where $\rho^\pm$ and $\nu^\pm$ are the density and kinematic viscosity of the heavy and light fluid, respectively. The mixture dynamic viscosity may be expressed in terms of the density ρ as follows

$$\mu = \mu_0 + \bar{\nu}\rho$$

where μ_0 and $\bar{\nu}$ are related to the densities and kinematic viscosities of the two fluids in the following way

$$\mu_0 = \frac{\nu^- \rho^- \rho^+ - \nu^+ \rho^+ \rho^-}{\rho^+ - \rho^-}$$

$$\bar{\nu} = \frac{\nu^+ \rho^+ - \nu^- \rho^-}{\rho^+ - \rho^-}$$

3.3 Smoothed Particle Hydrodynamics

Smoothed particle hydrodynamics (SPH) is a mesh-free Lagrangian method, first created for simulating astrophysical problems [58, 35]. Since then it has been applied in numerous field for simulating continuum applications, including fluid dynamics and the problem related to free-surface flow [63]. SPH obtain approximate numerical solutions of the fluid dynamics equations by replacing the fluid with a set of particles, representing interpolation point from which fluid properties can be calculated.

SPH uses interpolation as the numerical technique for approximating the physical quantities present in a fluid. Interpolation means finding approximate values for a unknown function f based on the values of f at different points. The interpolation technique is based on the following identity

$$f(\mathbf{x}) = \int_\Omega f(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}' \tag{3.3.1}$$

where $f(\mathbf{x})$ is a function of the position vector $\mathbf{x}$, which is any point the in the domain Ω. $\delta(\mathbf{x}-\mathbf{x}')$ is the Dirac delta function, having the following two properties.

$$\delta(\mathbf{x} - \mathbf{x}') = \begin{cases} \infty & \mathbf{x} = \mathbf{x}' \\ 0 & \mathbf{x} \neq \mathbf{x}' \end{cases}$$

$$\int_{-\infty}^{\infty} \delta(\mathbf{x} - \mathbf{x}') = 1$$

While the identity (3.3.1) is exact, the Dirac delta function is not really a valid mathematical function and may therefore not be used for establishing numerical models. A smoothing function is introduced to provide an approximation for equation (3.3.1). This approximation is usually termed a kernel approximation, and the smoothing function W is referred to as a smoothing kernel function, or simply a kernel function. The kernel approximation of $f(\mathbf{x})$ becomes

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

where h is termed the smoothing length, and defines the area of influence of the smoothing function W, effectively determining how much of the domain surrounding position $\mathbf{x}$ will be used to approximate $f(\mathbf{x})$.

The purpose of the smoothing function W is to best mimic the behavior of the delta function $\delta(\mathbf{x}-\mathbf{x}')$. Monaghan [67] specifies that in order for it to do that, it has to satisfy the following two conditions

$$\int_{\Omega} W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}' = 1$$

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

Additionally, $W(\mathbf{x}-\mathbf{x}', h) = 0$ when $|\mathbf{x}-\mathbf{x}'| > h$. The first condition is the normalization condition, and is making sure that $\langle f(\mathbf{x})\rangle$ is scaled properly. The second condition is the Delta function property, and states that as the smoothing function approaches zero, W approaches δ, and the approximation $\langle f(\mathbf{x})\rangle$ approaches the correct value $f(\mathbf{x})$.

This integral interpolation method is a general numerical method for approximating a value from surrounding values. In order to apply this to the problem of approximating fluid quantities, the fluid density ρ is incorporated into the approximation integral in the following way

$$\langle f(\mathbf{x})\rangle = \int_{\Omega} f(\mathbf{x}')W(\mathbf{x}-\mathbf{x}', h)\frac{\rho(\mathbf{x}')}{\rho(\mathbf{x}')}d\mathbf{x}'$$

Then, by representing the fluid volume with a finite set of particles and realizing that the volume integral of density over the domain Ω is the mass of the fluid ($m = \int_\Omega \rho(\mathbf{x}')d\mathbf{x}'$), the integral is replaced by a summation of neighboring particles

$$\langle f(\mathbf{x}_i)\rangle = \sum_{j=1}^{N} f(\mathbf{x}_j)\frac{m_j}{\rho_j}W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.3.2}$$

where N is the total number of particles within the area of influence around the particle at position $\mathbf{x}_i$. Equation (3.3.2) states that the value of a function at a particle can be approximated by using the average value of the surrounding particles weighted by the smoothing function, and is at the center of the SPH approximation method.

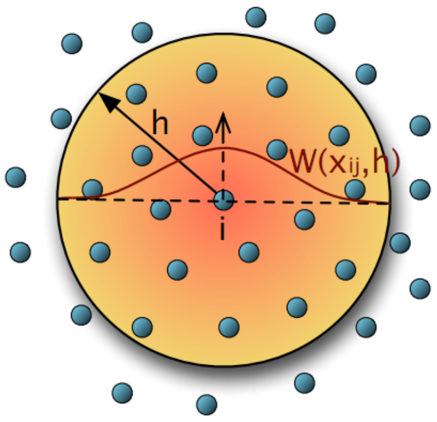


Figure 3.3.1 – In SPH, the quantities at each particle are smoothed over a neighborhood of size h. The weighted contributions of the neighboring particles are summed up, giving larger weight to closer particles than those farther away. Image from [85].

In order to fully describe fluid flow using SPH, there is a need to derive how the spatial differential operators are applied to the SPH formulation in equation (3.3.2). Below, it is only shown how the gradient operator $\nabla()$ is applied to the SPH formulation, but the same derivation holds for the divergence $\nabla \cdot ()$, the laplacian $\nabla^2()$, and the curl $\nabla \times ()$.

The gradient operator $\nabla f(\mathbf{x})$ is defined as $\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})$. Applying this operator to the SPH formulation in equation (3.3.2) yields

$$\frac{\partial}{\partial \mathbf{x}_i} \langle f(\mathbf{x}_i) \rangle = \frac{\partial}{\partial \mathbf{x}_i} [\sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} W(\mathbf{x}_i - \mathbf{x}_j, h)]$$

By using the product rule we get the following result

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}_i} [\sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} W(\mathbf{x}_i - \mathbf{x}_j, h)] &= \sum_j [\frac{\partial}{\partial \mathbf{x}_i} (f(\mathbf{x}_j) \frac{m_j}{\rho_j}) W(\mathbf{x}_i - \mathbf{x}_j, h) \\ &\quad + f(\mathbf{x}_j) \frac{m_j}{\rho_j} \frac{\partial}{\partial \mathbf{x}_i} W(\mathbf{x}_i - \mathbf{x}_j, h)] \\ &= \sum_j [0 \cdot W(\mathbf{x}_i - \mathbf{x}_j, h) + f(\mathbf{x}_j) \frac{m_j}{\rho_j} \frac{\partial}{\partial \mathbf{x}_i} W(\mathbf{x}_i - \mathbf{x}_j, h)] \\ &= \sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \end{aligned}$$

Since $f(\mathbf{x}_j)$ is not a function of $\mathbf{x}_i$, the directional derivative $\frac{\partial}{\partial \mathbf{x}_i} f(\mathbf{x}_j)$ is 0. We then end up with the fact that a differential operation on a function $f(\mathbf{x})$ is transformed into a differential operation on the smoothing function W. This means that we are able to approximate the spatial derivatives of a field function $f(\mathbf{x})$ by determining the values of this function and the derivatives of the smoothing function W, rather than from the derivatives of the field function itself. The following SPH kernel functions are the approximations of the spatial differential operators gradient, divergence, laplacian and curl of a field variable.

$$\begin{aligned} \langle \nabla f(\mathbf{x}_i) \rangle &= \sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \\ \langle \nabla \cdot f(\mathbf{x}_i) \rangle &= \sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} \cdot \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \\ \left\langle \nabla^2 f(\mathbf{x}_i) \right\rangle &= \sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{x}_i - \mathbf{x}_j, h) \\ \langle \nabla \times f(\mathbf{x}_i) \rangle &= \sum_j f(\mathbf{x}_j) \frac{m_j}{\rho_j} \times \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \end{aligned}$$

3.3.1 Smoothing Functions

The smoothing kernel functions determines both the accuracy and the computational efficiency of the SPH function representation, and plays therefore a very important role in the SPH approximation [1, 56, 32, 68]. There has been a large amount of research and investigation of the smoothing kernel in order to improve the performance of the SPH method. This section will introduce some of the most important generalized smoothing kernels that have been used. In the following presentation, $\mathbf{x}_{ij}$ is defined as the vector $\mathbf{x}_i - \mathbf{x}_j$, and r is defined as $\frac{\mathbf{x}_{ij}}{h}$.

A Gaussian kernel was applied to simulate non-spherical stars by Gingold and Monaghan in their original paper [35].

$$W(\mathbf{x}_{ij}, h) = \alpha e^{-r^2}$$

where α is $\frac{1}{\pi^{1/2}h}$, $\frac{1}{\pi h^2}$, and $\frac{1}{\pi^{3/2}h^3}$ in one-, two-, and three-dimensional space, respectively. Monaghan [62] considered this kernel to be a "golden rule" of SPH since it is very stable and accurate. Despite this consideration, however, this kernel is quite computationally expensive since it can take a long distance for the kernel to approach zero.

The cubic B-spline function was originally used by Monaghan and Lattanzio [66], and is the most widely used smoothing function

$$W(\mathbf{x}_{ij}, h) = \alpha \begin{cases} \frac{2}{3} - r^2 + \frac{1}{2}r^3 & 0 \leq r \leq 1 \\ \frac{1}{6}(2-r)^3 & 1 \leq r \leq 2 \\ 0 & r \geq 2 \end{cases}$$

where α is $\frac{1}{h}$, $\frac{15}{7\pi h^2}$, and $\frac{3}{2\pi h^3}$, in one-, two-, and three-dimensional space, respectively. It closely resembles a Gaussian function while having a more narrow compact support, making it less computationally expensive. The disadvantage of this kernel is that its second derivative is not a smooth function, making the laplacian unsuitable for use.

Morris [68] introduced two higher order splines that approximate the Gaussian more closely, in addition to being more stable. The first was the quartic spline

$$W(\mathbf{x}_{ij}, h) = \alpha \begin{cases} (r+2.5)^4 - 5(r+1.5)^4 + 10(r+0.5)^4 & 0 \le r \le 0.5 \\ (2.5-r)^4 - 5(1.5-r)^4 & 0.5 \le r \le 1.5 \\ (2.5-r)^4 & 1.5 \le r \le 2.5 \\ 0 & r \ge 2.5 \end{cases}$$

where α is $\frac{1}{24h}$, defined only in one-dimensional space. The second was the quintic spline

$$W(\mathbf{x}_{ij}, h) = \alpha \begin{cases} (3-r)^5 - 6(2-r)^5 + 15(1-r)^5 & 0 \le r \le 1 \\ (3-r)^5 - 6(2-r)^5 & 1 \le r \le 2 \\ (3-r)^4 & 2 \le r \le 3 \\ 0 & r \ge 3 \end{cases}$$

where α is $\frac{120}{h}$, $\frac{7}{478\pi h^2}$, and $\frac{3}{359\pi h^3}$, in one-, two-, and three-dimensional space, respectively.

Johnson [50] simulated high velocity impact problems by means of the following quadratic smoothing function

$$W(\mathbf{x}_{ij}, h) = \alpha(\frac{3}{16}r^2 - \frac{3}{4}r + \frac{3}{4})$$

where α is $\frac{1}{h}$, $\frac{2}{\pi h^2}$, and $\frac{5}{4\pi h^3}$ in one-, two-, and three-dimensional space, respectively. They made improvements over the cubic B-spline function by removing the problem of compressive instability. They did this by insuring that the derivative of the kernel function always increases as particles moves closer, and decreases when they move apart.

Liu et.al [56] constructed a quartic smoothing function that is more convenient and efficient to use, while still behaving very much like the cubic B-spline

$$W(\mathbf{x}_{ij}, h) = \alpha \begin{cases} \frac{2}{3} - \frac{9}{8}r^2 + \frac{19}{24}r^3 - \frac{5}{32}r^4 & 0 \leq r \leq 2 \\ 0 & r \geq 2 \end{cases}$$

where α is $\frac{1}{h}$, $\frac{15}{7\pi h^2}$, and $\frac{315}{208\pi h^3}$ in one-, two-, and three-dimensional space, respectively. Its improvement is a result of only using a single piece, instead of two.

Müller [69] designed special purpose smoothing kernels for the purpose of interactivity in their particle-based fluid simulation with free surfaces. They designed the following kernel

$$W_{poly6}(\mathbf{x}_{ij}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & r \geq h \end{cases}$$

This kernel is not, however, suitable for computation of the pressure forces because the gradient of the kernel approaches zero at the center, which results in the repulsion force vanishing. The pressure forces were computed using the spiky kernel by Desbrun et.al. [21]

$$W_{spiky}(\mathbf{x}_{ij}, h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3 & 0 \leq r \leq h \\ 0 & r \geq h \end{cases}$$

that generates the necessary repulsion forces.

Müller designed an additional kernel for the computation of viscosity forces

$$W_{viscosity}(\mathbf{x}_{ij}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & r \geq h \end{cases}$$

Its laplacian is positive everywhere, which removes the artifact resulting in a mistakenly increase in velocity when two particles get close to each other. The

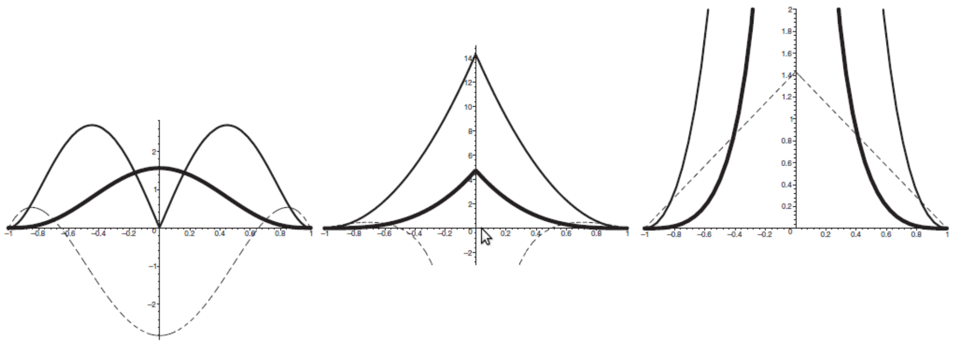


Figure 3.3.2 – The tree smoothing kernels applied to the SPH implementation by [69]. From left to right, there is W_{poly6}, W_{spiky}, and $W_{viscosity}$, including its gradients and laplacians, displayed in thin and dashed lines, respectively. Image from [69].

kernel functions used for the implementation of the powder-snow avalanche simulation will be the three presented by Müller [69].

3.3.2 Techniques for Deriving SPH Formulations

Having established the equations allowing us to numerically simulate fluid flow using SPH, it is now time to derive the SPH formulation of the governing equations presented earlier. Monaghan [67] presented an approach for deriving an SPH formulation of any partial differential equation. They used the following two identities for differential operators on scalar fields and vector fields, respectively

$$\frac{D_s(\varphi)}{\rho} = D_s(\frac{\varphi}{\rho}) + \frac{\varphi}{\rho^2}D_s(\rho) \tag{3.3.3}$$

$$\rho D_v(\mathbf{v}) = D_v(\rho\mathbf{v}) - \mathbf{v} \cdot D_s(\rho) \tag{3.3.4}$$

where ρ is the density of the fluid, φ is determines a scalar field, and $\mathbf{v}$ determines a vector field. $D_s()$ is a differential operator defined for scalar fields, namely the gradient $\nabla()$ and the laplacian $\nabla^2()$, while $D_v()$ is a differential operator defined for vector fields, consisting of the gradient $\nabla()$, the divergence $\nabla \cdot ()$, the curl $\nabla \times ()$, and the laplacian $\nabla^2()$.

This technique of placing the fluid density quantity inside the gradient operator is stated by Monaghan [67] to be the first golden rule of SPH. The reason for this consideration is that applying the above two identities will produce symmetric values between pair of particles, resulting in a conservation of linear and angular momentum.

The next step is then to apply the standard SPH formulation in equation (3.3.2) to each of the two terms on the right hand side of equation (3.3.3) and (3.3.4). This will lead to the following six SPH approximations for the relevant differential operations

$$\begin{aligned}
\langle \nabla \varphi \rangle &= \rho_i[\sum_j m_j(\frac{\varphi_i}{\rho_i^2} + \frac{\varphi_j}{\rho_j^2})\nabla W(\mathbf{x}_{ij}, h)] \in \mathbb{R}^3 \\
\left\langle \nabla^2 \varphi \right\rangle &= \rho_i[\sum_j m_j(\frac{\varphi_i}{\rho_i^2} + \frac{\varphi_j}{\rho_j^2})\nabla^2 W(\mathbf{x}_{ij}, h)] \in \mathbb{R} \\
\langle \nabla \cdot \mathbf{v} \rangle &= \frac{1}{\rho_i}[\sum_j m_j(\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla W(\mathbf{x}_{ij}, h)] \in \mathbb{R} \\
\langle \nabla \times \mathbf{v} \rangle &= \frac{1}{\rho_i}[\sum_j m_j(\mathbf{v}_i - \mathbf{v}_j) \times \nabla W(\mathbf{x}_{ij}, h)] \in \mathbb{R}^3 \\
\left\langle \nabla^2 \mathbf{v} \right\rangle &= \frac{1}{\rho_i}[\sum_j m_j(\mathbf{v}_j - \mathbf{v}_i)\nabla^2 W(\mathbf{x}_{ij}, h)] \in \mathbb{R}^3 \\
\langle \nabla \mathbf{v} \rangle &= \frac{1}{\rho}[\sum_j m_j(\mathbf{v}_j - \mathbf{v}_i) \otimes \nabla W(\mathbf{x}_{ij}, h)] \in \mathbb{R}^{3x3}
\end{aligned}$$

In addition to the these identities and formulations, there are some rules of operation that may be useful when deriving an SPH formulation for a complex system of equations [56]. The following two rules exist for two arbitrary function of field variables f_1 and f_2

$$\langle f_1 \pm f_2 \rangle = \langle f_1 \rangle \pm \langle f_2 \rangle$$

$$\langle f_1 f_2 \rangle = \langle f_1 \rangle \langle f_2 \rangle$$

These equations state that an SPH approximation of a sum of functions equals the sum of their individual SPH approximations, and likewise how the SPH approximation of the product of two functions equals the product of their individual SPH approximations.

The SPH approximation operator is a linear operator, as given by the following identity

$$\langle cf_1 \rangle = c \langle f_1 \rangle$$

Additionally, the SPH operator is commutative, giving the final two useful rules

$$\langle f_1 + f_2 \rangle = \langle f_2 + f_1 \rangle$$

$$\langle f_1 f_2 \rangle = \langle f_2 f_1 \rangle$$

By applying these SPH formulations, approximations to the system of equations describing powder-snow avalanche have been performed.

3.3.3 SPH Formulation of Governing Equations

The governing equations describing the dynamics of a powder-snow avalanche consists of three constitutive equations. They all depend on the density of the fluid, which an additional field quantity that has to be calculated using SPH approximation. The calculation of the density value at a particle are given by

$$\rho_i = \sum_j m_j W(\mathbf{x}_{ij}, h)$$

3.3.3.1 Incompressibility Condition

$$\nabla \cdot \mathbf{v} = 0$$

The incompressibility condition is a statement of how the fluid should behave, rather than a constitutive equation relating fluid quantities. It is expressed in terms of the divergence of the velocity field of the fluid continuum. The physical meaning of the divergence $\nabla \cdot \mathbf{v}$ is that the balance of outflow and inflow for a given volume element is zero at any time. Another way of thinking about such a divergence-free fluid flow is that there are no sources or sinks of fluid flow. In order to satisfy this incompressibility condition, the forces acting within the fluid continuum has to be modified to prevent an imbalance between outflow and inflow of a fluid volume element. More specifically, the force originating from the differences of pressure in the fluid has to be balanced.

There have been different strategies applied to enforce the incompressibility in particle-based fluid simulations. Incompressible SPH (ISPH) is the name given to methods modeling incompressible fluid flow by solving a pressure projection equation, first introduced in Eulerian methods [25]. These methods first integrate the velocity field in time without enforcing incompressibility, before projecting the solved field onto a divergence-free space through a pressure Poisson equation [55, 46]. These methods allow for large time steps in the simulation of fluid flow, but the iterative algorithms needed to solve the Poisson equation is very time consuming, leading to large computations at each time step.

Solenthaler et.al [84] introduced a method of enforcing incompressibility by applying a local prediction-correction scheme to determine the particle pressures (PCISPH). Their method involved a convergence loop at each time step, consisting of predicting velocity and density, followed by a correction of these based on a reference density. This method is less computational expensive at each integration step than ISPH, while still allowing for large time steps.

In the standard SPH model, the pressure is calculated from an equation of state (EOS) derived from the ideal gas law given by

$$pV = nRT \tag{3.3.5}$$

where p is pressure, V is volume per unit mass, n is the number of gas particles in mol, R is the universal gas constant, and T is the temperature. Assuming constant temperature and mass, the right hand side of equation (3.3.5) may be kept constant, and thereby replaced by a gas stiffness constant

k.

$$\begin{aligned} pV &= nRT \\ p(\frac{1}{\rho}) &= k \\ p &= k\rho \end{aligned}$$

Using this equation will lead to a large compressible behavior of the fluid, and is therefore not a appropriate choice for modeling incompressible flow. An extended EOS resulting in less compressibility was suggested by Desbrun et.al [21]

$$p = k(\rho - \rho_0)$$

where ρ_0 is the rest density of the fluid.

Weakly compressible SPH (WCSPH) is a model that applies an EOS that limits the amount of compressibility occurring in the fluid [64]. This model utilities the Tait equation, which is efficient to compute, while still enforcing an acceptable degree of incompressibility. The pressure is calculated as follows

$$p = \frac{k\rho_0}{\gamma}((\frac{\rho}{\rho_0})^{\gamma} - 1) \tag{3.3.6}$$

where k is a stiffness parameter and ρ_0 is the reference density. γ is the adiabatic index, a physical measurement of the ratio of the heat capacity at constant pressure to heat capacity at constant volume in a fluid. Numerically, this variable is determined experimentally, and are usually set to 7. Using WCSPH for approximating incompressible fluid flow will lead to an efficient computation without degrading the simulation a great deal, and will be therefore be the choice of model for the simulation of powder-snow avalanche flow.

3.3.3.2 Mass Conservation

$$\frac{D\rho}{Dt} = 2\bar{\nu}\nabla^2\rho$$

The mass conservation equation is a statement of how the density of the fluid changes as the fluid flows. In the classic formulation of the incompressible Navier-Stokes equations (equation 3.1.3 and 3.1.4), this mass conservation is not present. This is because the incompressibility condition implies that $\frac{D\rho}{Dt} = 0$, and it may therefore be omitted. In the model describing powder-snow avalanche flow, there will be a density change as a consequence of the fluid mixing governed by Ficks's law of diffusion, and the mass conservation equations therefore reappears. Applying the aforementioned formulations to determine the SPH approximations yields

$$\langle 2\bar{\nu}\nabla^2\rho \rangle = 2\bar{\nu}\rho[\sum_j m_j(\frac{1}{\rho_i} + \frac{1}{\rho_j})\nabla^2 W(\mathbf{x}_{ij}, h)]$$

3.3.3.3 Momentum Conservation

$$\rho\frac{D\mathbf{v}}{Dt} + \nabla\pi + 2\bar{\nu}(\nabla\mathbf{v})^T\nabla\rho - 2\bar{\nu}\nabla\mathbf{v}\nabla\rho = \rho\mathbf{g} + \mu\nabla^2\mathbf{v}$$

The momentum conservation equations governs the actual movement of the particles and relates the acceleration of each particle to the forces that acts upon it. As argued during the discussion of the incompressibility condition, the pressure in a fluid is more of a balancing force, and should therefore not contain additional terms that affect the flow of the fluid. An appropriate formulation of the momentum balance equation is then derived by rewriting the equation in terms of the regular pressure variable p. The second term of the alternative pressure definition $\pi = p + 4\bar{\nu}\mu_0\nabla^2 \log\rho$ is then placed in the momentum balance equation, resulting in the following formulation

$$\rho\frac{D\mathbf{v}}{Dt} + \nabla p + 2\bar{\nu}(\nabla\mathbf{v})^T\nabla\rho - 2\bar{\nu}\nabla\mathbf{v}\nabla\rho = \rho\mathbf{g} + \mu\nabla^2\mathbf{v} - 4\bar{\nu}\mu_0\nabla\nabla^2\log\rho$$

By reformulating the momentum balance equation, an expression similar to what was determined when presenting the incompressible, Newtonian Navier-Stokes equations is derived.

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

where $-\nabla\pi$ and $\mu\nabla^2\mathbf{v}$ represents the internal forces in the fluid flow, while $\mathbf{f} = \rho\mathbf{g} - 2\bar{\nu}(\nabla\mathbf{v})^T\nabla\rho + 2\bar{\nu}\nabla\mathbf{v}\nabla\rho - 4\bar{\nu}\mu_0\nabla\nabla^2\log\rho$ are the external forces governing the complex dynamics of the flow.

Internal Forces There are two SPH approximations performed on the internal forces. There is the pressure force $-\nabla p$, and the viscosity force $\mu\nabla^2\mathbf{v}$. The following two SPH approximations corresponds to these two terms

$$\begin{aligned}
\langle -\nabla p\rangle &= -\rho[\sum_j m_j(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2})\nabla W(\mathbf{x}_{ij}, h)] \\
\left\langle \mu\nabla^2\mathbf{v}\right\rangle &= \mu\frac{1}{\rho}[\sum_j m_j(\mathbf{v}_i - \mathbf{v}_j)\nabla^2 W(\mathbf{x}_{ij}, h)]
\end{aligned}$$

External Forces The seemingly complex expression making up the external forces governing the fluid flow originates from the mixing process that is occurring between the snow and the air in the powder-snow avalanche flow. The differential terms in the expression that needs to be approximated consists of $\nabla\mathbf{v}$, $\nabla\rho$, and $\nabla\nabla^2\log\rho$. In the last term, the evaluation of $\nabla^2\log\rho$ needs to be performed before the calculation of the entire term. The following are the SPH approximation of the external forces, where $\alpha = \nabla^2\log\rho$.

$$\begin{aligned}
\langle\nabla\rho\rangle &= \rho[\sum_j m_j(\frac{1}{\rho_i} + \frac{1}{\rho_j})\nabla W(\mathbf{x}_{ij}, h)] \\
\langle\nabla\mathbf{v}\rangle &= \frac{1}{\rho}[\sum_j m_j(\mathbf{v}_j - \mathbf{v}_i)\otimes W(\mathbf{x}_{ij}, h)] \\
\left\langle\nabla^2\log\rho\right\rangle &= \rho[\sum_j m_j(\frac{\log\rho_i}{\rho_i^2} + \frac{\log\rho_j}{\rho_j^2})\nabla^2 W(\mathbf{x}_{ij}, h)] \\
\langle\nabla\alpha\rangle &= \rho[\sum_j m_j(\frac{\alpha_i}{\rho_i^2} + \frac{\alpha_j}{\rho_j^2})\nabla W(\mathbf{x}_{ij}, h)]
\end{aligned}$$

3.3.4 Time Integration

After calculating the acceleration of each particle, they are advanced through time using a global fixed time step Δt. This advancement is done by calculating new positions and velocities of the particles based on Δt. How this time integration is performed may greatly affect the quality and the stability of the simulated system.

One of the most common numerical time integration schemes is the semi-implicit Euler technique. This technique is based on the explicit Euler technique, which is the most basic method for numerical integration. In explicit Euler, the advancements of position and velocity are dependent only on their previously calculated values, and their new values are calculated in parallel as follows

$$\begin{aligned} \mathbf{x}_{t+\Delta t} &= \mathbf{x}_t + \Delta t \mathbf{v}_t \\ \mathbf{v}_{t+\Delta t} &= \mathbf{v}_t + \Delta t \mathbf{a}_t \end{aligned}$$

Despite its simple nature, the explicit Euler method suffer from inaccuracy and instability in systems with rather complicated physical behavior. For more complex models, the semi-implicit Euler method is a better option, as it is more stable than its predecessor. This technique is not only dependent on the previously calculated values, but also the current. The position of a particle is calculated by evaluating the velocity at the current time step.

$$\begin{aligned} \mathbf{v}_{t+\Delta t} &= \mathbf{v}_t + \Delta t \mathbf{a}_t \\ \mathbf{x}_{t+\Delta t} &= \mathbf{x}_t + \Delta t \mathbf{v}_{t+\Delta t} \end{aligned}$$

This technique provides a simple, yet effective way of advancing the system through time.

3.4 GPU Computing

The graphics processing unit (GPU) has experienced a rapid increase in both performance and capabilities over the last couple of years. They provide large memory bandwidth and computational power, in addition to applying advanced processor technologies. Researchers and developers have become interested in taking advantage of this power for general-purpose computing. As a result, development has been made to improve the programmability of the GPU to make it a compelling platform for computationally demanding tasks in a wide variety of application domains. The term General-Purpose GPU (GPGPU) has been popularized as the effort to make the GPU an alternative to the traditional CPU as a general-purpose computation unit for high-performance computer systems.

This section will introduce GPGPU and the reason for its integral part in certain problem domains. Additionally, to be able to program efficiently on the GPU, its architecture has to be understood.

3.4.1 GPGPU

At the heart of GPU computing is the highly parallel characteristic of the graphics pipeline. Any application that exhibits data parallelism can be fully exploited by the graphics hardware and thereby allowing for higher performance than would have been achieved by a CPU. These applications include tasks such as sorting, image processing, linear algebra, and physics-based simulation.

Different applications, and even different phases of a single application, place unique and distinct demands on computing resources. Before developing a specific application on the GPU platform, one should first realize that not every application will benefit from the parallel architecture available on this platform. Asanovic et.al [6] surveyed the issues that are present when dealing with parallelism, and argued that successful parallel platforms should perform well on 13 classes of problems, which they termed dwarfs. Che et.al [16] examined three of these dwarfs implemented on a CPU, a GPU, and a FPGA. By comparing developments costs and performance, they concluded that GPU's excel at parallel workload with deterministic memory access. Several applications within different scientific fields satisfying these constraints have been successfully implemented on the GPU with great performance increase [80, 53, 77].

3.4.2 GPU Hardware Architecture

The GPU used to be a fixed-function special purpose processing unit designed for use in computer games. Its function was to compute the colors of the pixels of an output image on the screen in parallel. In recent years the focus has been shifted toward the programmable aspect of the GPU, and the hardware architecture has experienced an evolution centered around a large number of parallel processors with great arithmetic capability.

The graphics pipeline is a term referring to the method of producing a 2D raster image from a representation of the 3D scene by use of graphics hardware. The different stages in this pipeline perform specific operations on the geometric primitives created by the 3D application.

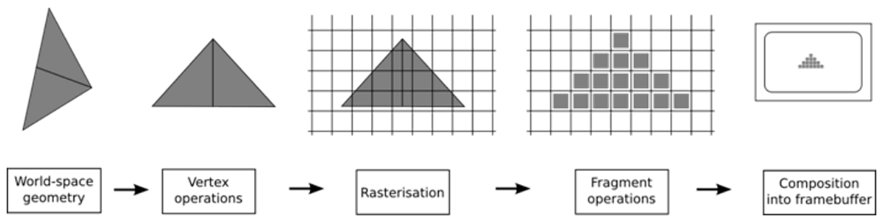


Figure 3.4.1 – The classic graphics pipeline. Image from [34]

These operations exhibit large data parallelism, as the task performed on each stage is executed on a large amount of data, e.g. vertices and pixels. To execute such a pipeline effectively with high throughput, the GPU divides the processing resources among the different stages, as well as within each stage. Using this organization, the GPU is capable of meeting the large computational needs presented by the graphics pipeline. Each task at the different stages could further be made more efficient by creating special purpose hardware for the operations present at the stage.

This fixed-function special purpose GPU architecture was implemented for graphics acceleration, and has evolved substantially over its lifetime. In recent years, the changes have become more dramatic, as the GPU has transitioned from a fixed-function to a programmable architecture.

The fixed-function pipeline was not able to efficiently express complicated shading and lighting operations. As a solution to this, the stages dealing with the processing of vertices and fragments was made programmable, thereby allowing for programs to be written to introduce a more general processing of vertices and fragments, making the development of games more exciting. These programs was called shaders and introduced the programming model called the shader model.

Even though the stages of the graphics pipeline became more programmable, the separation of tasks into stages was still a major disadvantage of the graphics pipeline. The separation introduced the problem of load balancing which limited the performance of the GPU pipeline, making it dependent on its slowest stage.
The introduction of the unified shader model converged the instruction sets of the vertex and fragment programs, making the introduction of a single programmable hardware unit inevitable. This unified shader architecture makes the programmable units divide their time among vertex, fragment, and geometry processing. This architecture is present in modern day graphics hardware, and is what makes GPGPU viable, by allowing developers to target the single programmable unit, rather than dividing work across multiple hardware units.

3.4.3 OpenCL

As an effort to develop applications that take full advantage of the parallelism present in modern processor architectures, several general purpose parallel programming models have been introduced. OpenCL is one of these, and support programming across CPU's, GPU's, and other processors. OpenCL consists of a programming language, in addition to an API for controlling and coordinating computation across heterogeneous processors.

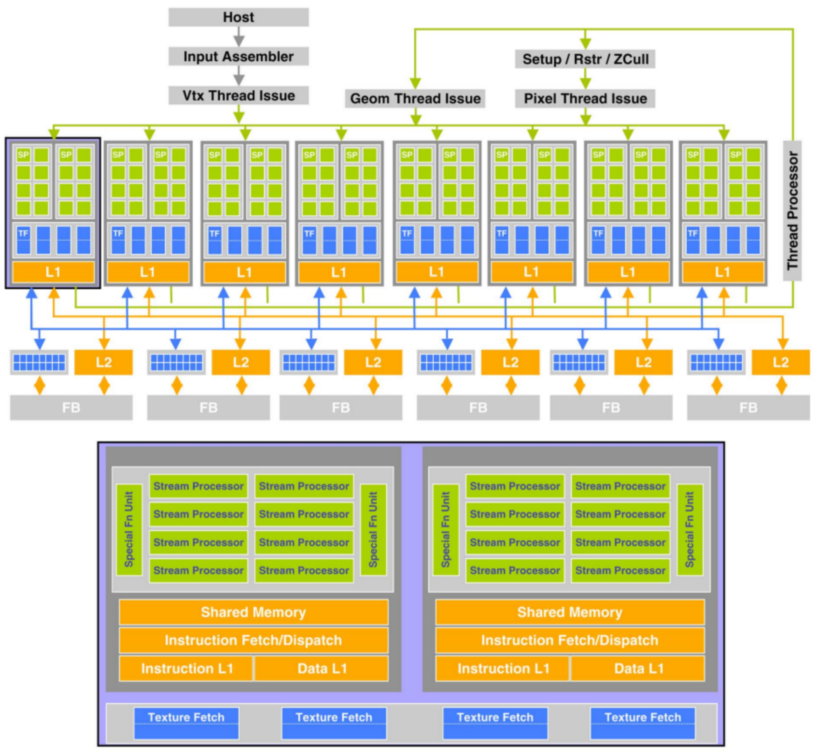


Figure 3.4.2 – GPU architecture from NVIDIA with massively parallel programmable units at the core. Image provided by NVIDIA

This section will introduce the OpenCL framework for parallel programming, describing the model architecture, consisting of the platform model, the execution model, the memory model, and the programming model.

3.4.3.1 Platform Model

The platform model is the conceptual model of how the underlying hardware is presented to the programmer. The model represent the system by dividing it into one host connected to one or more devices. These devices acts as co-processors to the host. They are subdivided into one or more compute units, which are further divided into one or more processing elements. It is within these processing elements the computations occur.
An OpenCL application runs on the host, which sends commands to the device that are to be executed on the processing elements within the device.

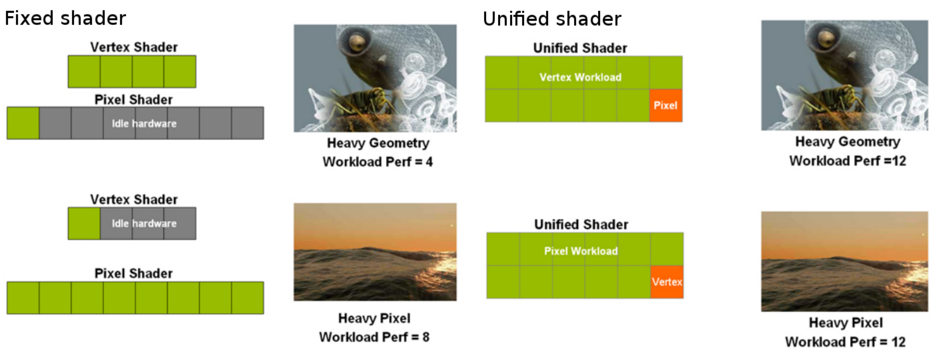


Figure 3.4.3 – The benefit from using a unified shader architecture. Programmable units divide their time among vertex, fragment, and geometry processing, leading to less idle hardware and larger throughput. Image from [75]

3.4.3.2 Execution model

The commands sent by the host to the device are instructions written in special function called kernels. The kernels execute on one or more devices, and they are managed by a host program that executes on the host.

An instance of a defined kernel is executed for each item in an index space. For a data parallel programming model, this index space is made up of the elements in a memory object containing the different data associated with the OpenCL application. The kernel instance is called a work-item, and there is a one-to-one mapping between the work-item and a data element. Each work-item then executes the same code in parallel, each operating on different data.

Work-items may be arranged into work-groups, which will provide a different organization of the index space. Each work-group is assigned a single compute unit, and the work-items within the group run concurrently on the processing elements present in the compute unit. This distribution of work-items into work-groups is important when synchronization between work-items is necessary. The OpenCL index space is called an NDRange, and is an N-dimensional index space, where N is 1, 2, or 3.

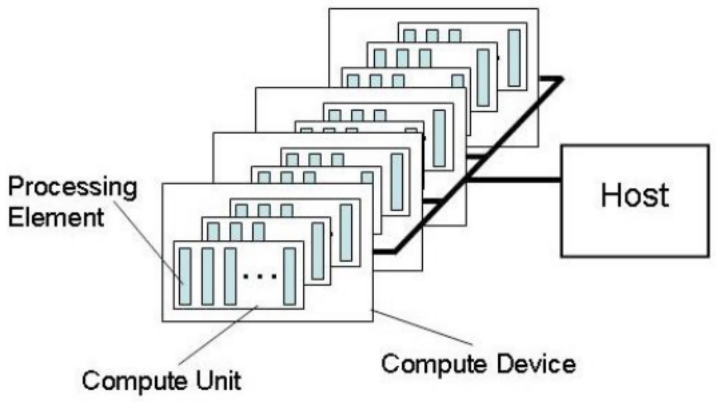


Figure 3.4.4 – The OpenCL platform model. There is one host and one or more computer devices. Each compute device is made up of one or more compute units, which again is made up of one or more processing elements. Image from [41]

3.4.3.3 Memory Model

The device memory domain is divided into four distinct memory regions, which work-items has access to. The **global memory** can be thought of as the main memory of the device. This region permits reads and writes access to all work-items. **Constant memory** is similar to global memory, except work-items may not write to this memory. It remains constant during the execution of the kernel. **Local memory** is a memory region local to a work-group, and shared by the work-items within this group. Finally, **private memory** is private to each work-item, and may not be accessed by another work-item. As memory access latency is one of the most important performance inhibitors of a computer application, the understanding and utilization of this memory query is vital in high performance computing.
A kernel can neither access host main memory nor dynamically allocate global memory. Memory management is therefore done by the host, which allocate memory blocks in global or constant memory, in addition to copying data to and from these blocks. In most cases, the host copies all input data to the device memory domain prior to kernel execution, and all output data back to the host memory domain afterward. Any data transfer between

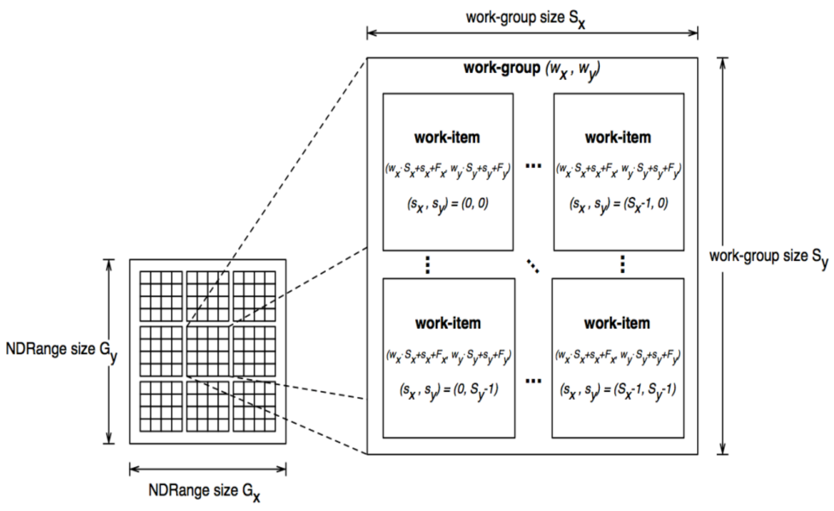

work-group size S_x
work-group (w_x, w_y)
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, 0)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, 0)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, S_y-1)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, S_y-1)$
work-group size S_y
NDRange size G_y
NDRange size G_x

Figure 3.4.5 – Example of an NDRange index space showing work-items, their global IDs and the mapping onto the pair of work-group and local IDs. Image from [41]

host and device during the execution of a kernel will greatly decrease the performance of the application.

3.4.3.4 Programming Language.

The OpenCL programming language is a variant of the C99 language optimized for GPU programming. The language is used just to write the performance or data-intensive routines in an application, making the transition to a GPU application of a previously implemented program much easier for developers. These OpenCL kernel are compiled for the GPU on the fly.

3.5 Smoothed Particle Hydrodynamics on GPU

Smoothed particle hydrodynamics needs a large amount of particles in order to achieve fine-scale flow details and smooth surfaces. Because of this, the method has in the past been less popular in applications requiring interactive simulation of fluid flow. Due to the ease with which computations on particle systems can be made parallel, the massively parallel computational capabilities of the modern GPU has recently been taken advantage of, so as

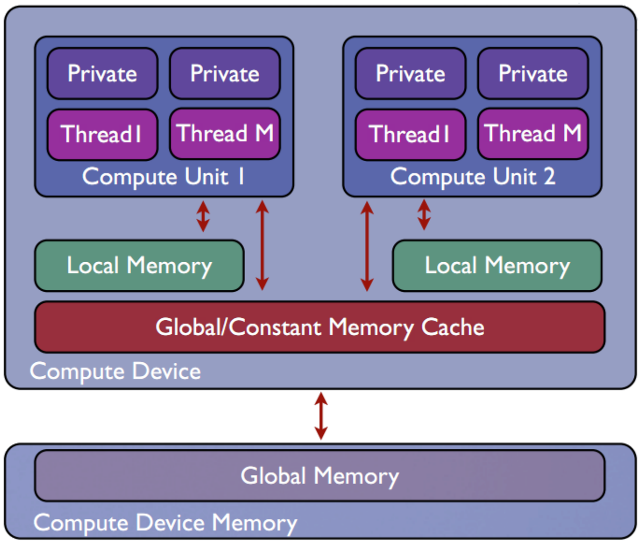


Figure 3.4.6 – The four address spaces that the work items has access to. Image from [36]

to make SPH more attractive for simulations of large systems at interactive rates. A GPU implementation of SPH will result in the simulation of detailed physics-based fluid flow being performed in hours rather than days.

The computationally most expensive part in SPH simulation is the neighborhood search that has to be performed for each particle at every iteration. This evaluation of the nearby particles may be performed in the most naïve way, by iterating through all particles in the simulation domain. The time complexity of such an approach in a simulation containing n particles is bound by $O(n^2)$, which leaves room for improvements. An important realization is that the only particles that contribute to the calculation of fluid quantities in SPH are the ones that are in close proximity to the particle being evaluated. For this reason, the neighborhood search can be made easier and faster by dividing the simulation domain into a uniform grid, which allows for the time complexity to be reduced to $O(mn)$, where m is the average number of particles found at neighboring grids.

The goal is then to combine the computational capabilities of the GPU with the time complexity decrease provided by the spatial subdivision technique. The GPU implementation is performed using OpenCL, a general purpose parallel programming model, that makes it possible to develop applications that take full advantage of the parallelism present in modern processor architectures.

CHAPTER 4

Implementation

The SPH method has been implemented to simulate powder-snow avalanche dynamics. The simulation consists of two main procedures, which are done at each time step. First, there is the creation of the grid-based spatial subdivision. A more detailed explanation of how the grid-based spatial subdivision is performed is presented, in addition to a discussion of its vital part in the nearest neighbor search. The second procedure presented is the calculation of the governing equations of the system that are made to determine the force applied to each particle. The details of the simulation loop is outlined, before discussing the determination of the physical parameters.

4.1 Uniform Spatial Subdivision

There are two main operations that are to be performed when using a spatial subdivision technique. There is the creation of the grid data structure, and there is the evaluation of neighboring grid cells. The spatial subdivision is performed using a uniform grid, which is the simplest possible spatial subdivision. Using a uniform grid will subdivide the three-dimensional spatial domain into a grid of equally-sized cells. Each cell will contain a number of particles, and the neighborhood search is performed by iterating through the 27 cells that are surrounding the cell currently being examined. The cell sizes may be determined by the smoothing length defined in the SPH kernel function, which will provide an effective iteration through every particle in the nearby cells. The grid data structure is constructed every iteration by using a spatial hashing method. This method was performed by Green [40] in his particle simulation using CUDA. Green build the data structure using sorting, which provides a simple and effective way of creating the grid data structure, in addition to improving memory coherence when accessing the grid.

The cells in the grid are then evaluated using a spatial hashing method [89], which allows for cell evaluation by assigning it a hash value based on its position in the grid. Before explaining the procedures involved in the construction of the spatial subdivision, some parameters detailing the setup of the simulation environment needs to be determined. These include the world size, the grid size, the cell size, and the particle size.

The **world size** is the size of the three dimensional spatial domain in which the simulation is performed, determining the boundaries of the simulation environment. In addition to the world size, the world origin needs to be determined to be able to calculate the cell index of a particle.

The **grid size** determines the amount of cells contained within the simulation environment, whereas **cell size** is the actual size of each of these cells. Using a single fixed cell size will result in a uniform grid structure. The cell size is calculated by dividing the world size by the grid size.

Finally, the **particle size** needs to be determined. The size of a particle should be set so as to fit within a cell. By allowing for several particles to be contained within a cell, a better approximation to the fluid quantities will be made, due to the large amount of particles involved in the evaluation of a quantity. There will be a performance tradeoff, however, as more computation needs to be done at each time step.

Having determined the simulation environment parameters, the actual construction of the spatial subdivision is performed by three implemented procedures: hash calculation, hash table sorting, and particle reordering.

4.1.1 Hash Calculation

The hash value given to a particle is determined from the cell in which it is contained. The cell index is calculated from the position $\mathbf{p}$ of the particle using the following function

$$cell\_index_{\{x,y,z\}} = \left\lfloor (p_{\{x,y,z\}} - world\_origin_{\{x,y,z\}})/cell\_size_{\{x,y,z\}} \right\rfloor$$

Using this function, the cell index is specified by means of Cartesian coordinates in the space defined by the grid, which makes determining neighboring cells very easy.

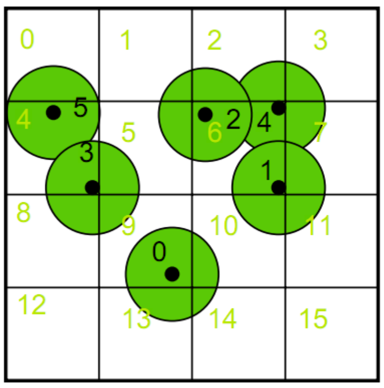
The hash value of a particle is then calculated by using the hash function presented by Green [40].

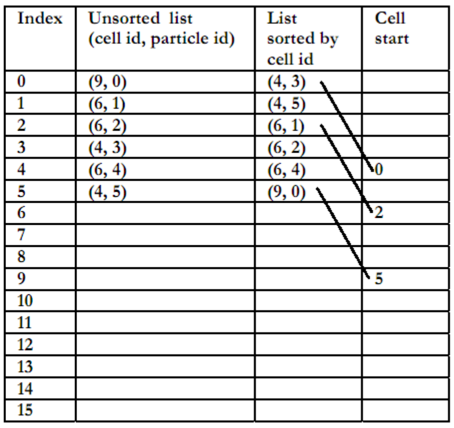
$$hash(\mathbf{p}) = (z \cdot gridsize_y + y) \cdot gridsize_x + x$$

which provides the linear cell id as the hash value. As it stands, this function assumes that the cell size is strictly determined by the grid size, as described earlier.

4.1.2 Hash Table Sorting

Sorting is performed on the hash values associated with the particles. This sorting procedure will create a list of particles ids in cell order, as depicted in Figure (4.1.1).





Index	Unsorted list (cell id, particle id)	List sorted by cell id	Cell start
0	(9, 0)	(4, 3)	
1	(6, 1)	(4, 5)	
2	(6, 2)	(6, 1)	
3	(4, 3)	(6, 2)	
4	(6, 4)	(6, 4)	0
5	(4, 5)	(9, 0)	
6			2
7			
8			
9			5
10			
11			
12			
13			
14			
15			

Figure 4.1.1 – The grid structure containing particles within cells, and the sorted list used for effective cell boundary determination. Image from [40].

The sorting procedure is performed using an implementation of bitonic sort on the GPU provided by NVIDIA [76].

4.1.3 Particle Reordering

After sorting the particles on the hash value, the sorted list is used to find the start and end particle indices for each cell. This is done by comparing each particles hash value with the hash value of the previous particle in the sorted list. If these hash values are identical the two particles belong to the same cell, and if they are different, the currently evaluated particle is contained within the next cell in the linearly ordered grid. Two one-dimensional arrays are used to store the indices of the particles related to a cell: *cell_start* and *cell_end*. These arrays are indexed by the hash value of a cell, and contains the indices of the first and last particle that are part of that cell, respectively. The arrays containing the particle quantities is then reordered into sorted order to improve the coherence of the texture look-up during the particle interaction procedure.

4.1.4 Nearest Neighbor Search

Due to the use of the grid-based spatial subdivision, the particle neighborhood is easily and quickly iterated through. First, neighboring cells are determined by subtracting and adding 1 to the Cartesian coordinate defining the cell index. This procedure is outlined in Algorithm (4.1).

Algorithm 4.1 Cell Loop

```
1: for $l = -1 \to 1$ do
2:     for $k = -1 \to 1$ do
3:         for $j = -1 \to 1$ do
4:             $nbcell_x = cell_x + j$
5:             $nbcell_y = cell_y + k$
6:             $nbcell_z = cell_z + l$
7:         end for
8:     end for
9: end for
```

Then the particles within these cells are evaluated by using the start and end particle indices determined in the particle reordering procedure explained previously. The cell iteration procedure is outlined in Algorithm (4.2).

Algorithm 4.2 Cell Iteration

```
1: procedure CellIteration($cell$)
2:     $hash\_value \leftarrow getHash(cell)$
3:     $start\_index \leftarrow cell\_start[hash\_value]$
4:     $end\_index \leftarrow cell\_end[hash\_value]$
5:     for $j = start\_index \to end\_index$ do
6:         $determine\ particle_j\ contribution$
7:     end for
8: end procedure
```

4.2 Calculation of Governing Equations

The SPH formulations for calculating the governing equations were presented in the previous chapter. The implementation of these calculations consists of three iterations over the particles at each simulation step. During the first iteration the density of the fluid particles is calculated. Then the pressure is evaluated based on the particles densities. During the calculation of the pressure, the quantity $\nabla^2 \log \rho$ is determined. The reason for this is that this quantity must be determined for every particle before evaluating the forces, which are calculated during the third iteration. The general fluid simulation algorithm is presented in Algorithm (4.3).

Algorithm 4.3 Fluid Simulation

```
 1: initialize()
 2: while animating do
 3:     for all particles i do
 4:         calcHash(i)
 5:     end for
 6:     sort()
 7:     for all particles i do
 8:         findCellBounds(i)
 9:     end for
10:     for all particles i do
11:         computeDensity(i)
12:     end for
13:     for all particles i do
14:         computePressure(i)
15:     end for
16:     for all particles i do
17:         computeForces(i)
18:     end for
19:     for all particles i do
20:         integrate(i)
21:     end for
22: end while
```

The procedures in lines 3 $\rightarrow$ 9 are related to the inclusion of the spatial subdivision of the computational domain, and were explained in the previous section. The *initialize*() procedure has two purposes: Determining the parameters of the simulation environment, as well as initializing the physical parameters of the particle system representing the fluid. The initialization of the physical parameters are discussed in the next section. In this section, the procedures performed at lines 10 $\rightarrow$ 21 as presented, relating to the actual calculation of the governing equations describing powder-snow avalanche flow.

4.2.1 Density Computation

The *computeDensity*() procedure is outlined in Algorithm (4.4). At each iteration over the particles the density is recalculated using the SPH approximation previously presented

$$\rho_i = \sum_j m_j W(\mathbf{x}_{ij}, h)$$

The density variation that occur within each particle will govern the calculation of the pressure that is needed to balance the forces in the fluid, so as to make the fluid incompressible. The new density that are calculated for a particle is compared to the reference density of the fluid. If there is a variation between these two values, compressible behavior has occurred, and the pressure that is calculated from this difference will seek to balance out this compressible artifact. This pressure computation is described in the next section.

Since the mass conservation equation describes a change in density that is suppose to happen, the recalculation of the density has to take this into account. This is done by storing both the rest density and a separate reference density for each particle. The reference density will change over time according to the mass conservation equation, and the difference between this density and the rest density of the fluid needs to be subtracted from the recalculation of the particle density.

4.2.2 Pressure Computation

The *computePressure*() procedure is outlined in Algorithm (4.5). The pressure computation is made to balance out unwanted compressibility in the

Algorithm 4.4 Density Computation

```
1: procedure ComputeDensity($i$)
2:     $\rho \leftarrow 0.0$
3:     for all neighbours $j$ do
4:         $\rho \mathrel{+}= m_j W_{poly6}(\mathbf{x}_{ij}, h)$
5:     end for
6:     $\rho_d \leftarrow \rho_0 - \rho_d$
7:     $\rho_i \leftarrow \rho - \rho_d$
8: end procedure
```

fluid. It is calculated using the Equation of State described in equation 3.3.6. This equation is presented using the rest density ρ_0 as the density to which the calculated density is compared. The newly introduced reference density should be used for this purpose in this case. The force that originates from the differences in pressure, will now be modified to counteract any compressibility.

In addition to calculating the pressure, this iteration through the particles will calculated the necessary value $\nabla^2 \log \rho$ for each particle, that is needed for the computation of the forces in next iteration.

Algorithm 4.5 Pressure Computation

```
1: procedure ComputePressure($i$)
2:     $\nabla^2 \log \rho \leftarrow 0.0$
3:     for all neighbours $j$ do
4:         $\nabla^2 \log \rho \mathrel{+}= m_j(\frac{\log \rho_j}{\rho_j^2} + \frac{\log \rho_i}{\rho_i^2})\nabla^2 W_{poly6}(\mathbf{x}_{ij}, h)$
5:     end for
6:     $p_i \leftarrow \frac{k\rho_r}{\gamma}((\frac{\rho}{\rho_r})^\gamma - 1)$
7:     $\alpha_i \leftarrow \rho_i \nabla^2 \log \rho$
8: end procedure
```

4.2.3 Force Computation

The *computeForce*() procedure is outlined in Algorithm (4.6). In addition to the computation of the internal and external forces on a particle, a density

Algorithm 4.6 Force Computation

```
1: procedure ComputeForces($i$)
2:     $\nabla p \leftarrow \mathbf{0.0}$
3:     $\nabla \alpha \leftarrow \mathbf{0.0}$
4:     $\nabla \rho \leftarrow \mathbf{0.0}$
5:     $\nabla^2 \mathbf{v} \leftarrow \mathbf{0.0}$
6:     $\nabla^2 \rho \leftarrow 0.0$
7:     $\nabla \mathbf{v} \leftarrow \mathbf{0.0}$
8:     for all neighbours $j$ do
9:         $\nabla^2 \rho \mathrel{+}= m_j(\frac{1}{\rho_j} + \frac{1}{\rho_i})\nabla^2 W_{poly6}(\mathbf{x}_{ij}, h)$
10:        $\nabla p \mathrel{+}= m_j(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2})\nabla W_{spiky}(\mathbf{x}_{ij}, h)$
11:        $\nabla \alpha \mathrel{+}= m_j(\frac{\alpha_j}{\rho_j^2} + \frac{\alpha_i}{\rho_i^2})\nabla W_{poly6}(\mathbf{x}_{ij}, h)$
12:        $\nabla \rho \mathrel{+}= m_j(\frac{1}{\rho_j} + \frac{1}{\rho_i})\nabla W_{poly6}(\mathbf{x}_{ij}, h)$
13:        $\nabla^2 \mathbf{v} \mathrel{+}= m_j(\mathbf{v}_j - \mathbf{v}_i)\nabla^2 W_{viscosity}(\mathbf{x}_{ij}, h)$
14:        $\nabla \mathbf{v} \mathrel{+}= m_j(\mathbf{v}_j - \mathbf{v}_i) \otimes \nabla W_{poly6}(\mathbf{x}_{ij}, h)$
15:    end for
16:    $\mathbf{f}_i^a \leftarrow -\nabla p$
17:    $\mathbf{f}_i^b \leftarrow \mu \nabla^2 \mathbf{v}$
18:    $\mathbf{f}_i^c \leftarrow -4\bar{\nu}\mu_0 \nabla \alpha$
19:    $\mathbf{f}_i^d \leftarrow -2\bar{\nu}(\nabla \mathbf{v})^T \nabla \rho$
20:    $\mathbf{f}_i^e \leftarrow -2\bar{\nu}\nabla \mathbf{v} \nabla \rho$
21:    $\mathbf{f}_i \leftarrow \mathbf{f}_i^a + \mathbf{f}_i^b + \mathbf{f}_i^c + \mathbf{f}_i^d + \mathbf{f}_i^e + \rho \mathbf{g}$
22:    $\Delta \rho \leftarrow 2\bar{\nu}\nabla^2 \rho \Delta t$
23:    $\rho_r \leftarrow \rho_r + \Delta \rho$
24: end procedure
```

change is performed, in correspondence with the mass conservation equation.

To be able to calculate the quantity $\nabla \mathbf{v}$, matrix operations has to be performed. These operation are not supported by OpenCL, and had to be implemented into the simulation environment. These operations include the outer product of two vectors, matrix-vector multiplication, matrix-scalar multiplication, and matrix transpose.

4.2.4 Time Integration

The *integrate*() procedure is outlined in Algorithm (4.7). This procedure

Algorithm 4.7 Time Integration

```
1: procedure INTEGRATE($i$)
2:     $\mathbf{v}_i \leftarrow \mathbf{v}_i + \mathbf{a}_i \Delta t$
3:     $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \Delta t$
4: end procedure
```

updates the velocity and position vectors of a particle, by means of the implicit Euler technique described in the previous chapter. During this step, a collision detection is also performed to determine whether some particles are outside the simulation domain. The current simulation environment is a rectangular box, so these collision detections are simple *if*-statements on particle positions. A no-slip condition is imposed, which means that the velocities of the particles colliding with the bottom plane of the rectangle are set to zero.

4.3 Initialization of Simulation System

The particle system representing the fluid is initialized by determining the physical parameters that are associated with the fluid. These parameters include particle mass, gas stiffness, and viscosity coefficient. In addition to these physical parameters, the smoothing radius of the kernel approximation function, as well as the simulation time step, has to be determined. The physical parameters associated with a numerical simulation does not have a perfect one-to-one mapping with the real world, due to the different spatial scales the simulation may be performed in. Most of the parameters needs to be determined experimentally, but the real world physical domain can provide a determination of some of the relevant parameters.

4.3.1 Particle Mass

The volume V of a specified fluid is determined by its density ρ and mass m in the following manner

$$V = \frac{m}{\rho}.$$

By representing the fluid with n number of particles, each with its fixed mass m', the equation can be rewritten as follow

$$V = \frac{nm'}{\rho}.$$

If we then specify the volume and the density of the fluid we wish to simulate, in addition to how many particles we will use to represent it, the mass of each particle can be determined by

$$m = \frac{\rho V}{n}.$$

The volume and density of a fluid are usually the quantities that are provided when simulating a specified fluid. One may then choose to specify the mass of each particle in the simulation system, and from that determine how many particles are needed to provide a physical accurate result of the fluid flow. However, since the amount of particles has a great effect on the computational complexity of the simulation, this is the normally the predefined parameter that are provided to the system.

4.3.2 Gas Stiffness and Simulation Time Step

During the discussion of incompressibility in the previous chapter, a pressure calculation was determined based on a gas stiffness constant k. This gas stiffness can be compared to a spring constant in a spring system. The equivalent of the spring system in computational fluid dynamics is the pressure force that arise from the differences in pressure. For this reason, the accuracy of the fluid flow is greatly dependent on the choice of this constant. The value of the gas stiffness constant is theoretically given by

$$k = nRT$$

as presented in the previous chapter. This formula is not appropriate to use in numerical simulations, however, as it will result in a very large value for k. In addition to being an important parameter in the fulfillment of the incompressibility condition, the stiffness constant will, if made too large, have a great impact on the simulation system. Large values of k will result in a stiff simulation system, which will result in numerical instability, due to a rapid variation in the solutions made each time step. Small time steps are then needed to prevent the simulation from “exploding”. A tradeoff is then presented, as the required large values for k in the incompressibility condition will result in small time steps, and thereby a slow simulation, which will reduce the interactivity of the system. The stiffness constant is therefore resolved by a tuning performed by the animator.

4.3.3 Viscosity Coefficient

The viscosity coefficient the dynamic viscosity μ, and is a physical measurement of how viscous the fluid is, that is how resistant the fluid is to deformation. Numerically, this coefficient provides a factor of stability to the simulated system, by providing a damping variable, and should be tuned in correspondence with the stiffness parameter.

4.3.4 Smoothing Length

The issue regarding the smoothing length of the kernel approximation function has been integrated into the issue of deciding the cell size of the spatial subdivision. Since local interaction between particles is evaluated by only iterating through the particles contained in the 27 surrounding cells, the connection between these two parameters is obvious. Regardless of which parameter is determined by the other, the size of area around a particle used to determine the fluid quantities has an impact of both the stability and the robustness of the fluid simulation. One might think that a larger radius of influence will always lead to more precise evaluation of the fluid quantities. This is not the case, however, as a large support radius will at certain instances lead to non-uniform weighting of surrounding particle values. The intuition of how different support radii is affecting the evaluation of fluid quantities is depicted in Figure (4.3.1). The support radius can be determined by predefining the average amount of particles one would want when evaluating a fluid quantity, and should be determined based on the total amount of particles in the system.

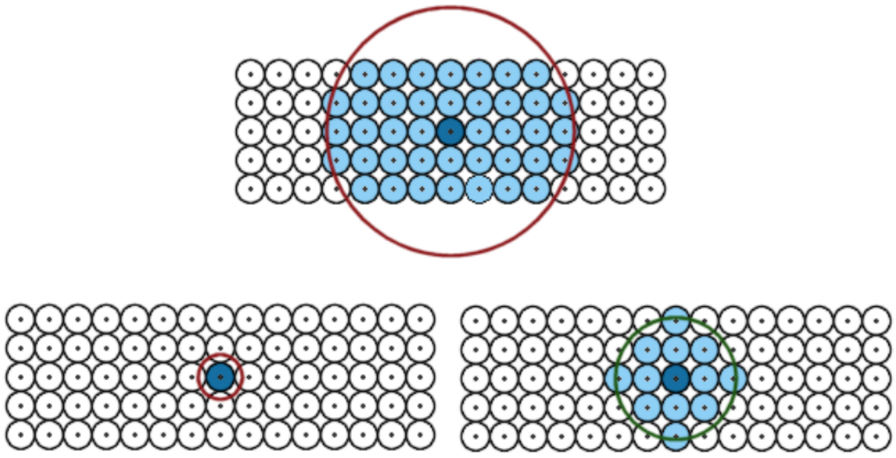


Figure 4.3.1 – The different effects from choosing different smoothing lengths for the kernel approximation function. Top: A large support radius may give a bad weight distribution on the surrounding particles. Bottom left: A small support radius may not provide enough averaging data. Bottom right: An appropriate support radius is important when evaluating fluid quantities. Image from [51]

CHAPTER 5

Results

This chapter will provide visual result from my implementation of the powder-snow avalanche flow.

The implemented model provides different simulation variables that are available to produce different effects of the fluid flow. These include the volume fraction parameter and the density difference between the light fluid and the heavy fluid. The volume fraction parameter is a measurement of how much of the entire volume is made up of the heavy fluid, and will therefore determine how dense the flow should be. By specifying a larger density difference between the heavy and the light fluid, the fluid flow should behave more turbulent. These different set of parameters have been employed to depict the different developments of the fluid flow, and what effect the different simulation parameters have on the flow structure. For every simulation the incline of the simulation environment is set to 30^o, to represent a downhill flow. The number of particles in the current simulation is set to 32768.

The two parameters having the greatest effect on the fluid flow is the volume fraction ϕ and the density difference between the heavy fluid ρ^+ and the light

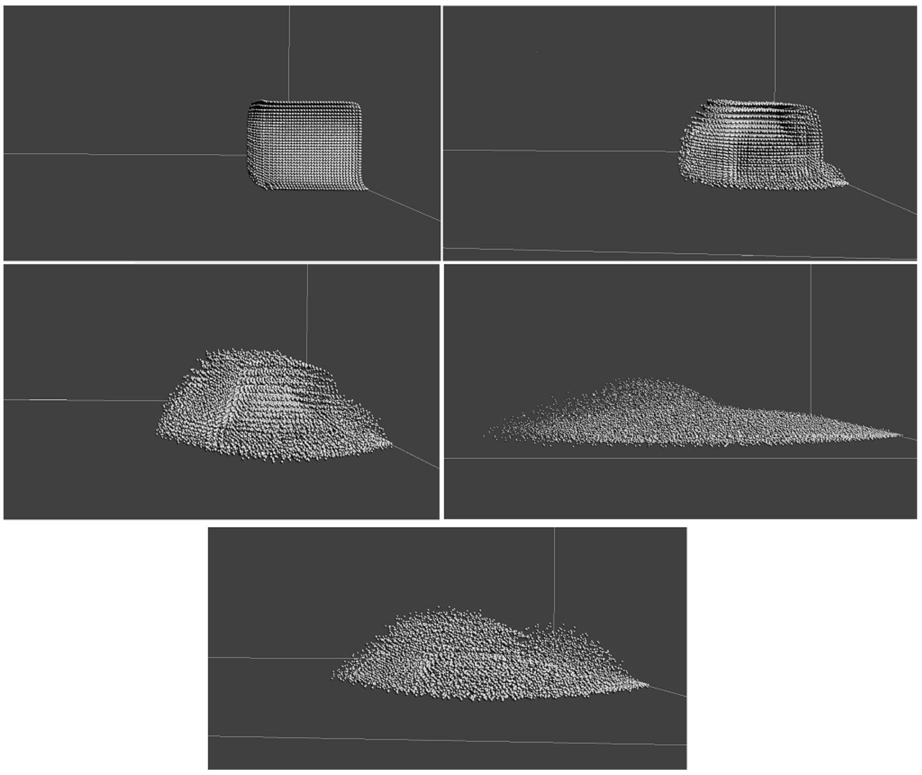


Figure 5.0.1 – Simulation evolution for $\phi = 0.4$, $\rho^+ = 90\frac{kg}{m^3}$, $\rho^- = 5\frac{kg}{m^3}$

fluid ρ^-. Figure (5.0.1) shows the development of flow having the following parameter values

$$\begin{aligned}\phi &= 0.4\\ \rho^+ &= 90\frac{kg}{m^3}\\ \rho^- &= 5\frac{kg}{m^3}\end{aligned}$$

The evolution of an area being less dense is observed at the outer regions of the flow. The potential turbulence is dropping of as the simulation evolve over time. The limited amount of particles used in the simulation neglect the continuation of a more cloud-like flow.
Figure (5.0.2) show the simulation performed using a different set of param-

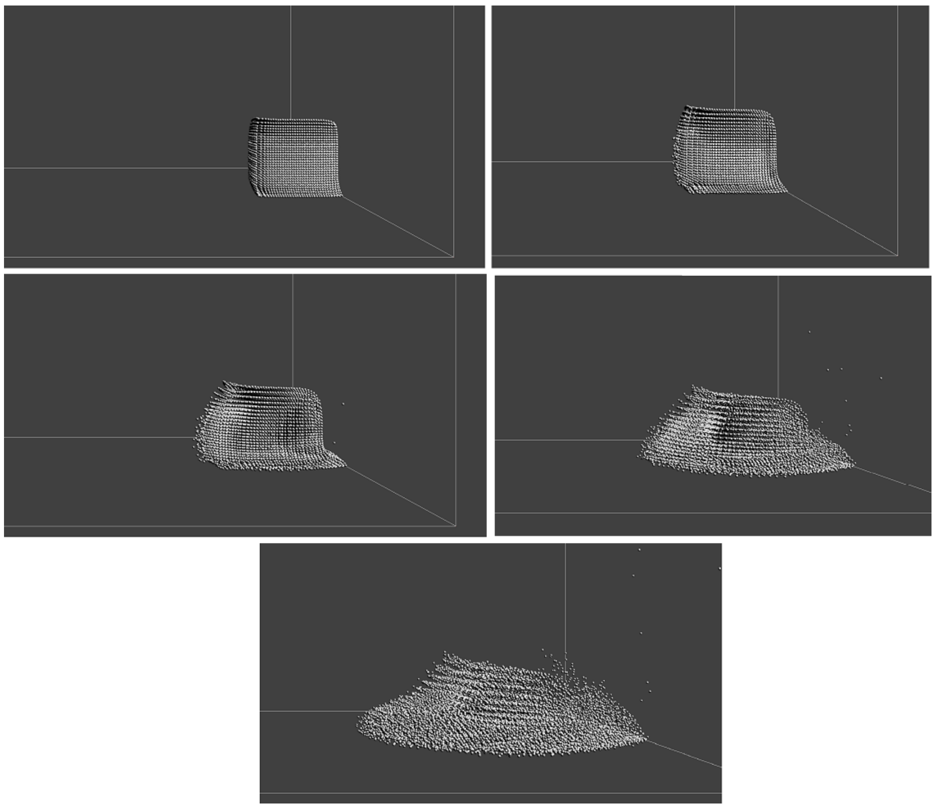


Figure 5.0.2 – Simulation evolution for $\phi = 0.4$, $\rho^{+} = 950\frac{kg}{m^3}$, $\rho^{-} = 10\frac{kg}{m^3}$

eter values

$$\begin{aligned} \phi &= 0.4 \\ \rho^{+} &= 950\frac{kg}{m^3} \\ \rho^{-} &= 10\frac{kg}{m^3} \end{aligned}$$

The larger difference in densities between heavy and light fluid are shown to result in a more turbulent initiation of the flow. The denser part of the flow at the bottom is seen to flow faster than that of the upper layer, giving indication of lighter fluid at the top.
The final set of visual result shows the effects of varying the volume fraction parameter governing the influence of the heavy fluid in the mixture substance.

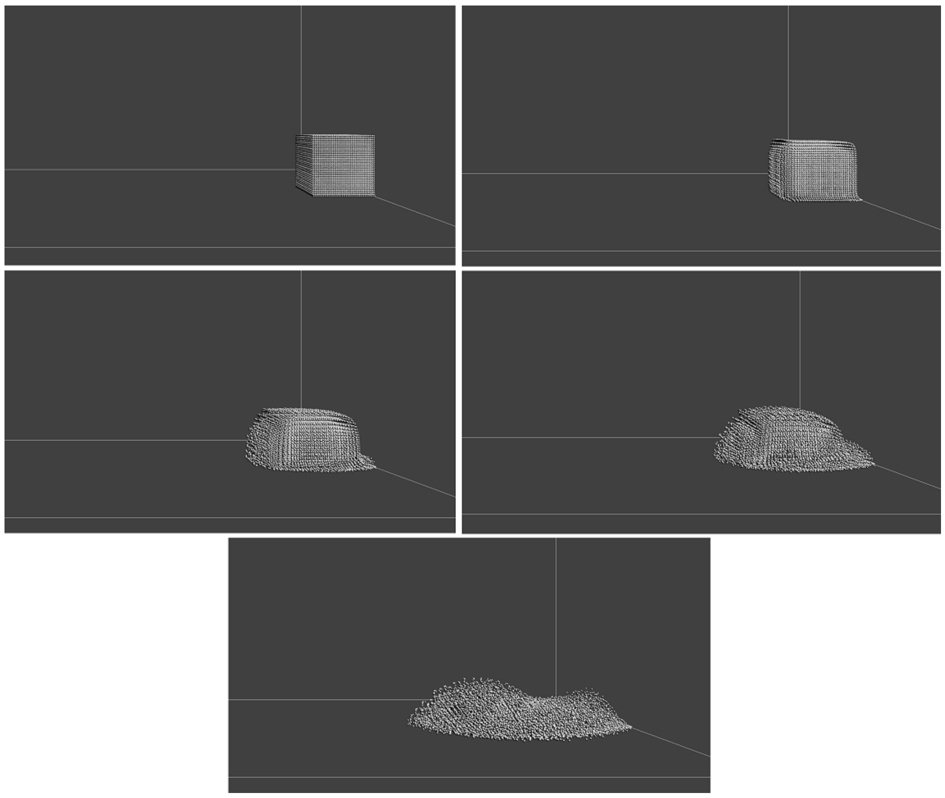


Figure 5.0.3 – Simulation evolution for $\phi = 0.3$

Figure (5.0.3) shows the simulation using a volume fraction of 0.4, while Figure (5.0.4) displays the snow avalanche development using a volume fraction of 1.0. By varying the volume fraction parameter, the flow of fluid is changed, due to the degree of influence the snow density has on the mixture density.

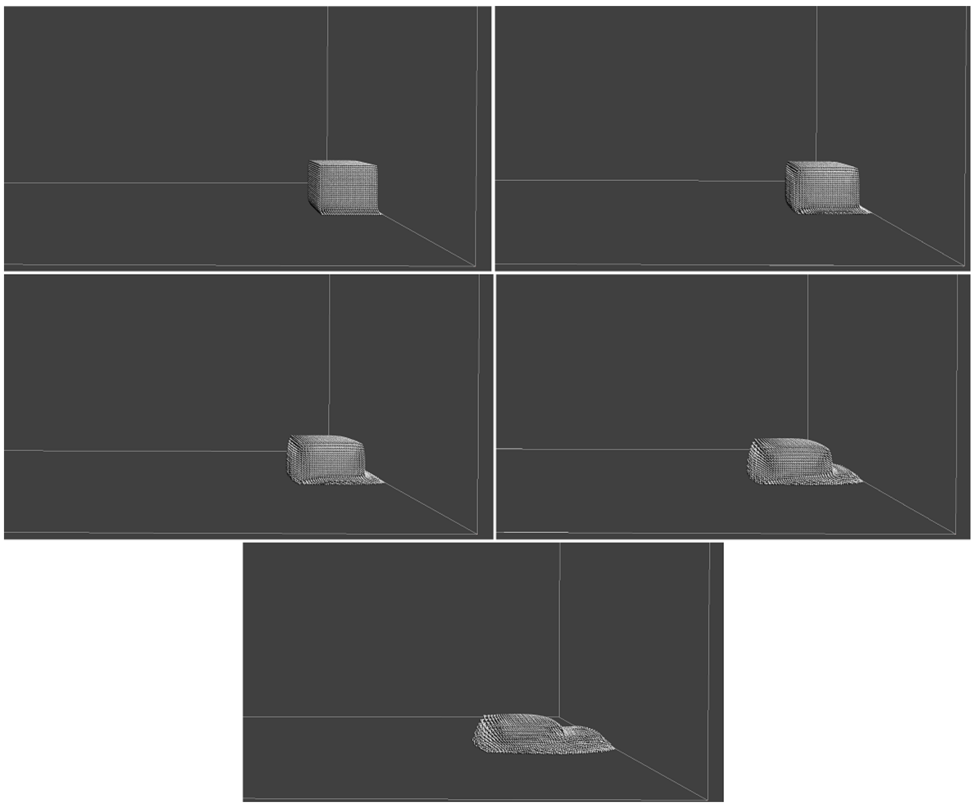


Figure 5.0.4 – Simulation evolution for $\phi = 1.0$

CHAPTER 6

Conclusion and Future Work

The main focus of this thesis was the simulation of a powder-snow avalanche flow. The simulation were implemented using the particle-based simulation solution SPH, from a mathematical model describing powder-snow flow dynamics by Dutykh et.al [23]. The simulation was accelerated by applying the computational power of the GPU, in order to provide a faster simulation time than would have been achieved on the CPU.

The particle-based approach for flow simulation has some limitations and potential issues when simulating the complex dynamics of a powder-snow avalanche. It suffers from an approximation issue when particles are separated by a distance larger than that of the smoothing length defined in the kernel approximation function. The consequence of this is that the behavior of the particles at the outer ridges of the flow, when the flow evolves into a powder cloud state, may not be calculated correctly, due to the lack of surrounding particle quantities to interpolate. In order to simulate in more detail the dynamics of these areas, a larger amount of particles are needed, which will slow down the simulation substantially. The current implementation is based on the SPH rules and formulation techniques presented by

Monaghan [67]. Several other techniques and modifications has later been developed that increase both the stability and the speed of SPH simulations [69, 17, 1]. There is therefore great potential to further improve on the simulation performance, and hence be able to provide good results of the dynamics of the flow at reasonable speeds.

Another issue regarding the simulation system is the handling of stratification that will occur as the flow is evolving. Stratification refers to the state of the flow being consisted of two or more layers of fluid with different properties, as is the case in powder-snow avalanches. The dense core at the bottom of the flow has different physical properties than the cloud of snow that will develop at the top. As the simulation progresses, the fluid mixing resulting from Fick's law of diffusion will modify the density of the flow at certain areas, resulting in this separation of fluid properties. The calculation of the pressure needs to take this density separation into account, in order for the pressure force to be calculated correspondingly [83, 7]. Currently, the pressure force too big at the outer regions of the flow, and at the same time too small near the bottom. This will result in compressible behavior in the dense flow, and expanding behavior in the powder flow. Possible solutions to this problem would be to abandon the weakly incompressible SPH approach and possible employ different incompressibility options.

There is no turbulence modeling in the simulation other than those resolved by the simulation model. By introducing an explicit turbulence modeling procedure, a more detailed and complex dynamic of the flow could be presented. Further improvements could be made by modifying the boundary conditions of the simulation environment, so as to relax the no-slip condition currently implemented. By improving the performance and robustness of the implemented simulation, in addition to extending the system with the aforementioned techniques, the product of this thesis may provide a good framework for general avalanche modeling.

Efforts should also be made regarding the problem of rendering the particle system. A good rendering procedure would be able to develop more visual pleasing results than those provided by OpenGL alone. Further improvements can be made by attempting a more effective implementation of the simulation on the GPU, by applying recent developments regarding the speedup possible by exploiting the computation potential of the GPU [37].

Bibliography

[1] *Smoothed Particle Hydrodynamics: A Meshfree Particle Method.* World Scientific Publishing Company, Incorporated, 2003.

[2] *Avalanche Dynamics.* Berlin: Springer, 2006.

[3] C. Ancey. Powder snow avalanches: Approximation as non-boussinesq clouds with a richardson number–dependent entrainment function. *J. Geophys. Res.*, 2004.

[4] Christophe Ancey. Snow avalanches. In *Geomorphological Fluid Mechanics: Selected Topics in Geological and Geomorphological Fluid Mechanic.*

[5] Christophe Ancey. Plasticity and geophysical flows: A review. *Journal of Non-Newtonian Fluid Mechanics*, 142(1-3):4 – 35, 2007. Viscoplastic fluids: From theory to application.

[6] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[7] Solenthaler Barbara and Markus Gross. Two-scale particle simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, pages 72:1–72:8.

[8] P. Beghin and G. Brugnot. Contribution of theoretical and experimental results to powder-snow avalanche dynamics. *Cold Regions Science and Technology*, 8(1):67–73, 1983. cited By (since 1996) 9.

[9] P. Beghin, E. J. Hopfinger, and R. E. Britter. Gravitational convection from instantaneous sources on inclined boundaries. *Journal of Fluid Mechanics*, 107:407–422, June 1981.

[10] Pierre Beghin and Xavier Olagne. Experimental and theoretical study of the dynamics of powder snow avalanches. *Cold Regions Science and Technology*, 19(3):317 – 326, 1991.

[11] Pierre Beghin and Xavier Olagne. Experimental and theoretical study of the dynamics of powder snow avalanches. *Cold Regions Science and Technology*, 19(3):317 – 326, 1991.

[12] Eloïse Bovet, Bernardino Chiaia, and Luigi Preziosi. A new model for snow avalanche dynamics based on non-newtonian fluids. *Meccanica*, 45:753–765, 2010. 10.1007/s11012-009-9278-z.

[13] A.N. Bozhinskiy and L.A. Sukhanov. Physical modelling of avalanches using an aerosol cloud of powder materials. *Annals of Glaciology*, 26:242–246, 1998. cited By (since 1996) 5.

[14] Henrik Bredmose, D.H. Peregrine, and G.N Bullock. Violent breaking wave impacts : Part 2: modelling the effect of air. *Journal of Fluid Mechanics*, 2009.

[15] H. Brenner. Kinematics of volume transport.

[16] Shuai Che, Jie Li, J.W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101 –107, june 2008.

[17] Andrea Colagrossi and Maurizio Landrini. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *J. Comput. Phys.*, 191:448–475, November 2003.

[18] X. Cui, J.M.N.T Gray, and T. Jóhannesson. Deflecting dams and the formation of oblique shocks in snow avalanches at flateyri, iceland. *J. Geophys. Res.*, 2007.

[19] J. D. Dent, K. J. Burrell, D. S. Schmidt, M. Y. Louge, E. E. Adams, and T. G. Jazbutis. Density, velocity and friction measurements in a dry-snow avalanche. *Annals of Glaciology*, 26:247–252, 1998.

[20] J.D. Dent and T.E. Lang. Experiments on mechanics of flowing snow. *Cold Regions Science and Technology*, 5(3):253 – 258, 1982.

[21] Mathieu Desbrun and Marie paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. pages 61–76, 1996.

[22] Denys Dutykh. *Mathematical modelling of tsunami waves.* PhD thesis, CMLA, ENS de Cachan, 2007.

[23] Denys Dutykh, Céline Acary-Robert, and Didier Bresch. Mathematical modeling of powder-snow avalanche flows. *Studies in Applied Mathematics*, pages no–no, 2011.

[24] M.E. Eglit and K.S. Demidov. Mathematical modeling of snow entrainment in avalanche motion. *Cold Regions Science and Technology*, 43(1-2):10 – 23, 2005. Snow and Avalanches - Papers presented at the European Geosciences Union General Assembly, Nice, France, April 2004.

[25] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21:736–744, July 2002.

[26] J. Étienne, P. Saramito, and E. J. Hopfinger. Numerical simulations of dense clouds on steep slopes: application to powder-snow avalanches. *Annals of Glaciology*, 38:379–383, 2004.

[27] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[28] Adolf Fick. Ueber diffusion. *Annalen der Physik*, 170(1):59–86, 1855.

[29] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 23–30, New York, NY, USA, 2001. ACM.

[30] Yusuke Fukushima, Tatsuji Hagihara, and Mitsuo Sakamoto. Dynamics of inclined suspension thermals. *Fluid Dynamics Research*, 26(5):337 – 354, 2000.

[31] Yusuke Fukushima and Gary Parker. Numerical simulation of powder-snow avalanches. *J. of Glaciology*, 1990.

[32] David A. Fulk and Dennis W. Quinn. An analysis of 1-d smoothed particle hydrodynamics kernels. *J. Comput. Phys.*, 126:165–180, June 1996.

[33] P. Gauer and D. Issler. Possible erosion mechanisms in snow avalanches. *Annals of Glaciology*, 38:384–392, 2004.

[34] Dominik Göddeke. Gpu computing with nvidia cuda. Online, 2009.

[35] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Mon. Not. Roy. Astron. Soc.*, 181:375–389, nov 1977.

[36] David W. Gohar. Opencl tutorial - opencl fundamentals. Online.

[37] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on*

Computer Animation, SCA '10, pages 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[38] Michael J. Gourlay. Fluid simulation for video games. Online, 2011.

[39] J. M. N. T. Gray, M. Wieland, and K. Hutter. Gravity-driven free surface flow of granular avalanches over complex basal topography. *Proceedings: Mathematical, Physical and Engineering Sciences*, 455(1985):1841–1874, 1999.

[40] Simon Green. Particle simulation using cuda, May 2010.

[41] Khronos OpenCL Working Group. *The OpenCL Specification.*

[42] H. Gubler. Measurements and modelling of snow avalanche speeds. In *Avalanche Formation, Movement and Effects.*

[43] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed Particle Hydrodynamics on GPUs. pages 63–70, 2007.

[44] F. Hermann and K. Hutter. Laboratory experiments on the dynamics of powder-snow avalanches in the run-out zone. *Journal of Glaciology*, 37(126):281–295, 1991. cited By (since 1996) 7.

[45] E.J. Hopfinger and J.C. Tochon-Danguy. A model study of powder-snow avalanches. In *J.Gaciol.*

[46] X. Y. Hu and N. A. Adams. An incompressible multi-phase sph method. *J. Comput. Phys.*, 227:264–278, November 2007.

[47] Bruce Hunt. Newtonian fluid mechanics treatment of debris flows and avalanches. *Journal of Hydraulic Engineering*, 1996.

[48] K. Hutter, M. Siegel, S. Savage, and Y. Nohguchi. Two-dimensional spreading of a granular avalanche down an inclined plane part i. theory. *Acta Mechanica*, 100:37–68, 1993. 10.1007/BF01176861.

[49] Kolumban Hutter, Yongqi Wang, and Shiva P. Pudasaini. The savage: Hutter avalanche model: How far can it be pushed? *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 363(1832):pp. 1507–1528, 2005.

[50] Gordon R. Johnson, Robert A. Stryk, and Stephen R. Beissel. Sph for high velocity impact computations. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):347 – 373, 1996.

[51] Micky Kelager. Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics. January 2006.

[52] M. A. Kern, F. Tiefenbacher, and J. N. McElwaine. The rheology of snow in large chute flows. *Cold Regions Science and Technology*, 39(2-3):181 – 192, 2004. Snow And Avalanches: Papers Presented At The European Geophysical Union Conference, Nice, April 2003. Dedicated To The Avalanche Dynamics Pioneer Dr. B. Salm.

[53] Jens Krüger and Rüdiger Westermann. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. Graph.*, 22:908–916, July 2003.

[54] A. Kulikovskiy and E. Sveshnikova. Model dlja rascheta dvizhija pilevoi snezhnoi lavini (a model for computing powdered snow avalanche motion). *Mat. Glatsiologicheskih Isseledovanii*, 1977.

[55] Jie Liu, Seiichi Koshizuka, and Yoshiaki Oka. A hybrid particle-mesh method for viscous, incompressible, multiphase flows. *J. Comput. Phys.*, 202:65–93, January 2005.

[56] M. B. Liu, G. R. Liu, and K. Y. Lam. Constructing smoothing functions in smoothed particle hydrodynamics with applications. *Journal of Computational and Applied Mathematics*, 155(2):263 – 284, 2003.

[57] Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. Two-way coupled sph and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14:797–804, July 2008.

[58] Leon B. Lucy. A numerical approach to testing the fission hypothesis. *The Astronomical Journal*, 82(12):1013–1924, 1977.

[59] J. McElwaine and K. Nishimura. Ping-pong ball avalanche experiments. *Annals of Glaciology*, 32:241–250(10), January 2001.

[60] Y. Meyapin, D. Dutykh, and M. Gisclon. Velocity and energy relaxation in two-phase flows. *ArXiv e-prints*, December 2009.

[61] Yannick Meyapin, Denys Dutykh, and Marguerite Gisclon. Two-fluid barotropic models for powder-snow avalanche flows. Technical Report arXiv:0906.2118, Jun 2009.

[62] J. J. Monaghan. Why particle methods work. *SIAM Journal on Scientific and Statistical Computing*, 3(4):422–433, 1982.

[63] J. J. Monaghan. Simulating Free Surface Flows with SPH. *Journal of Computational Physics*, 110:399–406, February 1994.

[64] J J Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, 2005.

[65] J. J. Monaghan, R. A. F. Cas, A. M. Kos, and M. Hallworth. Gravity currents descending a ramp in a stratified tank. *Journal of Fluid Mechanics*, 379:39–69, January 1999.

[66] J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. 149:135–143, August 1985.

[67] J.J. Monaghan. Smoothed particle hydrodynamics.

[68] J. P. Morris. A study of the stability properties of smooth particle hydrodynamics. 13:97–102, January 1996.

[69] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[70] Angelo Murrone and Hervé Guillard. A five equation reduced model for compressible two phase flow problems. *J. Comput. Phys.*, 202:664–698, January 2005.

[71] Mohamed Naaim and Ibrahim Gurer. Two-phase numerical model of powder avalanche theory and application. *Natural Hazards*, 17:129–145, 1998. 10.1023/A:1008002203275.

[72] Duc Quang Nguyen, Ronald Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph.*, 21:721–728, July 2002.

[73] K. Nishimura and N. Maeno. Experiments on snow-avalanche dynamics. In *Avalanche Formation, Movement and Effects.*

[74] Harad Norem, Fridtjov Irgens, and Bonsak Schieldrop. A continuum model for calculating snow avalanche velocities. *Avalanche Formation, Movement and Effects*, 1986.

[75] NVIDIA. Nvidia geforce 8800 gpu architecture overview. Technical report, NVIDIA, 2006.

[76] NVIDIA. Opencl sdk code samples. 2011.

[77] Lars Nyland, Mark Harris, and Jan Prins. Fast N-Body Simulation with CUDA. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 31. Addison Wesley Professional, August 2007.

[78] G. Parker, Y. Fukushima, and H. M. Pantin. Self-accelerating turbidity currents. *Journal of Fluid Mechanics*, 171:145–181, 1986.

[79] R. Perla, T.T Cheng, and D.M. McClung. A two-parameter model of snow-avalanche motion. *Journal of Glaciology*, 26(94):197–207, 1980. cited By (since 1996) 74.

[80] Christopher I. Rodrigues, David J. Hardy, John E. Stone, Klaus Schulten, and Wen-Mei W. Hwu. Gpu acceleration of cutoff pair potentials

for molecular modeling applications. In *Proceedings of the 5th conference on Computing frontiers*, CF '08, pages 273–282, New York, NY, USA, 2008. ACM.

[81] B. Salm, A. Burkard, and H. Gubler. Berechnung von fliesslawinen: eine anleitung für praktiker mit beispielen. *Eidg. Inst. Schnee- und Lawinenforsch*, 1990.

[82] S. B. Savage and K. Hutter. The motion of a finite mass of granular material down a rough incline. *Journal of Fluid Mechanics*, 199:177–215, 1989.

[83] B. Solenthaler and R. Pajarola. Density contrast sph interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 211–218, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[84] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. *ACM Trans. Graph.*, 28:40:1–40:6, July 2009.

[85] Barbara Solenthaler. *Incompressible fluid simulation and advanced surface handling with SPH.* PhD thesis, University of Zurich, 2010.

[86] B. Sovilla and P. Bartelt. Observations and modelling of snow avalanche entrainment. *Natural Hazards and Earth System Science*, 2(3/4):169–179, 2002.

[87] Betty Sovilla. Field experiments and numerical modeling of mass entrainment in snow avalanches. *J. Geophys. Res.*, 2006.

[88] Y.-C. Tai, K. Hutter, and J. M. N. T. Gray. Dense Granular Avalanches: Mathematical Description and Experimental Validation. In N. J. Balmforth & A. Provenzale, editor, *Geomorphological Fluid Mechanics*, volume 582 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 339–+, 2001.

[89] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vision Modeling and Visualization*, pages 47–54, 2003.

[90] Jocelyn Étienne, Emil J. Hopfinger, and Pierre Saramito. Numerical simulations of high density ratio lock-exchange flows. *Phys. Fluids*, 2005.

[91] B. Turnbull, J. N. McElwaine, and C. Ancey. Kulikovskiy–sveshnikova–beghin model of powder snow avalanches: Development and application. *J. Geophys. Res.*, 2007.

[92] A. Voellmy. Über die zerstörungskraft von lawinen. In *Schweizerische Bauzeitung*.

[93] Y. Zhang, B. Solenthaler, and R. Pajarola. Adaptive sampling and rendering of fluids on the gpu. In *Proceedings Symposium on Point-Based Graphics*, pages 137–146, August 2008.