Solidification using Smoothed Particle Hydrodynamics

Master Thesis
CA-3817512

Game and Media Technology
Utrecht University, The Netherlands

Supervisors:
dr. ir. J.Egges
dr. N.G. Pronost

July, 2014

Abstract

An increasingly popular way to display fluids in multimedia is by generating physically-based fluid simulations. In the last couple of years hardware has finally been powerful enough to simulate particle-based fluid simulation techniques in real-time. Existing works provide basic fluid simulators but lack more advanced fluid aspects. Advanced aspects include temperature, phase transitions such as the freezing or melting of water. More advanced aspects are the interactions, such as heat diffusion and collisions, between fluids and complex objects (i.e. terrain).
We propose extensions to the Smoothed Particle Hydrodynamics fluid simulation in order to achieve this functionality for the common fluid water. We attempt to retain the real-time nature of GPU-based SPH by extending an existing framework and by monitoring the performance impact for each extension.
Our contributions include the addition of temperature to the simulated fluids and heat conduction to manipulate this temperature. We have added phase transitions which can adjust the physical state of the fluid as a result of the change of properties of the fluid. Finally we have added interaction between the fluid and an environment in order to more realistically simulate heat transfer. Our extensions make use of existing functionality in the Smoothed Particle Hydrodynamics fluid simulator, this way we can provide computationally heavy extensions with a minimal degradation of the performance.

“The concept of free will and moral responsibility for our actions are really an effective theory in the sense of fluid mechanics.”

Stephen William Hawking

Contents

Chapter 1
Introduction

An increasingly popular way to display fluids in multimedia is by generating physically-based fluid simulations. Fluid simulation is an ongoing study in the field of fluid mechanics. The topic has been studied for centuries and theory dates back to ancient Greece [1]. It has only been recently, however, that we are able to use this theory to create our own fluid simulators.

Original solutions originate from the field of astrophysics where fluid simulations were used to model the formation of stars. The first adaptation of the current most widely used method for fluid simulation, Smoothed Particle Hydrodynamics, was introduced in 1977 [2]. Since it is a relatively young field of research, existing work focuses on realistic behavior of a fluid simulation in the sense of the flow of the fluid and hasn't yet deviated much to extensions of existing fluid simulators. More advanced aspects of fluids such as the phase transitions of freezing and melting have not yet been properly explored, while these are required to simulate scenarios such as the one in Figure 1.1.

A downside of Smoothed Particle Hydrodynamics is that it is computationally expensive in comparison to traditional Eulerian fluid simulations for real-time applications, but yet it offers a much more realistic result. Recent hardware makes it possible to use a real-time implementation of the fluid simulation method and offers the possibility of an interactive environment. This offers the opportunity to create real-time realistic fluids for applications such as simulations of geographical events (i.e. what happens when a volcano erupts), medical simulations related to for example blood research, or games. The possibility to interact with a fluid simulator in an interactive manner also triggers the desire for more advanced fluid properties.

The key difference between an interactive real-time simulation of fluids and the physically correct computations for fluid simulation is that the result only has to be visually plausible. In other words, the realism of the simulation is a trade-off between computation time and physically correctness. Therefore it is common that simplifying assumptions are made in order to make it possible to model complex fluids.

Figure 1.1: Frozen Lake Baikal, Russia. From [52].

1.1 Motivation

There are currently no methods which provide the possibility to freeze and melt a fluid in a physically correct manner. In order for physically-based fluid simulations to really succeed as the way to simulate fluids in applications such as games, more advanced aspects of fluids will have to be added to traditional simulators. Existing work exists on the field of heat diffusion but these are slow and not integrated with Smoothed Particle Hydrodynamics. Also, the interactivity of simulations presented by existing research is limited to simple geometry but also requires an extension for heat diffusion.

1.2 Goal

The goal of this thesis is to introduce an advanced Smoothed Particle Hydrodynamics model which can simulate phase transitions of fluids in a physically correct way. We will also introduce more advanced interaction models for the interaction between particles and an environment and extend this with heat diffusion. Furthermore, we attempt to preserve the real-time computational possibility of the simulation with our additions so that the more mature Smoothed Particle Hydrodynamics method can be used in games in the near future.

1.3 Thesis Overview

This thesis will provide the reader with the essential understanding of fluid mechanics in order to work with fluid simulation methods. In the next chapter, we will describe this essential theory and different simulation methods which will eventually lead to the Smoothed Particle Hydrodynamics method. Since this method forms the basis of our research we will also describe this method in depth. Furthermore, phase transition phenomena and interaction techniques are explained and later used for our solidification solution. In Chapter 3 we will describe the fundamentals of freezing a fluid. In the second section of this chapter heat diffusion is explained and we describe in detail how this is implemented in our Smoothed Particle Hydrodynamics extension. And finally the solidification extension is expanded with the phase transition of going from a solid to a liquid state: melting. Existing interaction models are expanded in Chapter 4 by providing a solution for interacting with other solid objects such as a terrain. Our results can be found in Chapter 5 and the conclusions in Chapter 6 will summarize our research and provide ideas for future work on the subject in order to make fluid simulations even more advanced and realistic.

Chapter 2
Related Work

This chapter will describe the basics of fluid mechanics and fluid simulation. Fluid simulation is generally done in two steps: the physical simulation of the fluid and the visualization of the fluid. We will mainly focus on the physical simulation of the fluid. This chapter will describe the difference between Lagrangian and Eulerian fluid simulators and give a thorough explanation of the Smoothed Particle Hydrodynamics fluid simulation technique. Related work regarding heat transferring and the interaction of fluids with other systems will also be discussed. A goal of this thesis is to present a solid fluid simulator which offers real-time solidification. With this in mind, the focus of the research is on efficient implementations which give a visually convincing approximation instead of a fluid simulation that is physically as accurate as possible.
First of all, section 2.1 will describe fluid dynamics which studies the motion of fluids in the real world. This section will contain the theory and mathematical background for the understanding of this subject. In section 2.2 we will discuss fluid simulation and how methods have been changed over the years to result in the current most widely used Smoothed Particle Hydrodynamics method. This method will be explained thoroughly in section 2.3. Previously existing work and theory supporting the solidification of a fluid will be discussed in section 2.4. Finally, the interaction between fluids with different properties will be discussed in section 2.5.

2.1 Fluid Dynamics

Fluid dynamics is the active field of research of fluid mechanics which studies the motion of fluids. The motion of fluids is described by Newton's laws of motion which we will outline in this section. The main sets of equations which are used to describe the motion of fluids are the Navier-Stokes equations which we will outline afterwards. The theory handled in this section forms the basis for physically correct computational simulation of fluids.

2.1.1 Newton's Second Law of Motion

Newton's first law states that an object remains at rest or moves at a constant velocity, unless an external force acts upon this object [3]. The second law states that the complete overall force of an object is equal to the acceleration of the object multiplied by its mass. This gives the following equation:

$$F = ma$$

where

- F is the net force
- m is the mass of the object
- a is the acceleration vector of the object

Eq. 1

This equation can be interpreted as the conservation of momentum. If there are no forces acting on the object it means there is no change of velocity. This can be deduced as acceleration equals the change in velocity over time as seen in equation (2) [4].

$$a = \frac{dv}{dt}$$

where

- a is the acceleration vector of the object
- dv is the change in velocity
- dt is the change in time

Eq. 2

The difference between a Laplacian and an Eulerian fluid influences the way how Newton's second law is interpreted. More about the differences between a Laplacian and an Eulerian fluid simulation can be found in Section 2.2.1 and Section 2.2.2. The main difference which should be understood is that in a Laplacian fluid simulator, particles describe the fluid and are moved according to their own and neighbouring properties, whereas in an Eulerian

fluid simulation the domain in which the particles reside describe the fluid and governs the state of the particles. Also keep in mind that Eulerian simulations are used in classic fluid solvers whereas Laplacian simulators are used in more modern solvers such as Smoothed Particle Hydrodynamics [5].

Newton's third law is also important for realistic fluid simulations. This law states that when an object applies a force upon another object, this second object applies an equal force in the opposite direction of the first force. Action equals minus reaction as shown in Figure 2.1.

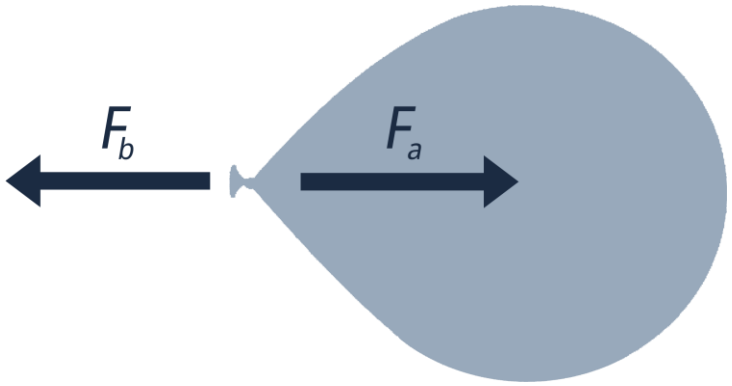


Figure 2.1: Newton's third law balloon demonstration. F_b is the force exerted by the balloon on the air. F_a is force exerted by the air on the balloon.

2.1.2 Navier-Stokes Equations

In 1822 Claude-Louis Navier and George Gabriel Stokes contributed the Navier-Stokes equations to the field of fluid mechanics. These equations describe the motion of fluid by applying Newton's second law to fluids. At the time that these equations were modelled, they could only be analyzed on a mathematical scale. In the second half of the 20th century progress could be made as computers became available for researchers. The Navier-Stokes equations are computationally expensive in their general forms as they are precise models for fluid flows [6]. Luckily, the equations can be simplified for real-time applications while not sacrificing any visual result. In other words, the simulations will be less accurate but the human eye will not notice the difference [7].

In order to understand the Navier-Stokes equations, we will start with the most general form of the equations and reduce this to a form with which we can work. The classical equations to compute the motion for incompressible fluid flow over time are the following two equations [8]:

$$\rho(\frac{\partial u}{\partial t} + u \cdot \nabla)u = -\nabla p + \mu\nabla \cdot (\nabla u) + F$$

where

- ρ is the density of the fluid
- p is the scalar pressure field of the fluid
- u is the vector velocity field of the fluid
- μ is the viscosity of the fluid
- F is the sum of the external forces acting on the fluid

Eq. 3

$$\nabla \cdot u = 0$$

where

- u is the vector velocity field of the fluid

Eq. 4

Equation (4) describes the conservation of mass for an incompressible fluid and is called the continuity equation [9]. This means that the velocity field has zero divergence. This enforces the incompressibility by stating that the volume of the fluid is constant over time.

2.1.3 Mathematical Background

In the equations used in fluid dynamics we encounter the nabla (∇) operator. This operator is used in the applications of the gradient, divergence and Laplacian operators [10]. This section will cover these applications of the nabla operator because they are used in normal fluid simulations, but even more so as they are also used in simulations which use heat diffusion.

Gradient

The gradient of a scalar field is a vector of the partial derivatives of this scalar field. The definition of the gradient operator can be found in Equation 5. The result of the gradient of a scalar field is a vector field.

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \ldots . \frac{\partial f}{\partial n}\right)$$

where

- $\partial x, \partial y, \partial n$ are the spacings of the scalar field in the x, y and n dimensions.

Eq. 5

Divergence

The divergence of a vector field is the measure of the rate of which the parameter of the field exits a given region in the field. In the Navier-Stokes equations, the divergence operator is applied to the velocity of the flow of the fluid and thus describes the change in velocity for a region in the velocity field. The result of the divergence of a vector field is a scalar field.

$$\nabla \cdot u = \left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} + \ldots \frac{\partial f}{\partial n}\right)$$

where

- $\partial x, \partial y, \partial n$ are the spacings of the scalar field in the x, y and n dimensions.

Eq. 6

Laplacian

The Laplacian operator is the result of when the divergence operator is applied to the result of the gradient. The Laplacian of a function f at a point in Euclidean space is the rate at which the average result value of the function over a sphere centered at the point deviates from the initial value when the radius of this sphere is changed [11].

$$\nabla^2 f = \left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \ldots \frac{\partial^2 f}{\partial n^2}\right)$$

where

- $\partial x, \partial y, \partial n$ are the spacings of the scalar field in the x, y and n dimensions.

Eq. 7

2.1.4 Vector Field

A vector field is a field in Euclidean space in which a vector is assigned to each point in this space. Generally the space is a grid with uniform sizes [12]. The results of the Navier-Stokes equations are vector fields. The vector quantity of those fields is the velocity of the fluid at a given point in the described space. Figure 2.2 displays such a velocity field. These velocity fields can be used to obtain the velocity of a fluid at a given point in the space. This velocity value can then be used to reposition the fluid accordingly.

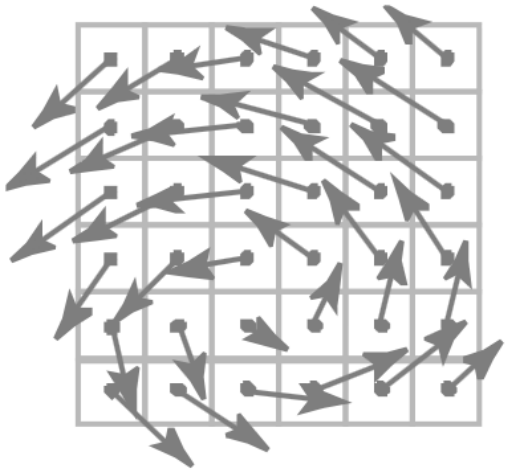


Figure 2.2: Velocity field from [8].

2.2 Fluid Simulation

Fluid simulation is the science of generating realistic animations of fluids. As described in the previous section, fluid simulation is based on the Navier-Stokes equations. In this section we will describe the most frequently used methods for fluid simulation: the Eulerian and Lagrangian methods. Other fluid simulation methods which are less widely used and which we will not describe are vorticity-based methods and Lattice-Boltzmann methods. Finally we will end this section by describing Smoothed Particle Hydrodynamics in detail.

2.2.1 Eulerian Methods

Leonhard Euler designed equations for fluid dynamics which manage inviscid flow. Inviscid flows govern the flow of ideal fluids which do not have viscosity, but also fluids which have very low values of viscosity are considered to be in the inviscid category. Ideal, or perfect, fluids are fluids which do not have any viscosity, heat conductivity or shear stresses [13]. Practical examples of inviscid fluids are the matter of which stars consist. The advantage of using the assumption that a fluid is inviscid is that the fluid mechanic problems are much easier to solve as the equations are simpler [14].
The Navier-Stokes equation can be reduced to the following Euler's equation in the case of a fluid with inviscid flow:

$$\rho g - \nabla p = \rho \frac{dv}{dt}$$

where

- ρ is the volumetric mass density of the fluid
- p is the pressure of the fluid
- g is the constant gravity force
- v is the velocity of the fluid
- t is the simulation time

Eq. 8

This equation can then be used to create the following equations to compute coordinates in the Euclidean space.

In an Eulerian fluid simulation the particles in the fluid are moved through specific locations in the space of the simulation. These locations can be defined by a grid [15]. The flow can be described by the following function:

$$v\,(x,t)$$

where

- v is the velocity
- x is the location in the space of the simulation
- t is the simulation time

Eq. 9

Other parameter quantities can also easily be obtained by using equations which are similar to Equation 9. For example the following equation can be used to obtain the density of the fluid:

$$\rho\,(x,t)$$

where

- ρ is the density
- x is the location in the space of the simulation
- t is the simulation time

Eq. 10

Each cell in the grid holds parameter quantities of the fluid such as velocity and density. This way these parameters of a particle can be retrieved from the grid by knowing the current position of the particle and the current time of the simulation. This method offers a

simple and quick implementation of a fluid dynamics solver. However, it also introduces challenges in regards to large fluid simulations. The largest drawback is that the scale of the fluid is limited to the dimensions of the grid. Another drawback is that the accuracy of the result depends on the resolution of the grid. The computational time also scales exponentially with the resolution size.
The velocity field (for example, see Figure 2.2 and Figure 2.3) will be adapted according to the particles which reside in the field. This is a result of mass conservation. An important difference between Eulerian and Lagrangian methods is that in the Eulerian solver particles have no knowledge of each other; all the adaptations of a particle are done through lookups of the velocity field, whereas in a Lagrangian solver particle adaptations are done through lookups of neighbouring particles. In the next section we will discuss Lagrangian particle methods.

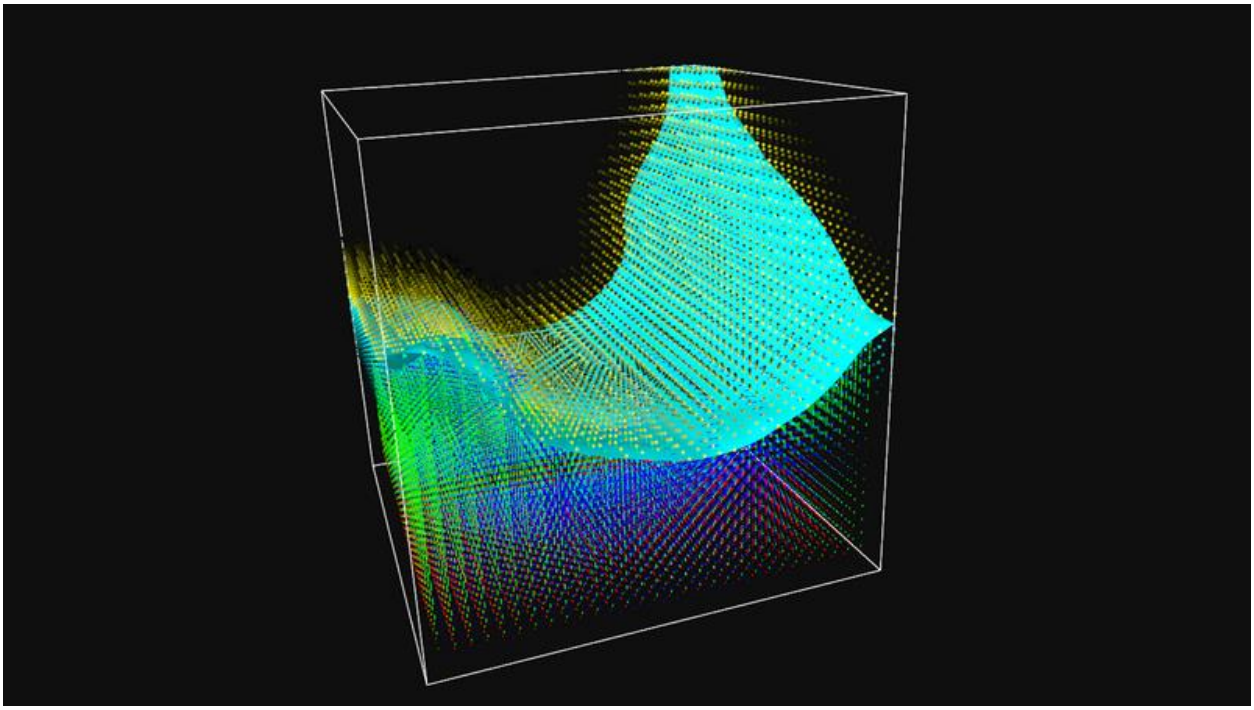


Figure 2.3: A 3D Eulerian simulation result: a velocity field, from [15].

Important to note is that with the grid-based method a lot of information about individual particles is lost as most information is stored in the grid. This offers challenges when additional parameters are required to be added to the simulation. Temperature for example would be a challenging parameter to add as heat diffusion is measured on molecule level.
Optimizations for Eulerian methods exist where the grid in which the fluid resides is adaptive [16]. It is advantageous to use adaptive grids over traditional grids with equal cell

sizes for simulations where the domain is not a square for example. For cases where the domain is a spherical volume different cell sizes might be required based on the location of the fluid in the domain. Grids which handle this problem by dynamically changing the sizes of their cells according to the domain are called adaptive grids. Similarly, a grid doesn't always have to be fixed in an arbitrary position in space, but it could move in certain situations. It is often better to move boundaries of the grid around with the fluid than to dynamically add new grid cells around the original domain and transfer mass and other properties [17].

2.2.2 Lagrangian Methods

In a Lagrangian method the new position and state of a particle are computed for each individual particle. Instead of looking at a grid for the information of the fluid, the particles depend on their own parameters and those of their neighbouring particles. Neighbouring particles are defined by a constant distance for which each particle within that distance is used to compute the new values of the particle [18]. All parameters of a particle in the Lagrangian description of fluid flow are described as a function of time. This gives the following generalized formula:

$$x\ (a, t)$$

where

- x is an arbitrary parameter of the particle
- a is the current state of the particle
- t is the simulation time

Eq. 11

With this information, the arbitrary parameter of a particle can be computed for any given time.

Lagrangian approaches are also called particle-based approaches as they completely depend on information of the particles. Almost all theory in fluid mechanics is originally developed with Eulerian systems in mind as Lagrangian systems were simply too computational expensive to develop on older hardware [19].

2.3 Smoothed Particle Hydrodynamics

One of the most widely accepted methods for real time fluid simulation is the Smoothed Particle Hydrodynamics method (SPH). This method has first been introduced in 1977 [2]. The method was designed for compressible flow problems and was first used in the field of Astrophysics to simulate the flow of interstellar gas. The first adaptation for computer

graphics was made in 1995 by Stam and Fiume [20].
As the name states, SPH is a particle-based fluid simulation method. The main advantage of a particle-based method is that the representation of the fluid consists of multiple objects rather than one. This gives the possibility for parallelization in order to get quick computing speed. Visually speaking, particle-based methods offer more realism in comparison to methods which are not particble based (i.e. vertex based) as it is possible to create for example sprays, splashing fluids and smoke. The combination of the broad range of opportunities and the possibility to parallelize the computations on current hardware make the SPH method an increasingly popular technique for real time fluid simulation.
In this section, the SPH method will be explained in the following order. First, the essentials of the method will be explained. Secondly we describe which kernels are used to smooth the properties of the SPH fluid. And finally, the steps of the method are described.

2.3.1 SPH Essentials

SPH is a Lagrangian simulation method for free surface flow. The method simulates fluid based on particles. These particles represent properties of the fluid on given positions in the fluid's domain. These properties are calculated with the kernels given later in this section and are smoothed over a given spatial distance for each particle.
The governing equation used as the basic interpolation formula is the following:

$$A(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h)$$

where

- A is a quantity which is to be computed
- r is the given 3D coordinate
- m_j is the mass of particle j
- ρ_j is the density of particle j
- W is the kernel function used to smooth the values
- h is the cut off radius for the interpolation

Eq. 12

The radius defined by h means that if $|\, r - r_j \,| > h$, the value for the sampled particle will be zero, thus it will not be included in the summation. The result of this equation is the value of a quantity at a given location in the fluid. Quantities which are required to be calculated in the default SPH implementation are density, pressure and viscosity. These values can then be used to compute the new state of the fluid.

If we would apply Equation 12 to the case of density, we would get the following equation:

$$\rho(r) = \sum_j m_j W(r - r_j, h)$$ Eq. 13

Note that this shows that the density of a particle in a SPH simulation is calculated by smoothing the mass of all particles in the simulation.

2.3.2 Kernel Functions

The value of a property of a particle in the SPH simulation has to be interpolated from a discrete set of sample points, as is the case in other numerical solutions in fluid dynamics. The interpolation is done through a kernel function which is a radial symmetric smoothing function. These smoothing functions result weights by computing a weighted average of the values of the nearest neighbours of the sample point. This means that the closer a neighbour is to the sample point, the more weight its value will have. A suitable smoothing kernel has to meet the following two requirements in order to result valid weights:

$$\int W(r,h)\, dr = 1$$ Eq. 14

$$W(r,h) = W(-r,h)$$ Eq. 15

The given requirements can be summarized into the following rules: the kernel has to be normalized (Equation 14) and symmetrical (Equation 15). As long as a smoothing kernel

meets the requirements, the chosen kernel is suitable. Important is however that the complexity of the chosen kernel strongly relates to the speed and of the simulation, whereas the quality of the kernel is related to the stability of the simulation. Also, the kernels should be able to support large time steps when integrating and they should have vanishing derivatives at their boundaries. Therefore we have chosen to use kernels defined in previous related work, rather than coming up with our own kernel functions [21]. The following three kernels are used to compute density, forces due to pressure and forces due to viscosity. They are called the poly6, spiky and viscosity kernel in respective order. In more advanced SPH implementations, more kernels might be required. Figure 2.4 shows the graphs of the used kernels.

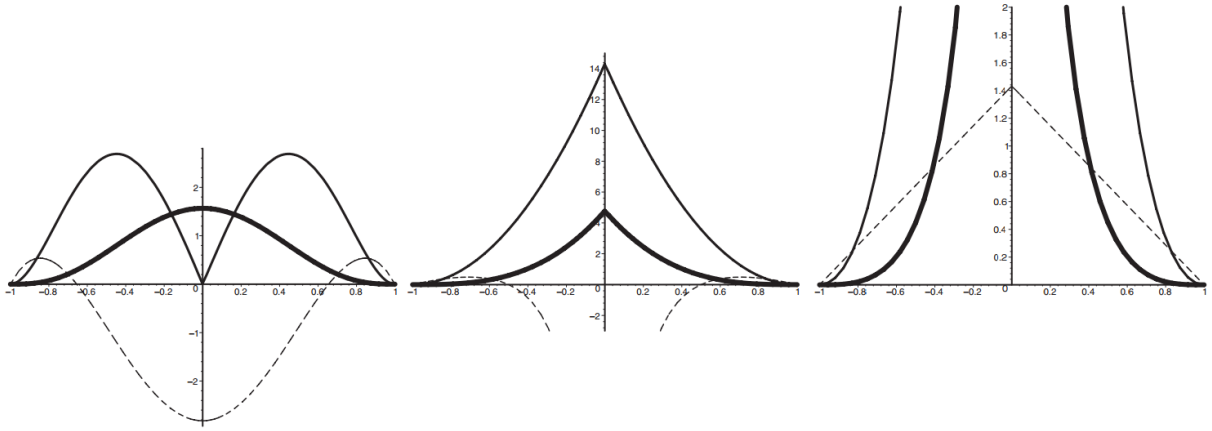


Figure 2.4: Plots of the three used kernel functions from [21]. From left to right: poly6, spiky, viscosity. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines show the Laplacians. The x-axis shows the values for r between –h and h for h=1, the y-axis shows the results of the kernel functions.

Density Kernel: poly6

The poly6 kernel (Eq. 16) is used in the computations for the density on a given location. Equation 13 tells us that we require the normal kernel for the calculation of the density and not the gradient or the laplacian. A drawback of the Poly6 kernel is that it has vanishing gradients closer to the center as can be seen on figure 2.4. If this kernel would be used for the computation of the pressure forces, it results in an error in the SPH simulation where when two particles get too close to each other, they don't generate enough force anymore to repel the other. Therefore the spiky kernel is chosen for such situations.

for $-h \leq r \leq h$

$$W_{density}(r,h) = \frac{315}{64\pi h^9}(h^2 - r^2)^3$$

otherwise

$$W_{density}(r,h) = 0$$

Eq. 16

Pressure Kernel: spiky

The Navier-Stokes equations (Eq.13) tell us that we need to use the gradient of a kernel to calculate the pressure. The spiky kernel (Eq.17) has an increasing gradient length closer to zero, which makes sure that particles which are close together will get pushed away from each other to avoid clustering. This kernel preserves the impact of the repulsive forces between two particles.

for $-h \leq r \leq h$

$$W_{pressure}(r,h) = \frac{15}{\pi h^6}(h - r)^3$$

otherwise

$$W_{pressure}(r,h) = 0$$

Eq. 17

Viscosity Kernel

The Navier-Stokes equations (Eq.13) tell us that we need to use the Laplacian of a kernel to calculate the viscosity. A requirement for the velocity kernel is that a particle should get slowed down when it is near other objects. A negative Laplacian increases the acceleration of a particle [21] and therefore we need a kernel for which the Laplacian stays positive. The following positive Laplacian kernel is chosen and will make sure velocities will be decreased between particles that are close to each other and thus result in clustering particles.

for $-h \leq r \leq h$

$$W_{viscosity}(r,h) = \frac{15}{2\pi h^3}\left(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1\right)$$

Eq. 18

otherwise

$$W_{viscosity}(r,h) = 0$$

2.3.2 Method Overview

One complete simulation step of the Smoothed Particle Hydrodynamics method applies the steps described in this section. Figure 2.5 displays the flow of the SPH method described in the following steps.

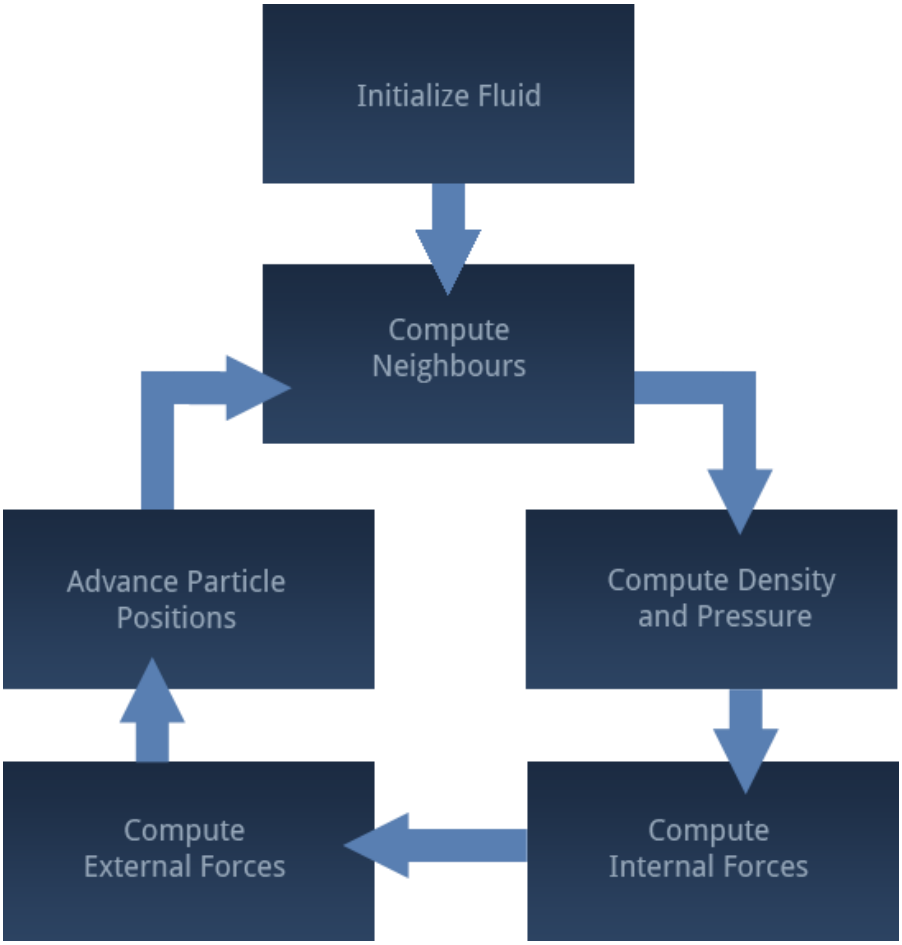


Figure 2.5. Flow of the SPH method.

Step1: Calculate the density of each particle
The density calculations can be done with the use of Eq. 13. The smoothing kernel used which in this case is the poly6 kernel (Eq. 16).

Step2: Calculate the pressure of each particle
The pressure of a particle can be calculated with the following equation [22]:

$$P_i = k(\rho_i - \rho_0) \qquad \text{Eq. 19}$$

Where P_i is the pressure of particle i and k is the gas stiffness constant. The standard gas law has been extended in this formula for it to work correctly with liquid fluids rather than gaseous fluids by the parameter ρ_0 which represents the constant rest density of the fluid. The correct values of those constants are dependent on the type of fluid.

Step 3: Calculate the internal pressure force
The following equation is used to calculate the internal pressure force for a particle:

$$f_i^{pressure} = -\sum_{j \neq i} \frac{\rho_i + \rho_j}{\rho_i^2 + \rho_j^2} \nabla W(r_i - r_j, h) \qquad \text{Eq. 20}$$

The kernel used in this calculation is the spiky kernel (Equation 17). The linear and angular momentums of this particle are conserved as the pressure force is symmetrical. Therefore Newton's 3rd law is held.

Step 4: Calculate the internal viscosity force
The viscosity of a fluid can also be read as the internal resistance to flow. A higher viscosity coefficient for a fluid, results in more resistance to flow and thus its kinetic energy.
The equation used for the viscosity forces calculations is the following:

$$f_i^{viscosity} = \mu \sum_{j \neq i} m_j \frac{v_j + v_i}{\rho_j} \nabla^2 W(r_i - r_j, h)$$

where

- μ is the viscosity coefficient of the fluid
- v is the velocity

Eq. 21

Step 5: Calculate the force applied by gravity (external force)

The gravitational force is included in order to make the simulation look realistic. It is not necessarily required for SPH and unrealistic gravitational forces can also be used to simulate fluid behaviour on different planets or simulations with local gravities. The used equation is the following:

$$f_i^{gravity} = \rho_i g$$

where

- g is the gravitational acceleration

Eq. 22

Step 6: Calculate the force applied by body collisions (external force)

External forces are forces which don't originate from within the fluid itself. An example which is commonly used in fluid simulations is the collision with boundaries in which the fluid is constraint. It should be impossible for a particle to penetrate the boundary of the domain.

As collision handling is a large field of research and the implementation depends on the required accuracy and realism, we will not further expand on this topic. Commonly, SPH uses the following method to calculate collision response by the particles.

When a particle gets closer to a boundary than an arbitrary value ε, it is considered a collision. The normal direction of the collision is then used to compute the acceleration that has to be added to the currenct acceleration of the particle. This is done by calculating the dot product between the velocity and the normal and multiply this with the damping value ζ of the boundary. This value can then be subtracted from the fluid stiffness k multiplied by the distance between the particle and the boundary. Finally this normal is multiplied by the computed value and added to the current acceleration of the particle.

The combination of the forces computed in step 2,3,4 and 5 gives the total acting force F_i:

$$F_i = f_i^{pressure} + f_i^{viscosity} + f_i^{gravity} + f_i^{boundary}$$

Eq. 23

Step 7: Calculate the new position for each particle

The integration scheme used to update the positions of the particles is the leapfrog integration scheme. The leapfrog integration scheme is a method for numerically integration [31]. The name leapfrog comes from the general idea of the method where different properties such as position and velocity are updated in different time intervals and thus leap over each other. This gives the following two equations:

$$v_i(t + \Delta t) = v_i + \frac{1}{2}(a_i + a_i(t + \Delta t))\Delta t$$

where

- t is the current time of the simulation
- Δt is the timestep

Eq. 24

$$r_i(t + \Delta t) = r_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2$$

Eq. 25

2.4 Solidification

In the previous sections we have described how fluids can be generated in a physically plausible manner. Other aspects of fluids which contribute to the realism of a simulation are freezing and melting. We will refer to the method which will handle these phase changes as solidification.
In this section we will first describe how previous research contributes to freezing simulations. Secondly we will describe fluid systems which consist of more than just one alligation (such as water and salt). Finally we will describe existing work on melting simulations.

2.4.1 Viscosity Solutions

Traditional solidification methods solved the phase transition problem by adjusting the viscosity of the fluid material so that the fluid appears less liquid. D.Stora, P.Agliati and M.Cani animate lava flows by linking viscosity to a grid-based temperature field which uses heat transferring. In terms of physics, lava is a liquid which is extremely viscous for which the viscosity increases exponentially when the lava cools down [23].
M.Carlson and P.Mucha use the Marker-and-Cell grid-based method [3] to simulate fluids in order to achieve a viscosity solution for the solidification problem. One drawback of the viscosity-based solutions is also addressed by them. Numerical instability is a risk because viscosity influences velocity greatly. In order to counter this, the time step of the simulation has to shrink according to the size of the viscosity. Near-solid fluids require around 60.000 more time steps in comparison to completely liquid fluids.
Another drawback of viscosity-based methods is that they can never reach a solid state. The fluids will appear increasingly more solid when the viscosity gets larger, but as there is no upper bound for the viscosity, a complete solid state is never reached.

2.4.2 Alligation Systems

Monaghan et al. [24] propose a method to simulate the freezing of one and two-component alligation systems as an extension of the SPH method. Multiple component solidification solvers are interesting for geology and industry where liquids do not only consist of oxygen and hydrogen. The emphasis of their research is on the realism of the freezing process of the alloy-based systems. The SPH and solidification calculations are simplified and they assume that particles which have been solidified remain in the position where the phase transition occurred. Thus ‘frozen’ particles remain in stasis when their phase changes based on their temperature. It is interesting to note that the accuracy of the temperature diffusion gets improved when the number of particles gets increased. However, this particular method does not actually turn an object into a solid completely, only partially.

2.4.3 Melting

Existing work exists for melting solid. Solenthaler et al. treat solids as mesh-based objects and not as fluids [4]. When they melt or burn based on temperature, the resulting liquid and gas are simulated as fluids. A great advantage of this method is that it can easily be incorporated in existing simulations such as games where objects are generally handled as meshes. Contrary to viscosity-based methods, this method does not treat solids as high-viscous fluids. However, it also does not offer the possibility to change a liquid into a solid. The proposed method is different from other methods in the way that the state change from solid into liquid has to be handled differently than in the other methods. The density of the source solid and a resulting liquid is equal in both states. So the volume of the resulting liquid is approximately the same as the volume in the solid. The resulting liquid is modelled using a grid-based system which forms a grid around the boundaries of the solid object [25]. When the solid is about to melt, particles are formed within the boundaries of the solid-based on the volume of the solid. These particles can then be released from the solid by removing their constraints so that they form a fluid.

2.5 Fluid-Fluid Interaction

An interesting and complex phenomenon in fluid simulation is the interaction between different fluids. In the case of fluid to fluid interaction there is a distinction to be made between fluids which share the same properties and fluids which have different properties. Besides fluid-fluid interaction, fluid-solid interaction is also an interesting behavior in interactive situations. The most common example for fluid-fluid interaction is the interaction between air and water. As air is a gas, it can also be modelled as a fluid. Following are a number of phenomena which can be addressed by using a fluid to fluid interaction scheme.

2.5.1 Multiple Fluids

In order to simulate multiple fluids interacting with each other there is no new theory that has to be introduced. Lagrangian methods are exceptionally suited for multiple fluids. In the case of an SPH-based simulation, multiple fluids will directly work. It does however offer some opportunities for optimizations such as making certain fluid parameters global which are the same for all the fluids in the simulation [26]. Examples of properties which are the same for multiple fluids can be the particle mass, rest density or viscosity coefficients.

2.5.2 Combining fluids

Complex chemical reactions can be handled in the case of certain fluids interacting with each other. Multiple fluids can be used to combine or convert themselves into a third new fluid [27]. Another method addressed more complex phenomena such as fire by using ghost values [28]. In order to achieve the phase change where one fluid material turns into another, surface reactions are used [29]. Surface reactions can be anything defined by the source fluids and the new combined fluid. Generally, surface reaction methods will adjust the parameters of existing particles in order to simulate the creation of a third.

2.5.3 Buoyancy

Buoyancy is the force which keeps objects floating on a fluid. The buoyancy force opposes the force which pulls an object downward. If the buoyancy force is larger than the force that pulls an object down, it will float. Archimedes' principle states that buoyancy is the weight of the fluid being displaced. So the deeper an object goes in a fluid, the higher the displacement, pressure and the buoyancy will be. An interesting phenomenon in the case of fluid to fluid interaction is that when the interacting fluids have a different rest density, the less dense fluid will be caused to rise inside the denser fluid. What actually happens is that the pressure of the particles will not only push particles which are close to each other away, but it also pushes particles away in order to reach their rest densities.

2.6 Conclusion

In this chapter we have described the theory of fluid mechanics we have given a detailed description of how fluid simulators work. Furthermore we have described related work in the field of fluid simulation and the Smoothed Particle Hydrodynamics is explained in depth. We have also provided mathematical knowledge relevant to fluid dynamics in a concise manner.

We will expand the Smoothed Particle Hydrodynamics technique by adding heat conduction to the equations. This means that we will add temperature as a new fluid property. Existing work on solidification attempts to recreate realism by either adjusting existing fluid properties in simulators which has the drawback of introducing numeric instability. By adding temperature based on the laws of thermodynamics we will offer more realistic simulations of fluids without having to constrain timesteps in order to counter the numeric instability. Finally, we will add phase transitions to the simulator. This offers for example the possibility to create a simple mesh when a fluid solidifies, rather than having to simulate a solid object as particles like existing work does. The next chapter will cover the heat conduction, temperatre addition and phase transitions in depth.

Chapter 3
Solidification

Solidification is the phase transition in which a liquid turns into a solid. In this chapter we will describe our implementation of Solidification as an extension of SPH. As water is so abundantly present in our daily life, it is the liquid of choice for the demonstration of our solidification solution. The states in which water can exist are liquid, solid and gaseous. In this chapter we will discuss the phase transition of liquid to solid (solidification) and of solid to liquid (melting) and how we have extended SPH with this functionality.
As general testcase for our simulation we have used the solidification of a lake. Since one of our subgoals is to offer a more realistic fluid simulator for applications such as games, we have chosen a case which offers extendability. By simulating the solidification of a lake, we introduce the requirement of more advanced domains which can be used for other fluids such as lava whereas more basic cases, for example a glass of water, would offer less extendability. However, our solution can also be used for such basic cases.
Water can be frozen by adjusting its pressure and temperature. In this chapter we will describe the physics of freezing and how this translates to the solidification of a lake of water. A required addition to the SPH method is the transferring of heat in order to change the temperature of the fluid. We will describe how heat transferring works and how heat conduction can be used in combination with a fluid simulator. In order to solidify a liquid, a phase transition has to be triggered. In the final section of this chapter we will describe phase transitions and how we manage phase transitions through heat transferring in our implementation.

3.1 Physics of Freezing

The first steps in designing a solidification method is to understand the physics of freezing. In this chapter we will first describe how a liquid freezes. Secondly we will describe what happens to the liquid when this phase change occurs and how this applies to a realistic simulation case such as the solidification of a lake.

A liquid can turn to its frozen state in several different ways. We will describe these methods for the most basic liquid: water. The first and most recognizable way of freezing

water is by changing its temperature. When water goes below its freezing point of zero degrees Celsius it will freeze and when it exceeds the freezing point it will melt. At this freezing point a phase change occurs. These phase changes are described in Section 3.3. Liquids can also be frozen by reducing the pressure of the liquid. The freezing point of water is also related to its pressure. Reducing the pressure of a liquid increases its freezing temperature, as is shown in Figure 3.1 which denotes the solid, liquid and vapor phases of water and their transition points dependent on pressure and temperature

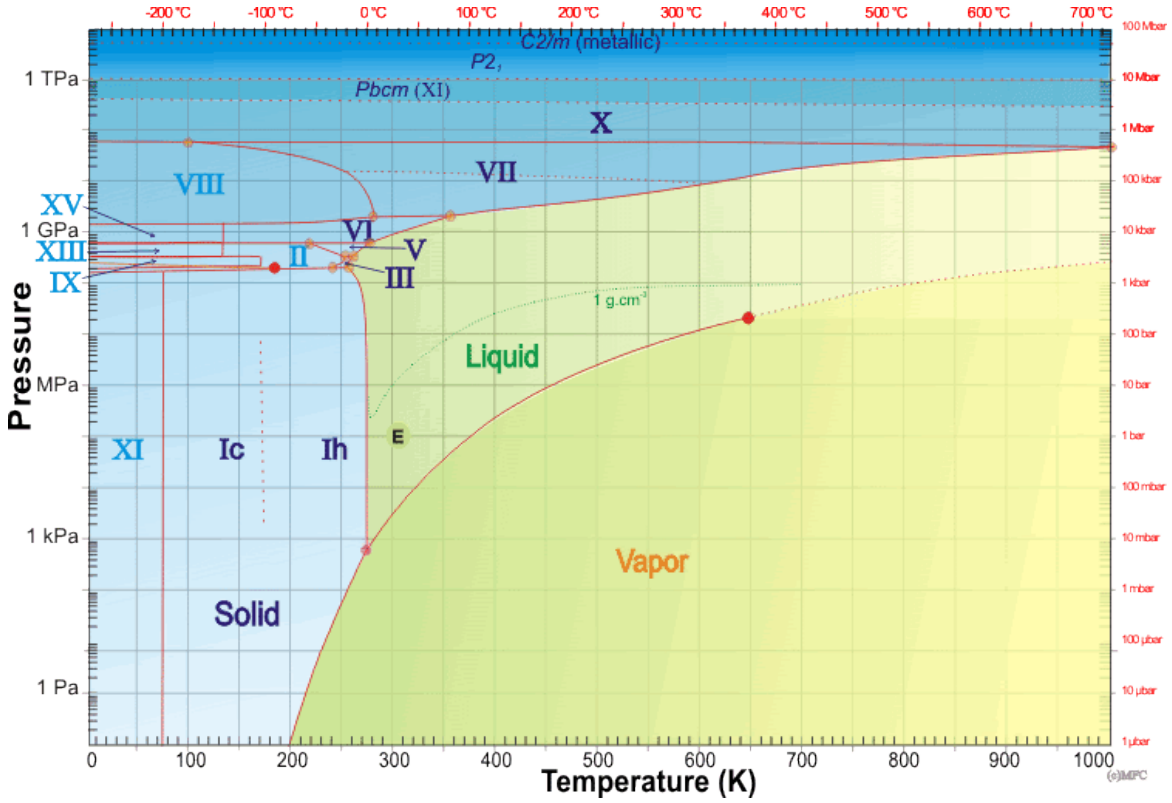


Figure 3.1. Water phase diagram from [12]. 273.15 Kelvin is 0 degrees Celsius. The Roman numbers indicate phases of ice.

3.1.1 Temperature based fluid properties

There are two fundamental properties of a fluid incorporated in fluid simulators which are based on temperatures. These are the viscosity and the density [30]. In traditional fluid simulators the relationship between viscosity and density is not modelled as they do not include temperature in their simulation. In our solidification extension we also model viscosity and density of water as functions of temperature. For the data in this section we assume a standard atmosphere (atm) value of 1 (= 1.0197 bar) for pressure.

Viscosity of water

The viscosity of a liquid is the measure of its resistance to angular and shear deformation. A higher viscosity value will result in a fluid which looks stickier such as syrup, whereas a

low viscosity fluid will appear more liquid such as water. Note that a more viscous looking fluid does not always mean that the fluid is also actually more sticky. A fluid which has no resistance to shear stress (zero viscosity) is called an ideal fluid. The viscosity of water depends on its temperature. Figure 3.2. shows the viscosity of water at certain temperatures.

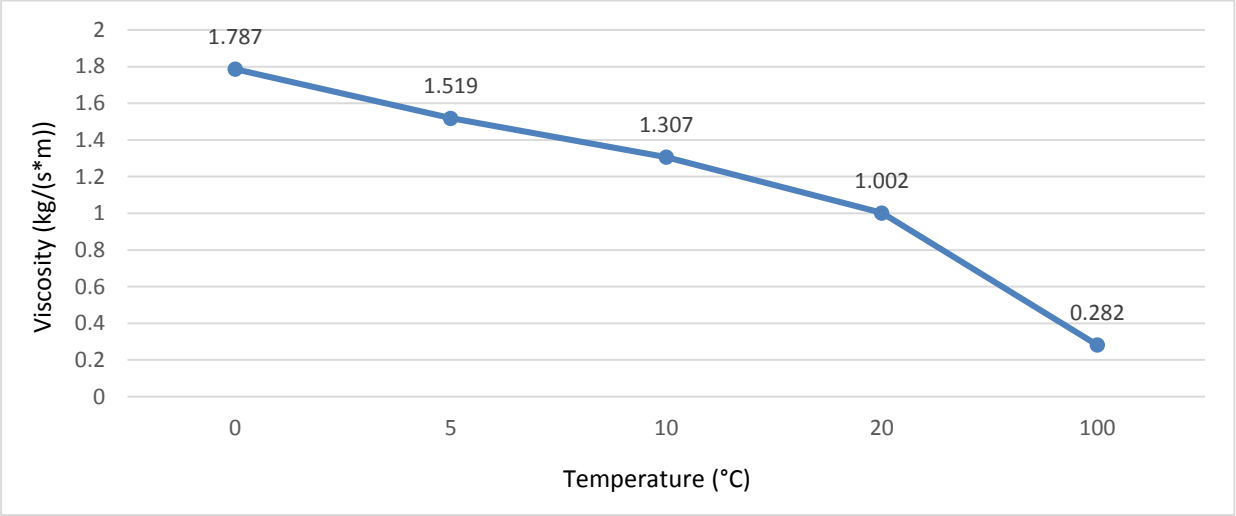


Figure 3.2. The viscosity of water at certain temperatures [31].

Density of water

Density of an object is its mass per unit volume. The density of water depends on its temperature. This relation is however not linear. Figure 3.3 shows the density of water at certain temperatures.

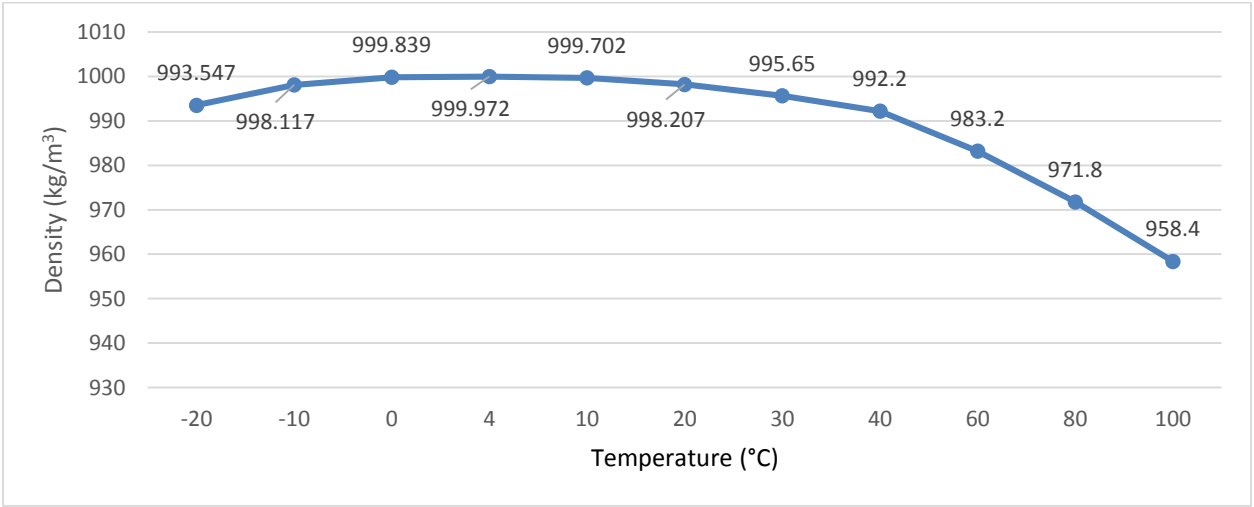


Figure 3.3 The density of water at certain temperatures [32].

3.1.2 Lake Solidification

When water freezes or solidifies it expands and becomes less dense. The expansion happens because the H_2O molecules start to bond and will thus take less space in comparison to a liquid. A solid object will retain its shape due to the molecules having larger attractive forces towards each other.

From Figure 3.3 can be deduced that liquid water gets denser as the temperature lowers and nears four degrees Celsius, where it starts getting less dense. When a liquid water fluid is in contact with a colder gaseous air fluid which resides above the water, the top layer of the water will cool down. As the top layer of the water cools down, the density will increase and thus the colder molecules will sink, resulting in the warmer water below to rise. This process will continue until the water reaches its turnover point of four degrees Celsius. As seen in Figure 3.3 water actually gets less dense from this point onward resulting in a layer of water having a smaller density than the water underneath. This layer of water then continues to cool and eventually freeze and form the first layer of ice on a lake.

3.2 Heat Transfer

In this section we will describe the theory of heat transfer and how this is strongly related to fluid dynamics. Afterwards we will describe the technique we have used in order to achieve the transferring of heat and changing the temperature of the particles in our fluid simulator.
Heat transfer is the description of the exchange of thermal energy between different objects. Thermal energy is the partial energy of the collection of energy of an object which results in temperature. Heat is the energy which is transferred from a hot object to a colder object. Temperature is the measure of average speed of the molecules within an object. When heat is transferred from an object with a high heat value, a hot object, to an object with low thermal energy, a cold object, it produces entropy. Entropy is important for fluid dynamics and has a strong relationship to the second law of thermodynamics. Both of these subjects are fundamental for heat transfer and will be explained further along this section. Our goal for the heat transferring process is to be able to transfer heat between different particles in our simulation and to transfer heat from the environment to the particles. Phase changes occur at certain temperatures based on which material the fluid consists of. More about phase changing can be found in the next section of this chapter.

3.2.1 The Second Law of Thermodynamics

The first law of thermodynamics states that the internal energy of an object or system changes as heat flows in or out of it because the law of conservation of energy states that the total energy of an isolated system cannot change. The second law expands this theory by stating that in an isolated system, *entropy* can only increase [33]. This means that the

temperature of two different objects will eventually reach equilibrium but that thermal energy will be lost. Other properties which try to reach equilibrium are pressure and density. The equilibrium value which properties are attempting to reach will be referred to as resting states.

3.2.2 Entropy

In the field of thermodynamics, entropy is commonly defined as the measure of progression towards the thermodynamic equilibrium state. This equilibrium state can be referred to as maximum entropy. The entropy of an object can be measured as the amount of energy that has been spread out in a process divided by the constant absolute temperature [34]:

$$\Delta S = \int \frac{dQ}{T}$$

where

- ΔS is the change in entropy
- dQ is the heat transferred into the object
- T is the absolute measure of temperature of the object

Eq. 26

Boltzmann hypothesizes that molecules in an object have a rest position [35]. If neighbouring molecules come close to each other, they repel each other. But if they move further away, this turns into attraction. We can use this hypothesis for molecules for the particles in our fluid simulation. In the case of a phase change, the rest position of particles will change, when heated, they get pushed apart.

The following example of entropy and the second law of thermodynamics describes the property which is interesting for our simulation. When a domain containing a liquid, such as a glass of water or a lake in the mountains, is contained by a larger domain which has a higher temperature, the temperatures of the domain and the liquid will equalize to the same temperature. Heat will be transferred from the warmer domain into the colder object until the temperature of both is the same. In terms of entropy, the entropy of the colder liquid has increased and the entropy of the warmer domain has decreased. Since the liquid system is smaller than the domain and will equalize more towards the temperature of the domain, the entropy of the liquid has increased more than the entropy of the domain has decreased, thus the complete change in entropy of the complete system (the domain and the liquid together) has increased [36].

3.2.3 Heat Conduction

There are four governing mechanisms which can transfer heat: conduction, convection, advection and radiation [37]. Conduction is the mechanism where heat is transferred from neighbouring molecules when these interact with each other. Conduction is the heat transfer mechanism which happens within an object. The thermal conductivity of an object describes how much heat is conducted between particles when they are in contact. The amount of thermal conductivity changes based on the properties of an object. Solid objects have a higher thermal conductivity than fluids. And ultimately, gases have the least thermal conductivity of the states.
Convection is the heat transfer mechanism where heat is tranferred due to the mass movement of molecules, it is the transfer of heat in the vertical direction. An example is that a warm air fluid is less dense than a cold air fluid. This results in the warmer air rising and the cold air descending as the cold air is heavier. Another example is a pan of cold water heating up from the bottom, warm water will rise and the colder water will get heated.
Advection is the heat transfer mechanism where heat is tranferred in the horizontal direction. An example is the wind moving molecules around.
Lastly, radition is the heat transfer mechanism which transfers heat through waves of energy. The most common example is the sun radiating heat towards the earth.
In our simulation we will require the conduction and the convection mechanisms. Heat will be transferred between particles through conduction, and the change in density due to the temperature will result in the convection-based heat transferring.

3.2.4 Computing Heat Conduction

Heat transferring is a challenging subject in SPH fluid simulation because of the computational heavy nature of the equations. However, making some assumptions can result in simpler heat conduction methods and methods suited for real time simulations. We use a particle strength exchange (PSE) which is a particle method for simulating diffusion processes in a continuous space [38]. For this, particles have to be treated as carriers of heat. The transferring of heat can then be computed using conduction [39].

For the heat conduction we refer to Fourier's law of heat conduction. This law states that heat transfers in the opposite direction of the temperature gradient. In other words, heat transfers from hot to cold. Equation 27 displays Fourier's law of heat conduction [53].

$$Q = -k\nabla T$$

where

- Q is the heat flux
- k is the thermal conductivity coefficient
- ∇T is the temperature gradient

Eq. 27

Heat flux is the rate at which heat transfers (flows) to a certain surface, per unit area and unit time. It is proportional to the gradient of the difference in temperature. The thermal conductivity of a material is related to the temperature of the material, but since this difference is so small for water, we can treat it as a constant [53]. The units in which thermal conductivity is represented is watts per meter per Kelvin (W/mK).

The following partial differential equation describes how heat diffuses through a fluid, by indicating what the change in temperature is. This is the equation we will use in our simulation [40].

$$\frac{DT}{Dt} = k\nabla^2 T$$

where

- $\frac{DT}{Dt}$ is the change in temperature
- k is the thermal conductivity coefficient
- $\nabla^2 T$ is the temperature Laplacian

Eq. 28

Normally computing the Laplacian on the right hand side of Equation 28 would require computationally heavy code. However, since the used base SPH implementation already includes nearest neighbour searching, we can use this to find nearby particles. We can use the difference in temperature between the neighbours.

3.2.5 Implementation

In our simulation we assume that the terrain which surrounds the lake of water is so large that the heat transfer process between the particles and the terrain does not influence the overall temperature of the environment. In other words, the terrain around the lake has a constant temperature. If we were to simulate this with particles, an enormous number of particles would have to be generated for the terrain while the result would be equal to a constant temperature, for the cost of an increased simulation time. The implementation of the equations used for computing the heat transfer process will offer some important performance challenges and short-cuts. Following are the formulas and algorithms used for the heat transfer interaction between neighbouring particles and the interaction between particles and their environment.

Particle-Particle Heat Transfer

We compute the particle-particle heat transfer for each particle with its neighbouring particles. The heat transfer equation used for interacting particles is the following, which is based on the particle strength exchange method in [38]:

$$T_i(t + \Delta t) = T_i + \sum_{j \neq i} k_j (T_j - T_i) \Delta t$$

where

- T_i is the temperature of particle i
- T_j is the temperature of particle j
- k_j is the thermal conductivity of particle j
- Δt is the timestep

Eq. 29

Since particles keep exchanging heat until they reach equilibrium, the order in which particles get processed does not influence the result of the simulation visually enough to notice, however when the value for h is increased, this will influence the amount of temperature that is being exchanged for each timestep. Algorithm 3.1. is used for the heat conduction between particles.

Algorithm 3.1 Conduct Temperature

```
 1: function ConductTemperature ( int i, float dt )
 2:    foreach(int j in neighbourParticles)
 3:       // get temperature of both particles
 4:       tempHere = temperatures[i];
 5:       tempThere = temperatures[j];
 6:       // if the temperature of both particles is not the same
 7:       if(tempHere != tempThere)
 8:          // compute difference
 9:          tempDifference = tempThere - tempHere
10:          // compute exchange
11:          thermalConductivity = thermalConductivities[j]
12:          exchange = thermalConductivity * tempDifference * dt
13:          // add exchange to temperatures of particles
14:          temperatures[i] += exchange
15:          temperatures[j] -= exchange
16:       end if
17:    end foreach
18: end function
```

Particle-Terrain Heat Transfer

We assume that the volume of the terrain surrounding the particles is so large in comparison to the volume of the fluid, that the average temperature of the terrain will not be influenced. With this assumption we can use a constant temperature for the terrain which does not change over through interaction with the particles. Heat is transferred from the terrain to the particles whenever a particle gets close enough to the terrain. The collision check between the terrain and the particles is described in Section 4.2.2.
The following formula is used for computing the heat conduction for particles which collide with the terrain:

$$T_p(t + \Delta t) = T_p + k_t(T_t - T_p)\Delta t$$

where

- T_p is the temperature of particle p
- k_t is the thermal conductivity of the terrain
- T_t is the temperature of the terrain

Eq. 30

Particle-Air Heat Transfer

Equal to the assumption that the terrain surrounding the fluid has a constant temperature, we assume that the air above the fluid also has a constant temperature. Because of this, we

don't have to model the air fluid as a set of particles which provides us with more computational power to add more particles to the water fluid. With this in mind we can come up with the following equation for the heat transfer between air and the fluid:

$$T_p(t + \Delta t) = T_p + k_a(T_a - T_p)\,\Delta t$$

where

- T_p is the temperature of the interacting particle
- k_a is the thermal conductivity of the air
- T_a is the temperature of the air

Eq. 31

The change in temperature for a particle is computed by adding the temperature transfers from all other objects (i.e. neighbouring particles and environment).

3.3 Phase Transition

A phase transition happens when once state of matter transforms into another state of matter. The four possible states of matter are: solid, liquid, gas and plasma as seen in Figure 3.4.

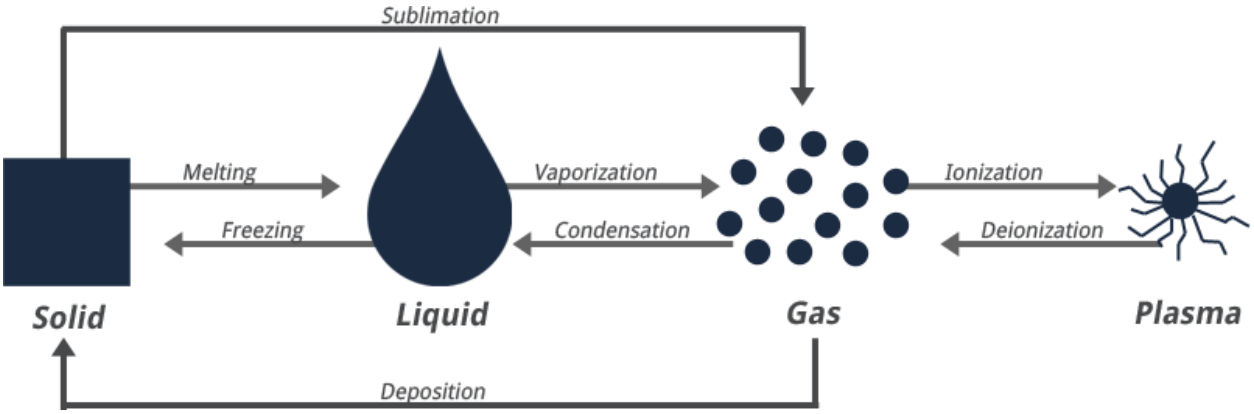


Figure 3.4 States of Matter.

When a phase transition occurs, the physical properties of a material can change. For example: the density of water changes drastically when it gets vaporized and transitions to gas. The phase transition which is relevant for our simulation is the one happening when an object goes from solid to liquid and from liquid to solid. We will not further discuss phase changes including gaseous and plasma fluids. Plasma is also not mentioned in Figure 3.1 because either a very high temperature is required to turn water into plasma (around 12.000K = 11.726 °C) or through other means independent of temperature.

3.3.1 Freezing

Liquid water makes the phase transition to ice when it reaches its freezing point of zero degrees Celsius. When this happens, the interaction between the new solid particles and the existing liquid particles will be different in comparison to the interaction when they were both liquid. More about the liquid to solid interaction can be found in Section 4.1.

For our frozen objects we use a mass-spring system . The basic idea of a mass-spring system is that particles will have attraction towards each other. When a particle turns solid it will freeze to other nearby solid particles. We solve the newly made bond between solid particles by creating constraints (the springs). Since there is no bending involved in ice, we will only have to add distance constraints between the new solid particle and the nearby existing solid particles which are inside its interaction radius. After new positions have been computed in a frame advance in the simulator, these constraints are satisfied in order to reposition the frozen particles to the positions defined in the constraints.

Since we don't require all the aspects of a mass-spring system, we will only describe the aspects which we require. The most common example of a mass-spring system is the one used in cloth simulation. Cloth simulation requires additional constraints such as shear and bend constraints which we do not.

Mass-Spring Systems

For the sake of simplicity we will describe the mass-spring method for two dimensions; however it can be adjusted to three dimensions with no extra work other than adding a third dimension to the equations. In a mass-spring system an object is modelled as a grid of $m \times n$ elements. These elements are then connected by massless springs which have a length which is larger than zero [41]. There are three different kinds of springs; structural, shear and flexion springs. Structural springs counteract tension by linking only the closest elements with each other. Shear springs connect shear elements and flexion springs handle bending [42]. Figure 3.5 displays those three different springs for clarification.

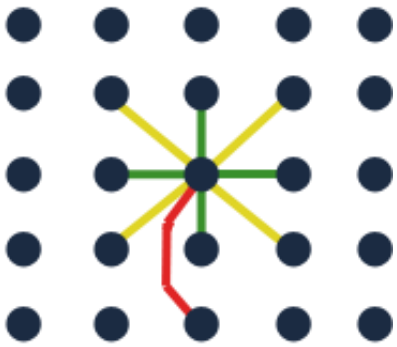


Figure 3.5: Different kind of springs. The green lines indicate structural springs. The yellow connections are shear springs and the red connection is a flexion spring.

Create Constraints

When a particle turns solid it will freeze to other solid particles near its interaction radius. This freezing is simulated by creating structural springs between said particles as seen in Algorithm 3.2. A new structural spring constraint is created between the position of the recently frozen particle and its neighbouring particles. The distance at the moment of freezing defines the *rest distance*. The rest distance defines the distance which the newly constrained particles will enforce during the satisfactory step of the simulation.

Algorithm 3.2 Create Constraints

```
1: function CreateConstraints(Particle p)
2:   foreach(particle np in neighbourParticles)
3:     distanceSquared = (np.x - p.x)² + (np.y - p.y)² + (np.z - p.z)²
4:     // if particles are close enough to each other
5:     if distanceSquared <= interactionRadiusSquared
6:         new Constraint(p, np, distanceSquared )
7:     end if
8:   end foreach
9: end function
```

Satisfy Constraints

Each time step of the simulation, the spring constraints have to be satisfied. Algorithm 3.3 is used to satisfy the constraints. The current distances of particles which have a spring between each other are compared with the distance bound to the constraint. When this difference exceeds an arbitrary error value, the particles have to be repositioned in order to satisfy the constraint. Repositioning is done by moving both particles towards each other by half of the difference in distance.

An issue which occurs with this method is that when particles are getting repositioned in order to satisfy a constraint, other constraints might not be satisfied anymore. This is countered by iterating through the constraints for a number of times. The higher this number is, the less error there will be in the satisfaction of constraints. When particles are further apart, and constrained to multiple particles, it will require more iterations in order to converge to a realistic result. Different fluid properties and environment parameters will influence the impact of the change in distance between particles and thus the exact number of iterations will differ per simulation setup. We have chosen an iteration count of 15 for this as this resulted in no visual errors.

Algorithm 3.3 Satisfy Constraint

```
function SatisfyConstraint(Constraint c)
    Particle p1 = c.p1
    Particle p2 = c.p2
    // current distance
    distanceSquared = (p1.x - p2.x)² + (p1.y - p2.y)² + (p1.z -
p2.z)²
    // constraint distance
    constraintDistanceSquared = c.restDistance
    // difference fraction between distances
    difference = constraintDistanceSquared / distanceSquared
    // if the difference is large enough to correct particles
    if difference > error
        // correct both particles towards each other
        correction = distanceSquared.length * ( 1 - difference )
        correctionHalf = correction * 0.5f
        p2.pos += correctionHalf
        p1.pos -= correctionHalf
    end if
end function
```

3.3.2 Melting

We handle the melting of a solid in the following manner: once a particle in a solid fluid exceeds its temperature melting point, we simply remove all the constraints which are related to this particle and continue the simulation normally. Other consequences which should be executed on the phase change can be defined, but this is not required for a water-based simulation. Density is a function of temperature in our simulation. As a positive effect this avoids the requirement of adding density changes to the phase transition between solid and liquid for our water-based simulation. However, when changing to a gaseous form, or in the case of other fluid compositions, the change of density is a consequence of the phase transition.

Chapter 4
Interaction

In this chapter we will describe how we manage the interaction with more advanced models other than the interaction between multiple fluids. First we will describe how we manage the interaction between a liquid fluid and solid fluids. Afterwards we will describe how we have incorporated a terrain into the simulation. We will describe how we take advantage of the already existing graphics pipeline used by the fluid simulation to render this terrain. And finally we describe how the interaction between fluids and the terrain is handled.

4.1 Frozen Objects

With our implementation where density and viscosity are both functions of temperature and where we use a mass-spring system for the implementation of solid frozen objects we do not have to make additional adjustments to the simulation in order for collision between liquids and fluids to work correctly. Since we do not create new geometry when particles are frozen to each other and form solid objects, we can just use the particle interaction from SPH to handle the interaction between solids and liquids.
The density of a frozen object is lower than liquid water above the phase transition point as shown in Table 3.2. As a result, the frozen particles will float on top of the liquid particles and they are kept together by the constraint satisfaction.

4.2 Terrain

Interesting for real-time applications is the interaction between a fluid simulation and the surrounding terrain. In this section we will describe which representation we have used to simulate this phenomenon and how we have implemented this. The fluid simulator which we use as the basis for our simulation extension is accelerated with the use of CUDA [50]. CUDA is a parallel programming model which harnesses the power of the graphics processing unit (GPU). The advantage of using a parallel programming model is that multiple particles can be simulated at the same time whereas traditional programming models only support one computation at a time. The use of this model introduces challenges because data has to be transferred to the GPU in certain formats in order for it to be used by the simulator. This section will also describe how we integrated our terrain data with the CUDA-based implementation.

4.2.1 Terrain Representation

In order to avoid having to manually enter the elevation data we use a height map to define the elevation of the vertices which in combination represent the terrain. Figure 4.1 shows one of these height maps used in our simulation. We load the height map as a texture, each pixel in the height map represents one vertex and the dimensions of the image define the width and length of the fluid domain. The elevation of the terrain can be deduced from the color channel. The grayscale value ranges from 0 – 255, the height of a vertex is equal to the grayscale value of its corresponding pixel in the texture (i.e. a grayscale value of 5 will result in a vertex with height 5 in unit sizes).

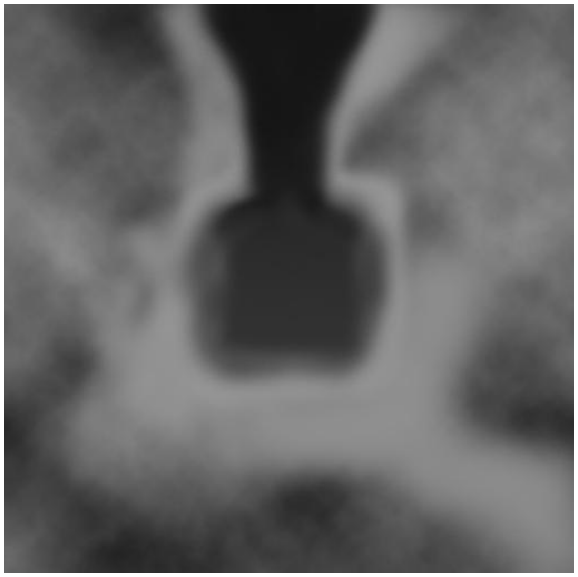


Figure 4.1 Height map of the terrain, used by testcases found in sections 5.3 through 5.6.

4.2.2 Particle Collision

Particle to particle collision occurs when neighbouring particles enter each others arbitrary interaction radius. The response to this collision is defined by the properties of the fluids in which these particles reside. In our simulation, there is no required distinction between interactions of particles in a solid or liquid because the difference in fluid properties will result in the required collision response. Collision between particles and objects which don't consist of particles should be managed in a different way. The collision between the particles in our simulation and the surrounding terrain is described in this section.

Since we compute the particle simulation on the graphical processor unit, we need the following information of the terrain on the GPU in order to calculate collision between particles and the terrain:

- Position of the vertices of the terrain
- Normals of these vertices of the terrain

Terrain collision is checked in each simulation step of the fluid simulator. Instead of using a rectangular domain in which the particles should reside, we can now use the terrain boundaries as the domain. This means that we don't have to use the general domain satisfactory check anymore. Algorithms 4.1 and 4.2 are used to check the collisions with the terrain. Algorithm 4.1 loops through all the particles and does a distance check between the particles and the terrain. When the distance is closer than an arbitrary chosen value ε there is a collision. Algorithm 4.2 is used to obtain the height of the terrain at a given *x,z* position. Since only the height data of the vertices of the terrain is known, an interpolation between the four vertices surrounding the *x,z* position is required in order to get the proper height value. Note that these algorithms are not optimized, solutions for this can be found in Chapter 6.

Algorithm 4.1 Check Terrain Collision

```
 1: function CheckTerrainCollision()
 2:   foreach(particle p in particles)
 3:     // get the height of the terrain at the particle x,z position
 4:     terrainHeight = GetTerrainHeightAt(p.x, p.z)
 5:     distance = p.radius - (p.y - terrainHeight) * simulation_scale
 6:     if(distance < EPSILON)
 7:       // Boundary collision, explained in section 2.3.2
 8:     end if
 9:   end foreach
10: end function
```

Algorithm 4.2 Get Terrain Height

```
 1: function GetTerrainHeight(float posX, float posZ)
 2:   // get the height of the terrain at the x,z position
 3:   int ix = (int)posX
 4:   int iz = (int)posZ
 5:   float topLeft = heights[(ix % 256 + (iz % 256) * 256)]
 6:   float topRight = heights [((ix+1) % 256 + (iz % 256) * 256)]
 7:   float botLeft = heights[(ix % 256 + ((iz+1) % 256) * 256)]
 8:   float botRight = heights[((ix+1) % 256 + ((iz+1) % 256) * 256)]
 9:   float fractX = posX - ix
10:   float fractZ = posZ - iz
11:   return interpolate(topLeft, topRight,
12:                      botLeft, botRight,
13:                      fractX, fractZ)
14: end function
```

4.2.3 Hardware Implementation

The computations for the collision between the terrain and the particles are done on the GPU in the simulation step of the fluid simulator. The generation and rendering of the terrain is not done during the fluid simulation steps. For this reason only information which is required for the collision computations have to be stored on the GPU. This offers some optimizations in regards to memory and speed. For example: we can construct the data collection which stores the terrain height map on the graphical device in such a way that we can retrieve a vertex without checking for its x and z position. The vertices are stored in a one-dimensional array ordered by adding the vertices on the x-axis to the array in an ascending order for each row on the z-axis, also in an ascending order. Figure 4.2 displays this ordering.

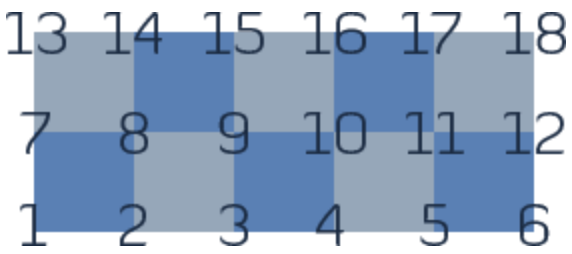


Figure 4.2: Ordering of vertices. The horizontal axis is the x-axis and the vertical axis is the z-axis.

This structure of the vertex position array would not be sufficient for the rendering of the terrain, however, for the collision computations it will be. In order to compute the height of the terrain at the position of a particle, we can obtain the index of the surrounding vertices and interpolate between their heights, as shown in Algorithm 4.2.

4.2.4 Rendering

It is in important that the rendering of the terrain does not generate any overhead for the simulation. With this in mind, the rendering of the terrain is done as an extension of the governing rendering step. We render the terrain using the OpenGL renderer which also renders the simulation. Rendering is done through a texture blending technique which is implemented on a fragment shader. Blending of the terrain textures is done based on the height of a vertex as shown in Equation 32. We define height-ranges for textures for which weights can be computed. Weights for each texture are then multiplied by their corresponding texture color and these are finally combined additively for the final result. Figure 4.3 shows an example of a rendered terrain with this technique which makes use of the textures shown in Figure 4.4.

$$W = \frac{R_{max} - R_{min} - |(T_{height} - R_{max})|}{R_{max} - R_{min}}$$

where

- W is the weight of this height region
- R_{max} is the maximum height value for this region
- R_{min} is the minimum height value for this region
- T_{height} is the height of the terrain at the to be computed pixel

Eq. 32

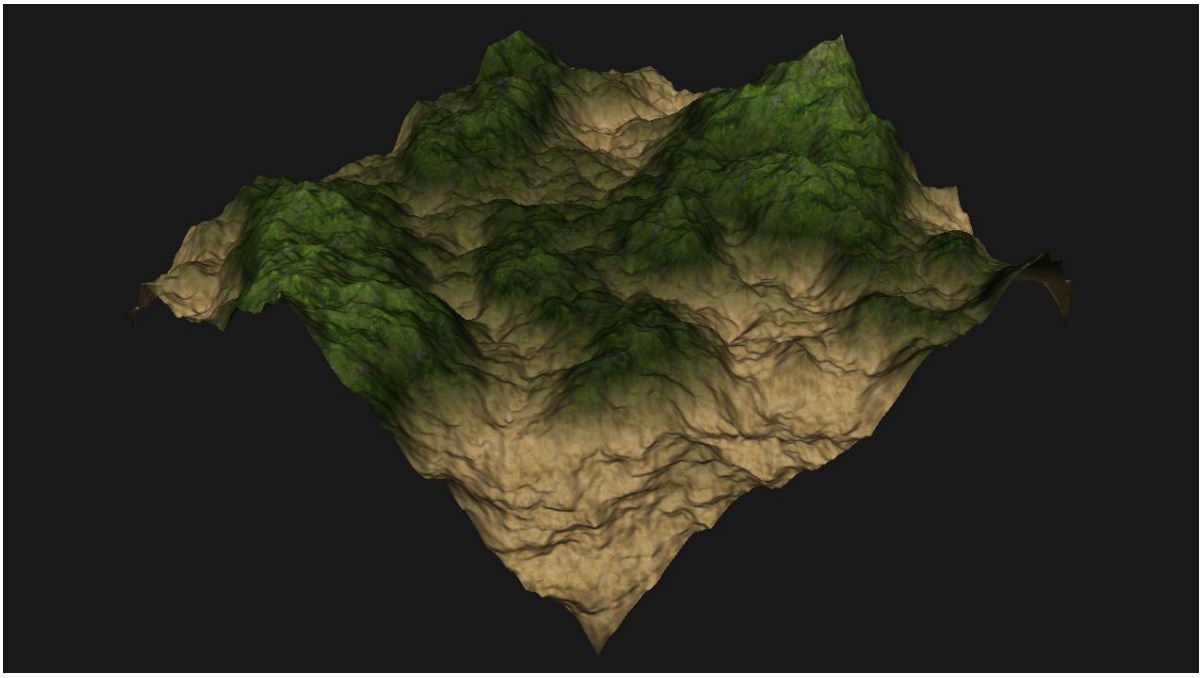


Figure 4.3. A rendered terrain using the texture blending method of Equation 32.

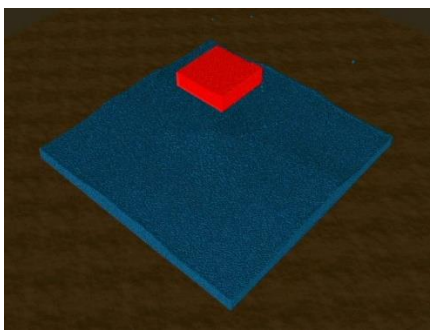
Figure 4.4. A combination of terrain textures.

Chapter 5
Results

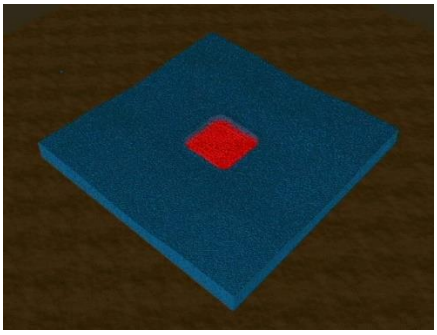
In this section we will present some of the results. We have a collection of scenes which demonstrate the contributions of our research. They demonstrate heat conductivity, phase transitions between liquid and solid and the interaction with a terrain. Our final scene simulates the solidification of a lake; this particular scene makes use of all the aspects described in previous chapters. First, we will show screen captures of these scenes and describe what happens in the scenes. Since traditional fluid simulators can run in real-time on current hardware, it was important for our research that we don't add too much computation time to the simulation and risk that the simulation would become too slow; therefore we have added a brief performance analysis of the added functionality in comparison to the bare bone Smoothed Particle Hydrodynamics implementation.

5.1 Scene: Heat Conduction

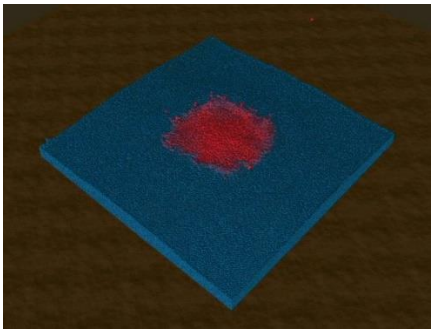
524,288 particles.



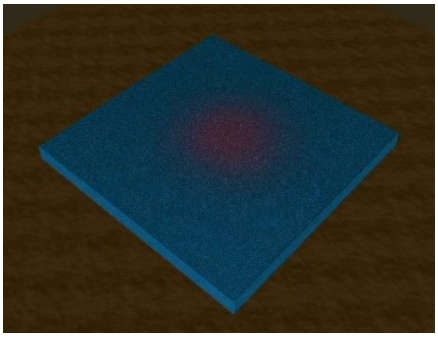
t = 0 seconds



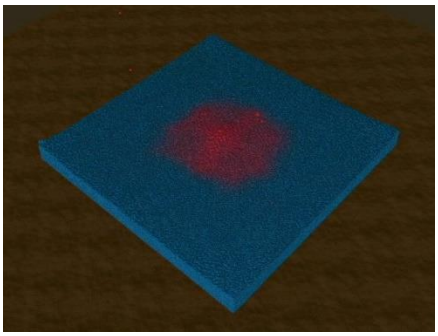
t=10 seconds



t = 20 seconds



t =30 seconds



t = 40 seconds

Figure 5.1. Scene: Heat Conduction images.

The heat conduction scene demonstrates the addition of temperature to the simulation. In the first image, the initial state of the scene is shown. Once started, the hot red fluid is being released and will fall into the cold blue fluid. Eventually the temperature of the red fluid will go down due to the colder fluid being larger. If it were the other way around, the cold fluid would actually warm up. The temperature of the combined fluid will eventually always reach equilibrium.

5.2 Scene: Terrain Interaction

1,048,567particles.

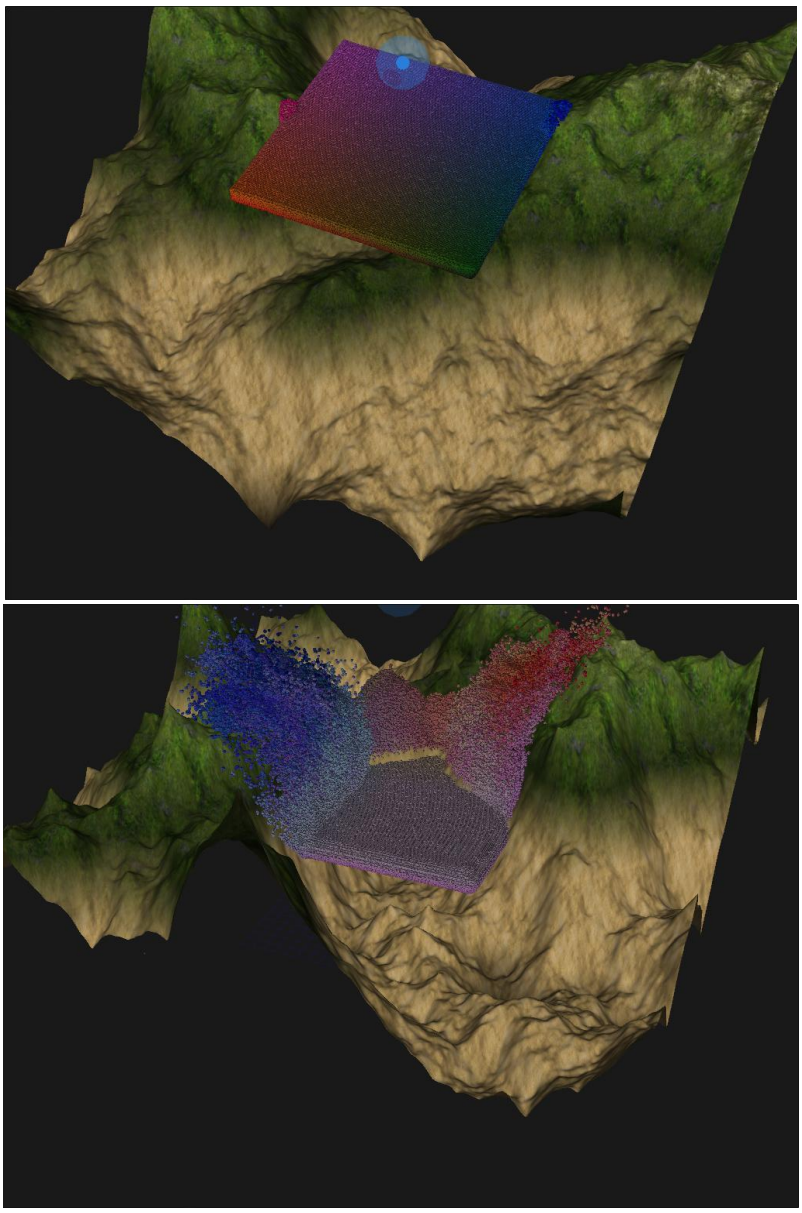


Figure 5.2. Scene: Terrain Interaction images.

This scene demonstrates the interaction between the fluid and a more advanced geometrical object such as a terrain. The initial state of the scene is displayed in the first image. Once started, the particles start interacting with the terrain as shown in the second image. It is shown that particles collide with the terrain and result in a visually pleasing simulation.

5.3 Scene: Bucket Terrain

2,097,152 particles.

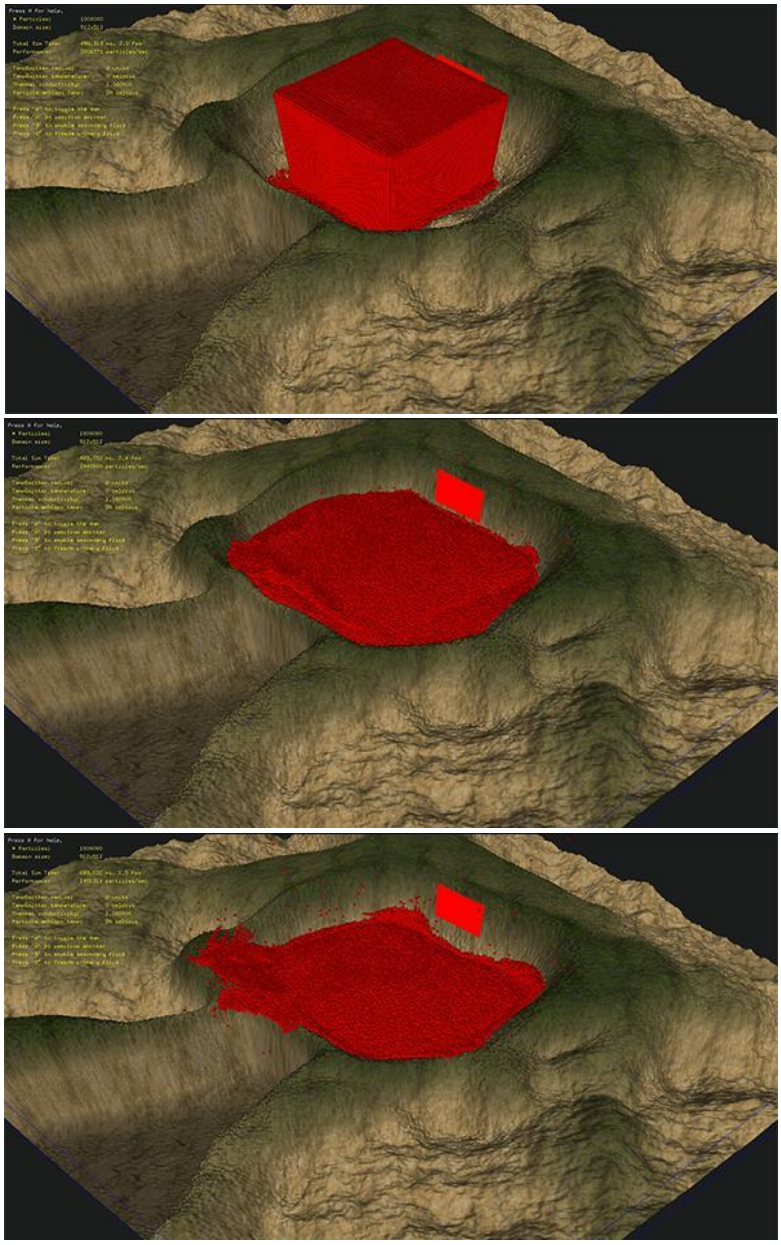


Figure 5.3. Scene: Bucket Terrain images.

This scene demonstrates a realistic test case where a fluid is contained by a surrounding terrain, which replaces normal domain constraints of fluid simulators by an advanced domain. In this particular scene we fill this 'bucket' with a large fluid as shown on the images above. Since the number of particles is too large it will actually overflow the boundary because of the pressure of the fluid as seen in the bottom image. The red square can be used as an emitter for additional particles.

5.4 Dam Break

1,048,567particles.

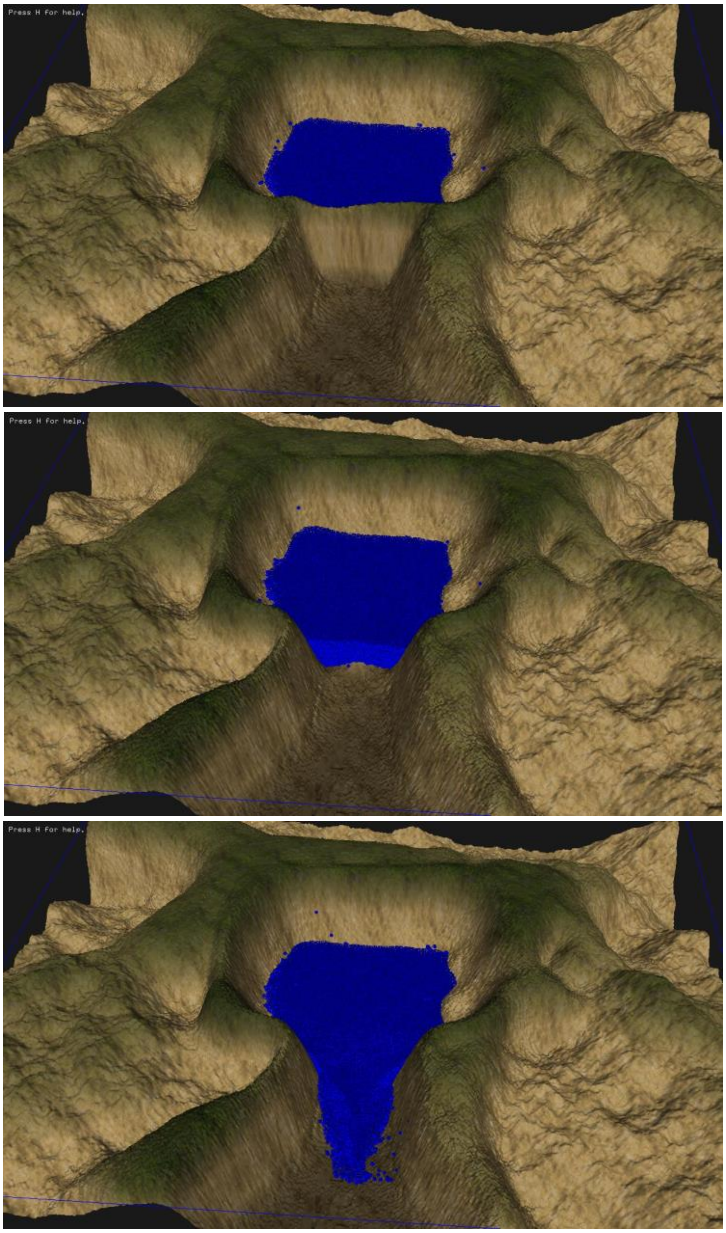
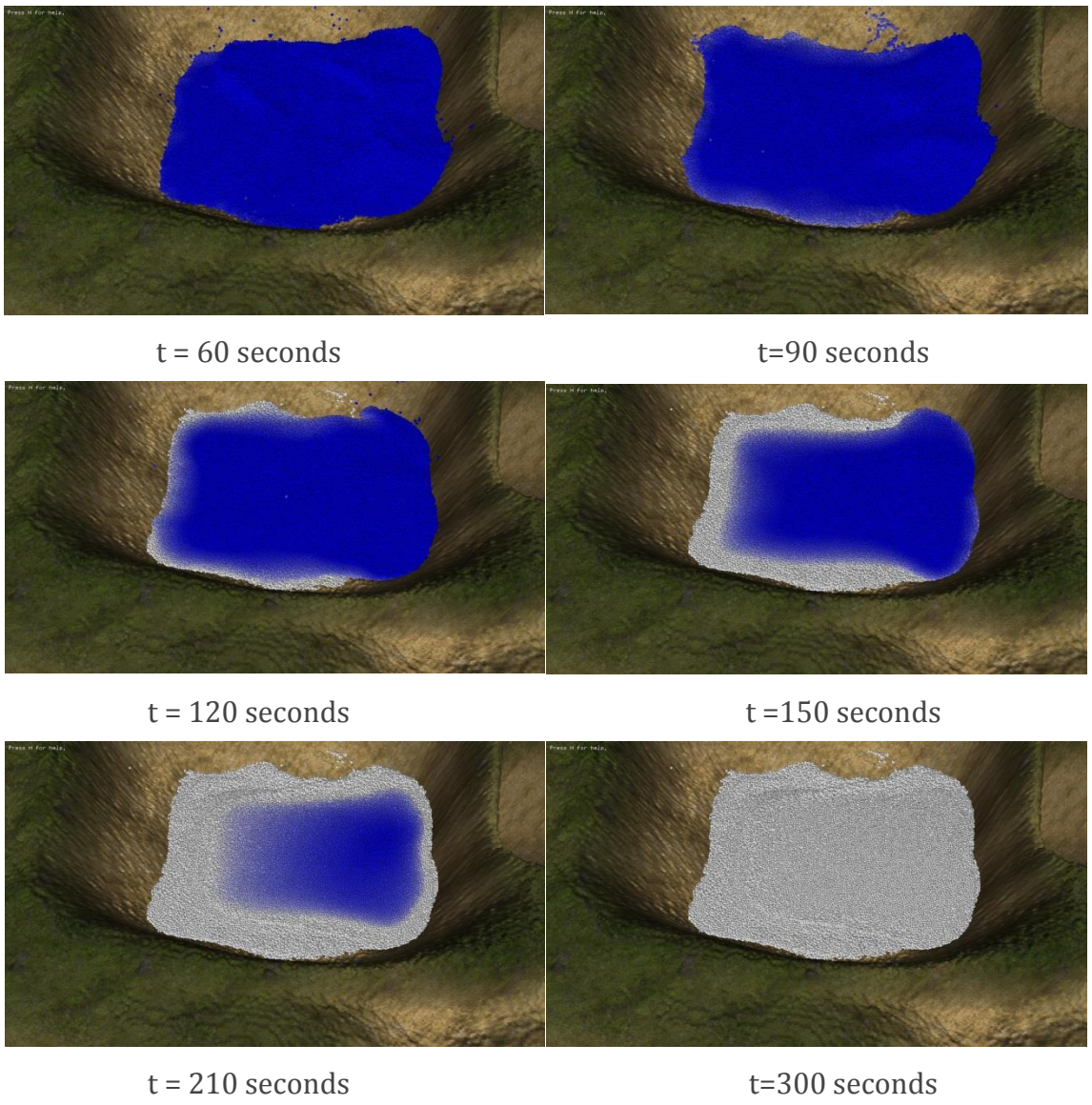


Figure 5.4. Scene: Dam Break images.

In the ‘dam break’ scene we demonstrate a scenario which could be plausible in a real-time interactive simulation. In the first image from above the dam which is blocking the fluid from leaving the ‘bucket’ is still intact. In the second image, this dam is broken and thus the fluid starts flowing out of the bucket. The last image shows the flow of the fluid after the removal of the dam.

5.5 Scene: Quick Freeze

1,048,567particles.



t = 60 seconds t=90 seconds

t = 120 seconds t =150 seconds

t = 210 seconds t=300 seconds

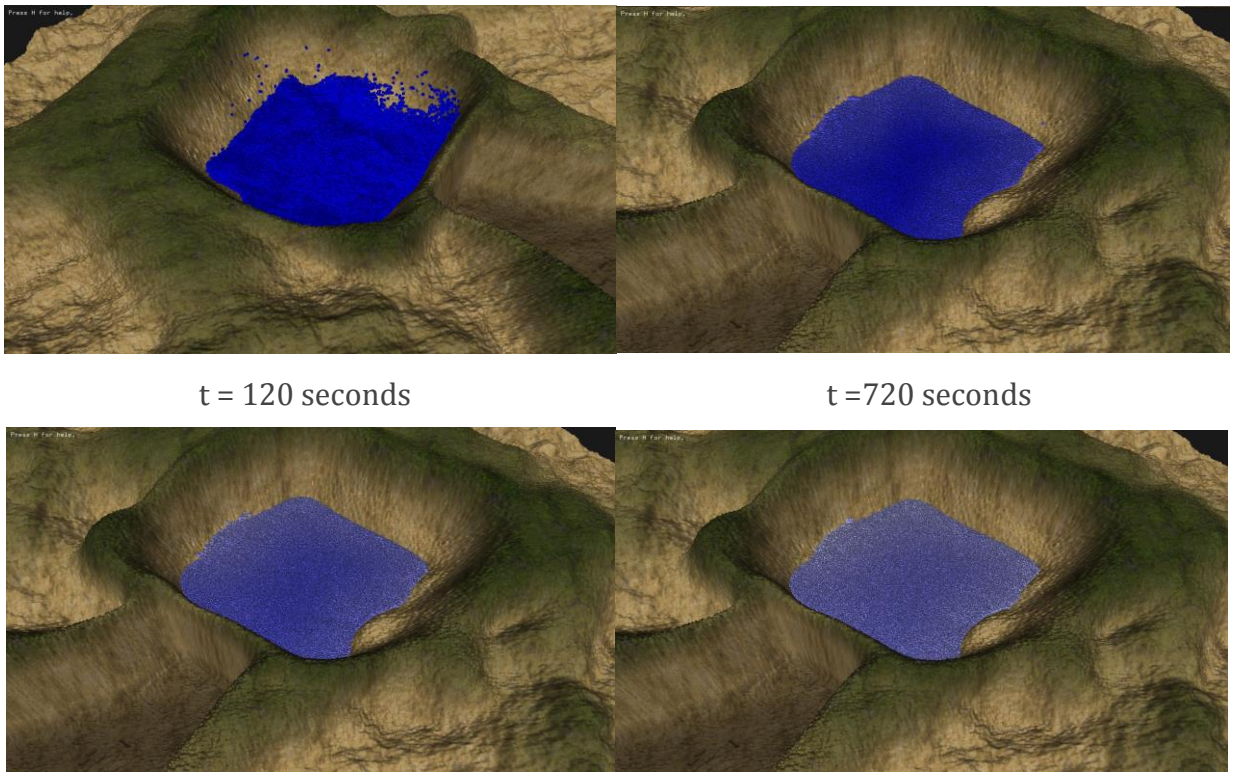
Figure 5.5. Scene: Quick Freeze images.

One situation we have encountered is that in some real-time applications the solidification of a fluid should happen faster than in the real world. There is a distinction between two types of quick freezes. The first one being a quick freeze where the freezing process is accelerated by decreasing the time scale of the simulation (i.e. simulating 1 second is done in 0.5 seconds.) This quick freeze will result in a more rapid freezing simulation which has the same visual end state as a real world time scale simulation. The other quick freeze simulation is the one where the difference in temperature is so large, that the liquid fluid will freeze faster due to large change in temperature. This particular simulation is more

interesting for applications because this is used to freeze something manually in the medical field for example. The latter explained quick freeze simulation is demonstrated above. We can insert a very low value for the surrounding environment (terrain and air) such as -200 degrees celsius. The outer particles of the fluid, the ones closer to the terrain, will freeze faster as the simulation has no time to reach equilibrium.

5.6 Scene: Normal Freeze

1,048,567 particles.



t = 120 seconds t =720 seconds

t = 2040 seconds t = 3420s seconds

Figure 5.6. Scene: Normal Freeze images.

The above images show the scene where we have used a slower freezing method in comparison to the 'Quick Freeze' scene. In the 'Normal Freeze' scene, the temperature of the terrain containing the water is zero degrees Celsius. This results in a more realistic behaviour as the fluid has enough time to reach an equilibrium state of its temperature constantly. The number of particles influences the rate at which a fluid freezes, above simulation uses approximately 1,048,567particles. Freezing this particular simulation took around one hour. For comparison, the 'Quick Freeze' scene took around 5 minutes to completely freeze for the same number of particles.

5.7 Performance

In this section we make a brief performance analysis by comparing the performance of the SPH implementation without our extensions and the implementation with our extensions. We will do this by describing how much extra computation time it takes for each extension. For this analysis, the normal freezing scene from section 5.6 is used. We've run each setup for two minutes and we take the average values as benchmarks. We also display the average number of particles which are calculated per second as this gives a better understandable visualization of the performance. At the end of this section, a graph is shown so different setups can easily be compared.

The measuring is done with the following hardware:
- Graphics Processing Unit: *GeForce GT 750M.*
- Central Processing Unit: *Intel Core i7-4700HQ, 2.4GHz.*
- RAM: *8.00 GB.*

Note that this is a rather poor Graphics Processing Unit. High-end computer systems will be able to simulate fluids a lot faster.

No Extensions

Number of Particles	Avg. Time per Frame (ms)	Avg. Particles per Second
32,768	4.02	8,151,243
524,288	92.08	5,693,831
1,048,576	222.74	4,707,623
2,097,152	632.21	3,317,172
4,194,304	2000.87	2,096,240

Table 5.1. Time comparison of the SPH simulation without extensions with different number of particles setups. The column 'Avg. Particles per Second' contains how many particles could be simulated within a second with the given setup.

Extension: Heat Diffusion

Number of Particles	Avg Time per Frame (ms)	Avg Particles per Second
32,768	4.43	7,396,839
524,288	99.89	5,248,653
1,048,576	232.72	4,505,740
2,097,152	718.6	2,918,385
4,194,304	2247.82	1,866,026

Table 5.2. Time comparison of the SPH simulation with the 'heat diffusion' extension with different number of particle setups.

Extensions: Constraints & Heat Diffusion

Number of Particles	Avg Time per Frame (ms)	Avg Particles per Second
32,768	4.56	7,185,964
524,288	108.17	4,484,889
1,048,576	261.56	4,008,931
2,097,152	755.64	2,775,332
4,194,304	2485.23	1,687,692

Table 5.3. Time comparison of the SPH simulation with the 'constraints' and 'heat diffusion' extension with diferent number of particle setups.

Extensions: Environment Interaction & Constraints & Heat Diffusion

Number of Particles	Avg Time per Frame (ms)	Avg Particles per Second
32,768	5.82	5,630,240
524,288	117.3	4,469,633
1,048,576	271.24	3,865,860
2,097,152	763.75	2,745,861
4,194,304	2493.51	1,682,088

Table 5.4. Time comparison of the SPH simulation with the 'environment interaction', 'constraints' and 'heat diffusion' extension with diferent number of particle setups.

Deducible from above data is that the addition of heat diffusion has the impact of slowing down the SPH implementation by approximately 9% for low particle counts and 11% for higher particle counts. The addition of the terrain and air in combination with the heat diffusion and the constraint satisfaction slows the original SPH implementation by approximately 31% and on higher particle counts by 19.5%. This makes sense as the initial impact of rendering the terrain does not scale with the number of particles, however the interaction with the terrain and the satisfaction of constraints as there will be more constraints does scale with the number of particles in the scene.

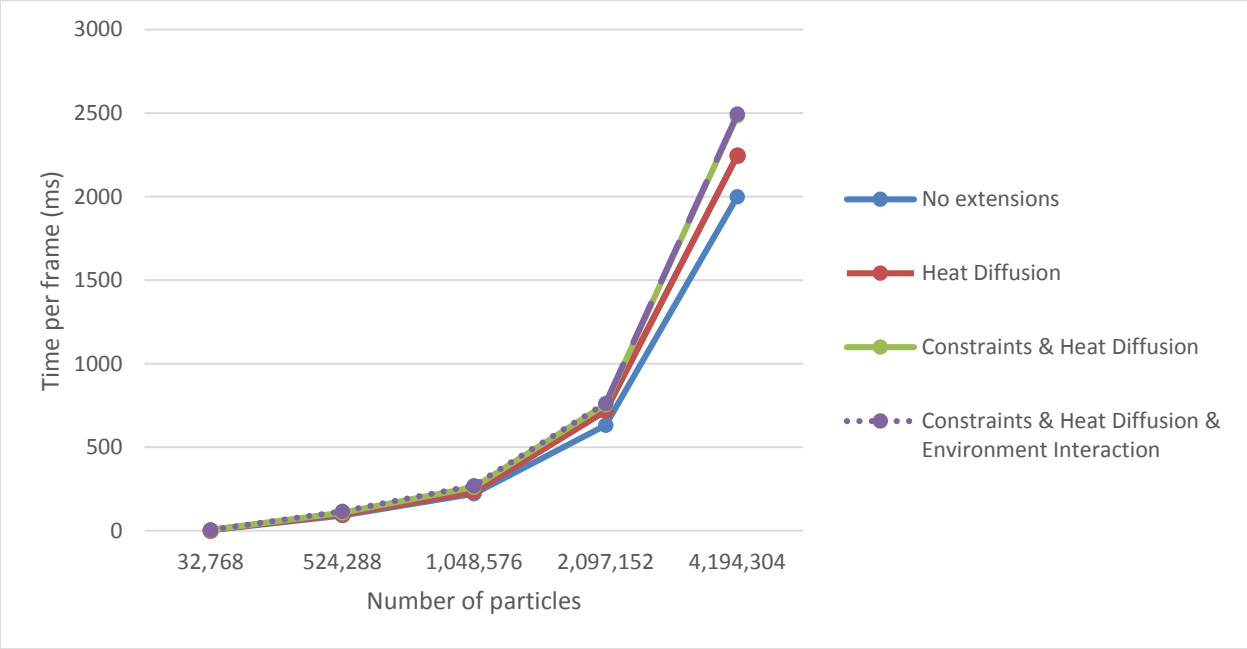


Figure 5.6. Plots of all performance scenario's in one figure.

Chapter 6
Conclusion

In this thesis, we have presented an advanced Smoothed Particle Hydrodynamics method which incorporates the phase transitions of freezing and melting for fluids. We have shown that it is possible to solve the temperature computations of a fluid independently of its current phase. In order to achieve our results we have modified an existing Smoothed Particle Hydrodynamics method to integrate heat conduction.

6.1 Contributions

We contribute to the field of fluid simulation by extending Smoothed Particle Hydrodynamics with more advanced fluid properties and interactions. In this section we will summarize the proposed extensions.
The basis for our contributions is the addition of adding the temperature property to fluid particles. With this temperature addition in place, we have added heat conduction functionality to the fluid simulator by using the nearest neighbour search algorithm already in place. Following these extensions we define the density and viscocity of particles as functions of temperature.
Another contribution is the phase transition mechanism. Different consequences for different types of phase transitions and different fluid compositions can be accomplished with this mechanism. And as final addition we have added more advanced interactions such as the interaction with terrain boundary.

The current state of our implementation is technically sufficient for real-time interactive applications as it runs real-time. Yet, some future work will still have to be done such as the rendering of the fluid's surface. In the following section we will describe future work which would enhance the result of this research.

6.2 Future Work

In this section we will describe some future work which will enhance the fluid simulation methods even more. Some of this work has been done in existing research but not in the combination with solidification and phase changes. What the current state of fluid simulation lacks is an unified method which combines research into one advanced simulator.

Surface rendering is one of these fields which have been researched before. Some existing research consists of the following: point-based visualization of metaballs [43], marching cubes [44] and screen space curvature [45]. These researches already provide visually appealing results, however as stated before, making an unified method is the biggest challenge.

Our current implementation of the terrain environment is straight forward and not optimized. Adding spatial divisioning such as an octree would offer the possibility to avoid having to compute collision between the environment for particles which are not near the terrain [54].

The time it takes to freeze a fluid is dependent on the number of particles in this fluid. Adding the simulation scale to the heat conduction equations could result in a scalable way to manipulate the temperature of the fluid. With this, the freezing or melting of larger fluids can be simulated with a relatively low number of particles. However, more research should be made in order to confirm this.

In our research we are only working with the fluid water. However, fluids which consist of multiple types of alloy will influence the way in which the fluid simulator should handle them. Not only properties such as the melting and freezing points of fluids but also the generation of new objects once a phase change occurs is interesting. When a phase change occurs in high order alloy systems, multiple new fluids or solids might be created rather than just one. This also means that phase changes in such systems might not be reversible such as in the case of water and ice. For example, once lava solidifies, gases and solids are formed. Melting the formed rock will not once again result in lava.

In the real world, on the surface of ice there is always a small layer of water. This small layer of water is actually the reason why ice melts together rather than the closest solid particles as we assume in our research. Extending our research with this phenomenon would enhance the realism of the simulation.

References

[1] Archimedes of Syracuse. "The works of Archimedes". p. 257. Retrieved 11 March 2010.

[2] R.A. Gingold and J.J. Monaghan. “Smoothed particle hydrodynamics”. Monthly Notices of the Royal Astronomical Society 181, pp. 375-389, 1977.

[3] D. Cline, D.Cardon and P.K. Egbert. “Fluid Flow for the Rest of US: Tutorial of the Marker and Cell Method in Computer Graphics”. 2004

[4] B. Solenthaler, J. Schläfli and R. Pajarola. “A Unified Particle Model for Fluid-Solid Interactions. 2007.

[5] M. Kelager. “Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics”. 2006

[6] J. Stam. “Real-Time Fluid Dynamics for Games”. *Proceedings of the Game Developer Conference.* 2003

[7] J.Stam. “A Simple Fluid Solver based on the FFT”. *Journal of Graphics Tools, Volume 6, Number 2.* pp. 43-52. 2001.

[8] K. Erleben, J. Sporring, K.Henriksen, and H. Dohlmann. “Physics-Based Animation”. *Charles River Media*, 2005

[9] M.J. Kirkby. “Hillslope process-response models based on the continuity equation”. 1969

[10] M.J. Harris. “Fast Fluid Dynamics Simulation on the GPU”. *GPU Gems*, pp. 637-665, 2007.

[11] M.A. Shubin, “Laplace operator”. *Encyclopedia of Mathematics.* 7 Februari 2011. Web. 15 November 2013.

[12] A.Galbis and M.Maestre. “Vector Analysis Versus Vector Calculus”. 2012.

[13] M.D. Roberts. “A Fluid Generalization of Membranes”. 2004

[14] A.J. Chorin. “Numerical solution of the Navier-Stokes equations”. *Mathematics of Computation, pp. 745-762, 1968.*

[15] INRIA. “Simulation Open Framework Architecture”. 2007-2011. Web. 07 June 2014.

[16] R.W. Anderson. “An arbitrary LAgrangian-Eulerian method with adaptive mesh refinement for the solution of the Euler equations”. *Journal of Computational Physics*, pp 598-617, 2004.

[17] M.J. Berger. and A.Jameson,. “An adaptive multigrid method for the euler equations”. *Ninth International Conference on Numerical Methods in Fluid Dynamics Lecture Notes in Physics*, pp 92-97, 1985.

[18] S. Premoze, T. Tasdizen, J.Bigler, A.Lefohn and R.T. Whitaker. “Particle-Based Simulation of Fluids”. *Eurographics*, 2003.

[19] J.F. Price. “Lagrangian and Eulerian Representations of Fluid Flow: Kinematics and the Equations of Motion”. 2006.

[20] J. Stam and E. Fiume. “Depicting Fire and other Gaseous Phenomena using Diffusion Processes”. Computer Graphics, 29th Annual Conference Series, pp. 129-136, 1995.

[21] M. Müller, D. Charypar, and M.Gross. “Particle-based Fluid Simulation for Interactive Applications”.
In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 154-159. Eurographics Association, 2003.

[22] M. Desbrun and M.P. Cani. “Smoothed Particles: a new paradigm for animating highly deformable bodies”. Computer Animation and Simulation, pp. 61-76, 1996.

[23] N. Bagdassarov. “Transient phenomena in vesicular lava flows based on laboratory experiments with analogue materials”. *Journal of Volcanology and Geothermal Research*., pp 115-136. 2004.

[24] R.C Kerr, A.W. Woods, M.G. Worster and H.E. Huppert. “Solidification of an alloy cooled from above Part 1. Equilibrium growth”. *J. Fluid Mech. vol. 216*, pp 323-342. 1989.

[25] D. Enright, S. Marschner, and R. Fedkiw, “Animation and rendering of complex water surfaces,” *ACM Trans. Graph. (SIGGRAPH Proc.), vol. 21, no. 3*, pp. 736–744, 2002.

[26] M. Müller, B. Solenthaler, R. Keiser, M. Gross. “Particle-Based Fluid-Fluid Interaction”, *Eurographic/ACM SIGGRAPH Symposium on Computer Animation*. pp. 1-7. 2005.

[27] F. Losasso, T. Shinar, T. Selle and R. Fedkiw. “Multiple Interacting Liquids”, Proceeding *SIGGRAPH (2006)*, pp 812-819. 2006.

[28] A. Lamorlette and N.Foster, “Structural modeling of natural flames.” *ACM Trans. Graph*, pp. 729–735. 2002.

[29] D. Nguyen, R. Fedkiw and H. Jensen. “Physically based modeling and animation of fire.” *ACM Trans. Graph,* pp. 721–728. 2002

[30] J.B. Franzini and E.J. Finnemore. “Fluid Mechanics with Engineering Applications”, 1997.

[31] P. Hut, J. Makino, S.McMillan. “Building a better leapfrog”, Astrophysical Journal, Part 2, vol. 443, no. 2, pp. 93-96, 1995.

[32] D.R. Lide, “CRC Handbook of Chemistry and Physics (70th Edn.).” *Boca Raton (FL).* 1990.

[33] E.T. Jaynes. “The Evolution of Carnot’s Principle”. *Carnot.* 1996.

[34] F. Lambert. “A Student’s Approach to the Second Law and Entropy”. *Occidental College*. 2008. Web. 02 February 2014.

[35] L. Boltzmann. “Lectures on Gas Theory”. *Dover (reprint)*. 1995.

[36] A. Saha, S. Lahiri, A.M. Jayannavar. “Entropy production theorems and some consequences”. *The American Physical Society.* pp 1-10. 2009.

[37] J.H. Lienhard. “A Heat Transfer Textbook (3rd ed.)”. *Cambridge, Massachusetts: Phlogiston Press*. 2003.

[38] I.F. Sbalzarini. “Particle Methods for the Simulation of Diffusion Processes in Space”. 2007.

[39] M.J. Gourlay. “Fluid Simulation for Video Games (part 10)”. *Intel Developer Zone*. 2012. Web. 18 August 2013.

[40] P.L. Garrido, P.I. Hurtado, B. Nadrowski. “Simple One-Dimensional Model of Heat Conduction which Obeys Fourier’s Law”. *Physical Review Letters.* 2001.

[41] G.Wallner. “Simulating, animating and rendering clothes”.

[42] J.Ciesko. “Practical Clothes Modeling and Simulation”. *University of Erlangen-Nuremberg.* 2008.

[43] K. van Kooten, G. van den Bergen, A. Telea. “Point-Based Visualization of

Metaballs on a GPU". *GPU Gems 3*. 2007.

[44] W.E. Lorensen, H.E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". *Proceeding SIGGRAPH '87*. pp 163-169. 1987.

[45] W.J. van der Laan, S. Green and M. Sainz. "Screen Space Fluid Rendering with Curvature Flow". *Proceedings of the 2009 symposium on Interactive 3D graphics and games.* pp 91-98. 2009.

[46] M. Carlson, P.J. Mucha, R. Brooks van Horn III and G.Turk. "Melting and Flowing". *ACM SIGGRAPH*. 2002.

[47] M. Müller, S. Schirm, M. Teschnes, B. Heidelberger and M. Gross. "Interaction of Fluids with Deformable Solids". *Journal Computer Animation and Virtual Worlds*. pp 159-171. 2004.

[48] Y.A. Çengel. "Heat Transfer: a practical approach". *McGraw-Hill series in mechanical engineering. (2nd ed.). Boston: McGraw-Hill*. 2004.

[49] D. Baraff and A. Witkin. "Large steps in cloth simulation.", *Proceedings of SIGGRAPH*. pp 43-54. 1998.

[50] 2012 Hoetzlein, Rama C. Fluids v.3 – A Large-Scale, Open Source Fluid Simulator. Published online at http://fluids3.com. Released under Z-lib license.

[51] M.E. Browne. "Schaum's outline of theory and problems of physics for engineering and science". *McGraw-Hill Companies*. p. 58. 1999.

[52] Wal+, "Frozen Lake Baikal". *Panoramio..* 3 April 2011. Web. 22 Juni 2013.

[53] C.A. Pickover. "Laws of Science and the Great Minds Behind Them: Archimedes to Hawking". *Oxford University Press, Inc.* 2008.

[54] F. Losasso, F.Gibou, R.Fedkiw, "Simulating water and smoke with an octree data structure". *Proceeding SIGGRAPH '04.* Pp 457-462. 2004.