Computational Fluid Dynamics in 2D Game Environments

Kalle Sjöström

August 3, 2011
Master's Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Niclas Börlin
Examiner: Fredrik Georgsson

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

Games are becoming increasingly realistic. Real-time physics simulation were almost unimaginable just decades ago but are now a vital part of many games. Even dynamic physics simulation e.g. interactive fluids has found a place in game development.

This paper investigates and evaluates three methods of simulating fluids with the purpose of testing these in a 2D game environment. These methods are all *Lagrangian* i.e. particle-based, SPH methods, and chosen because of their differences but also their importance to the field of interactive fluid simulation. In order to integrate the methods, they will be implemented with use of the 2D mechanics engine BOX2D which is a popular choice in 2D game development.

To evaluate the methods, water is the fluid of choice. Water is the most abundant of fluids and is bound to be found in most games containing fluids. Water is almost incompressible, therefore, the methods ability to withhold incompressibility is tested. Also, the convergence properties of kinetic energy is tested in order to find out more about stability.

The results showed that the method based on Müller et al. [2003] demanded a prohibitively small time-step to be especially useful. The method from Clavet et al. [2005] managed to keep a sufficiently large time-step but failed in simulating incompressible low-viscosity fluids. However, it excelled in the simulation of highly viscous fluids like gel. Finally, the method based on Bodin et al. [2011] showed impressive result in incompressibility but is more difficult to implement and requires a bit more resources and run-time.

The paper concludes that if easy-to-implement cool effect is sought, then the method of Clavet et al. [2005] could be used with great results. However, Bodin et al. [2011] would be the best choice for games where physical accuracy is of greater importance. Also, BOX2D is a good choice to *extend* with a fluid engine. However, it will never be as good as creating a physics engine with an *integrated* fluid engine.

Contents

Acknowledgements

First and foremost, I would like to thank my mentor at Umeå University, Niclas Börlin. He has guided me through the journey of writing my first scientific paper on this scale.

Thanks, also, to all of Arrowhead Game Studios for welcoming me into the wonderful world of game-making. I would especially like to thank Johan Pilestedt for giving me this opportunity and to my mentor at Arrowhead, the one and only Anton "Tantor" Stenmark for helping me with technical issues and for being there to bounce ideas off.

Some images has been kindly lend to me by Kelager [2006], whose paper was a great source of information. Emil Ernerfeldt was kind enough to lend me a draft of his master's thesis which contained much inspirational content.

This thesis is dedicated to my parents, who has always been there for me.

Notations

Scalars

x	General scalar.
$x^{(t)}$	General scalar at time t.
$\dot{x}$	Time derivative of the scalar x, i.e. $\dfrac{dx}{dt}$.

Vectors

$\boldsymbol{x}$	General column vector.
$\boldsymbol{x}^T$	General row vector.
$\boldsymbol{x}_i$	The ith element of vector $\boldsymbol{x}$.
$\boldsymbol{x}^{(t)}$	The vector $\boldsymbol{x}$ at time t.
$\dot{\boldsymbol{x}}$	Time derivative of the vector $\boldsymbol{x}$, i.e. $\dfrac{\partial \boldsymbol{x}}{\partial t}$. In two dimensions, this becomes, $\dfrac{\partial x}{\partial t}\boldsymbol{i} + \dfrac{\partial y}{\partial t}\boldsymbol{j}$.

Special Vectors

$\boldsymbol{\lambda}$	The vector of Lagrange multipliers.
$\boldsymbol{r}$	Position vector, in this thesis $\boldsymbol{r}$ is usually a 2D vector with x and y component. The magnitude, i.e. $\lvert\boldsymbol{r}\rvert$, is written simply as r.
$\boldsymbol{r}_i$	Position vector of particle i.
$\boldsymbol{r}_{ji}$	Relative position vector between particle i and j, i.e. $\boldsymbol{r}_{ji} = \boldsymbol{r}_j - \boldsymbol{r}_i$. The magnitude, or distance between particle i and j, i.e. $\lvert\boldsymbol{r}_{ji}\rvert$ is, written simply as r_{ji}.
$\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$	Unit vectors in x, y and z direction respectively.

Matrices

$\boldsymbol{M}$	General matrix (Latin letters).
$\boldsymbol{M}^T$	Transpose of matrix $\boldsymbol{M}$.
$\boldsymbol{M}^{-1}$	Inverse of matrix $\boldsymbol{M}$.
$\boldsymbol{M}_i$	The row i and column j of matrix $\boldsymbol{M}$.
$\boldsymbol{M}_{ij}$	The element on row i and column j of matrix $\boldsymbol{M}$.
$\boldsymbol{\Sigma}$	Diagonal matrix (Greek letters).
$\mathbb{T}$	Tensor.

Functions

$f(\boldsymbol{x}) = y$	Scalar function, taking vector parameter $\boldsymbol{x}$ (possibly of length 1) and returning scalar value y.
$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}$	Vector of multiple scalar functions, each taking vector parameter $\boldsymbol{x}$ (possibly of length 1) and returning a scalar contribution to resulting vector $\boldsymbol{y}$. The i:th function $\boldsymbol{f}_i(\boldsymbol{x})$ is a scalar function.
$F(\boldsymbol{x}) = \boldsymbol{y}$	Vector field function, taking vector parameter $\boldsymbol{x}$ and returning vector value $\boldsymbol{y}$. In 2D, usually written $F(x, y) = \boldsymbol{y}$.
$\nabla f(\boldsymbol{x}) = \boldsymbol{y}$	The gradient of a scalar function. In 2D, this becomes $\frac{\partial f}{\partial x}\boldsymbol{i} + \frac{\partial f}{\partial y}\boldsymbol{j}$.

Chapter 1

Introduction

1.1 Background

Video games has been around longer than most believe. In the late forties, a few selected people enjoyed "homemade" games built with cathode ray tubes [Goldsmith Jr. and Mann, 1948], yet it was not until the early seventies before the first commercial game saw the light of day[1]. Roughly one decade later, video consoles and computers started invading peoples homes and a new medium was born. Some games of that era let people strove around in magical two dimensional worlds. However, due to limitations imposed by hardware and undeveloped methods, the worlds were often rather static, e.g. interactions, if any, were predefined.

A lot has happened since. The simulated worlds now let players freely roam the third dimension as well. Improved hardware, especially the advent of a programmable specialized Graphics Processing Unit (GPU) has lead to the separation of graphics from the rest of the software, resulting in the concept of the so-called *graphics engine* [Eberly, 2010]. This idea has been extended further into the creation of *sound engines*, *multi-player engines*, *AI engines*, and so on. Around the turn of the millennium, the *physics engine* was introduced. Games could now respond in a physically plausible way which caused the level of realism and immersion to quickly rise. Seemingly endless possibilities were suddenly available to the game developers. The wheel need not be reinvented because professionally made software with well tested and stable physics simulation were available.

There is no such thing as a *complete* physics engine and there will never be. It would not be plausible to simulate every natural phenomena exactly. Therefore, simplifications are made such as assuming that bodies are rigid. This assumption constitutes the foundation of most physics engines. As an extension to this, real-time methods for simulating *soft* bodies, e.g. cloth, ropes, balloons, as well as fluids have emerged, mostly focusing on 3D.

[1] *Computer Space*, N. Bushnell and T. Dabney, Nutting Associates - 1971

However, the two dimensional games of past times have once again gained life. This time in electronic form available on the current console generation, e.g. Wii Virtual Console[2], Xbox Live Arcade[3] and PlayStation Store[4]. With this evolution, the demand for powerful, general purpose, physics engines for *two* dimensions is increasing. Would it be possible to combine advanced features of modern physics engines with the simplicity inherit in the two dimensional gaming worlds? This thesis strives to answer this by examining how fluid dynamics together with a rigid body mechanics simulator can be used in a two dimensional game environment.

1.2 Problem Description

Arrowhead Game Studios[5] is a game company located in Skellefteå, Sweden. Arrowhead has recently released the award-winning game *Magicka*[6] whose key feature is dynamic interaction with an artificial world. In spirit of this idea, the company now has ideas about a game based on physical puzzles in a freely explorable 2D world. This world would contain interactive fluids which would be part of the puzzles. This thesis will focus on the simulation of fluids as an extension to an already existing 2D physics engine.

1.3 Goals

The goal of this project is to implement three methods of simulating fluids in a 2D game environment. The methods must be implemented using an existing 2D rigid body mechanics engine called Box2D[7]. The methods will then be compared and evaluated on how well they can simulate water and other similar fluids. In game environments, interaction with rigid bodies is key. It is therefore an important goal to evaluate the ability to interact with rigid bodies. The methods should be able to run, with as few modifications as possible, on multiple platforms including the current console generation.

1.4 Related Work

Physics engines was pioneered by the companies Havok[8] and MathEngine. Around the year 2000, both companies had working engines to show. A couple of years later, Ageia Technologies, Inc., created specialized hardware support for real-time physics called PhysX[9]. Ageia was later purchased by Nvidia Corporation and the product is now a fully fledged physics engine with heavy GPU support [Eberly, 2010].

[2] http://www.nintendo.com/wii/online/virtualconsole/
[3] http://www.xbox.com/en-US/live
[4] http://us.playstation.com/psn/playstation-store/index.htm
[5] http://arrowheadgamestudios.com/
[6] http://www.magickagame.com/
[7] http://www.box2d.org
[8] http://www.havok.com
[9] http://developer.nvidia.com/object/physx.html

During the past decade, many physics engines have been developed for many different purposes, e.g. AgX Multiphysics[10], Bullet physics[11], and Wild Magic engine[12]. The engine of most importance for this thesis is Box2d, an open source rigid body mechanics engine developed by Catto [2005, 2010].

A general and intuitive book on physics for games is Millington [2010]. This would be suitable as introductory material to the subject. A more advanced book is Eberly [2010] in which topics such as fluids and deformable bodies exist. Both these books are shipped with a software physics engine. The book by Erleben et al. [2005] is a more general book on physics-based animation, not necessarily for games.

There are many methods used to simulate fluids. Usually, either Eulerian or Lagrangian methods are used in the context of graphical applications. These are referred to as grid-based and particle-based methods respectively. This thesis focuses on particle-based fluids. One such method is *The Smoothed Particle Hydrodynamic* method (Sph) which has its roots in astrophysics [Lucy, 1977, Gingold and Monaghan, 1977]. However, the work by Müller et al. [2003] constitutes the basis for most modern interactive Sph methods. There are many tutorials on Sph, some of which are aimed at novice level [Pelfrey, 2010, Braley and Sandu, 2009] and some who take a more rigours path [Cossins, 2010]. For an introduction to grid-based methods, see e.g. Braley and Sandu [2009] and Stam [2003]. Couplings between the two also exists, e.g. Losasso et al. [2008], in which they use a grid-based method on dense liquid bodies and Sph on the diffuse parts, e.g. foam and sprays.

There are many variants on Sph focusing on different aspects of fluids. Melting and highly viscous flow are described in Carlson et al. [2002] and Iwasaki et al. [2010] where the latter concentrates solely on water and ice while Clavet et al. [2005] focuses on viscoelasticity. Furthermore, interaction between fluids with high density is dealt with in Solenthaler and Pajarola [2008].

General interaction between Sph-fluids and the environment, called *boundary conditions*, is a huge problem with many suggested solutions. The problem is twofold. First, the smoothing of the fluid values becomes erroneous near boundaries due to lack of neighbors. Second, it is non-trivial to represent the force that keeps the particles inside its container. One method dealing with this is the multiple boundary tangent method [Yildiz et al., 2008] in which fixed boundary particles are placed on the boundary. These boundary particles mirror their neighbors in the tangent to the boundary, creating *ghost particles*. Regular particles can then get these mirrored particles as neighbors. Another technique is to represents the boundaries (and other rigid bodies) as particles. This will simplify collision detection and handling but can potentially increase the number of simulated particles drastically. Kurose and Takahashi [2009] resolve the collisions between fluid and the particle-based boundaries by introducing constraints on their relative velocity. However, the rigid bodies must be unconstrained; in other words, no jointed bodies would be allowed. The work by Carlson et al. [2004] tries to bridge a rigid body solver with a grid-based fluid solver. A recent article Bodin et al. [2011] describes a novel approach to Sph in which each particle's density is constrained to the reference density of the fluid. One can then formulate non-penetration constraints between the fluid particles and the boundary and solve the complete system

[10] AgX Multiphysics is a 3d multi-physics engine for professional and industrial applications, http://www.algoryx.se/agx

[11] Bullet physics is a general, open source, 3d physics engine for games, http://bulletphysics.org

[12] Wild Magic engine is a physics engine for educational purposes, [Eberly, 2010]

simultaneously. They also suggests a novel boundary technique where density is added to particles near boundaries. This causes the constraint solver to try and remedy the increased density by applying forces in the direction from the boundary to the particle.

Density constrained fluids is also covered in Nilsson [2009] which implements it on Gpu using Cuda, see also Lacoursière et al. [2010] where granular matter, rigid bodies and fluids are simulated all with the same solver. For more Gpu-based Sph methods, see Harada et al. [2007], Grahn [2008] and Iwasaki et al. [2010].

For a survey on fluids, see Tan and Yang [2009] and for general deformable bodies, including fluids, see Nealen et al. [2006].

1.5 Related Software

There are many physics engines available focusing on a variety of features. There are several open source engines dealing with general 3d physics such as Open Dynamics Engine[13], Bullet[14][Coumans, 2010], Newton Game Dynamics[15]. There are also proprietary ones in the same category such as Havok[16] and Nvidia PhysX[17]. For open source 2d engines focusing on rigid body mechanics, see Box2d[18] [Catto, 2010] and Chipmunk[19], however, no 2d physics engine dealing with general physics was found.

There are a couple of 2d games which implements interactive fluids, but they are still few. As early as 1991, a game called Cataclysm[20] was released where the player manipulate the flow of water in order to solve puzzles. A more modern game is PixelJunk Shooter[21] on PlayStation 3 in which the player manipulates fluids, e.g. pouring water on magma so that it turns into stone. In Vessel[22] the player can spray highly viscoelastic fluids.

General 2d "sandbox" programs also exists in which the user can play around in a physical world. A general such sandbox is Algodoo (previously Phun [Ernerfeldt, 2009]) focusing on physical accuracy and multibody physics including fluids, optics and electricity [Dahlberg, 2011]. OE-Cake[23] on the other hand focuses on the state of matter. Here, the user can melt rigid object and create fluids with different viscosity and other properties. A similar sandbox is called The Powder Toy[24].

[13] http://www.ode.org/
[14] http://bulletphysics.org/
[15] http://newtondynamics.com/
[16] http://www.havok.com/
[17] http://www.nvidia.com/object/physx_new.html
[18] http://www.box2d.org/
[19] http://code.google.com/p/chipmunk-physics/
[20] Cataclysm is made by *4th Dimension* in 1991
[21] PixelJunk Shooter is made by *Q-Games* in 2009
[22] Vessel is made by *Strange Loop Games* in 2011
[23] http://www.octaveengine.com/
[24] http://powdertoy.co.uk/

1.6 Organization of this Thesis

A short introduction of physics for games will be covered in Chapter 2. It will address the key concepts needed to understand the text to come. If the reader feels comfortable with multivariate calculus, multibody physics simulation, iterative methods and so on, feel free to skip all or part of Chapter 2.

When the background theory of physics is covered, the paper steers towards its key concept, fluid mechanics. In Chapter 3, the background theory of fluid mechanics, as well as the field of computational fluid dynamics, is presented.

Chapter 4 presents the methods that the implementation is build upon and also how these methods are modified to fit the intended purpose, real-time interactive simulation in a game environment. The implementation of the methods and the coupling with Box2d is covered in Chapter 5. The experiments that should be run to test the different methods are listed and explained in Chapter 6. The result of these experiments are presented in Chapter 7 followed by a discussion in Chapter 8.

The thesis ends with a conclusion in Chapter 9 which will make some final points about the methods. Also, some ideas for future work is presented here.

Chapter 2

Physics for Games

This chapter gives some background information concerning the physics and math used in two dimensional physics simulations. It will begin by describing how some physical formulae and quantities used in 3D applications are mapped to their two dimensional counterparts, in Section 2.1. *Multivariate calculus* is important for fluid simulation and therefore follows, in Section 2.2.

Section 2.3 discusses how time is discretized and how objects are moved over time, i.e. how acceleration and velocity is *numerically integrated* to give position. The chapter goes on by introducing the concept of *constrained mechanical systems* in Section 2.4. These constraints are often formulated as system of linear equation in matrix form. One way to formulate a constraint mechanical system in such a way, is the *spook method* [Lacoursière, 2007], described in Section 2.5. Section 2.6 discusses how to solve such formulations using general iterative methods.

How to handle collisions is the final topic of this chapter and is described in Section 2.7.

2.1 Physics in Two Dimensions

The physics in two dimensions are obviously very similar to the physics in three dimensions. However, many simplifications can be made, especially concerning rotations. The information in Section 2.1.1 is retrieved from Bedford et al. [2008] while the main sources of Section 2.1.2 is Servin [2010] and Millington [2010] unless otherwise stated.

2.1.1 Moment of Inertia

Mass is way of measuring the resistance to linear movement. A greater force is required to move an object of greater mass which is easily seen through, the well known, Newton's second law, $f = ma$.

In the same way, *moment of inertia* is a way of measuring the resistance to *angular* movement, or resistance to rotate. This is calculated from the distribution of mass over the volume (or area) of the object and is measured in $kg \cdot m^2$. It will, in other words, depend on the size, shape and density of the object. The moment of inertia (I) is defined as the following integral over the volume of a body

$$I = \int_V \rho(\boldsymbol{r}) d(\boldsymbol{r})^2 dV(\boldsymbol{r}), \tag{2.1}$$

where $\boldsymbol{r}$ is a point in the volume, $\rho(\boldsymbol{r})$ is the mass density at that point, $d(\boldsymbol{r})$ is the distance from the point to the axis of rotation and lastly dV is the infinitesimal volume element parametrized by $\boldsymbol{r}$. Note that if the axis of rotation is the origin from which the vector $\boldsymbol{r}$ is defined, $d(\boldsymbol{r})$ is just the length of $\boldsymbol{r}$. Also note that if the density distribution is constant, it can be factored out of the integral. Assuming both these previous notes the integral collapses to

$$I = \rho \int_V ||\boldsymbol{r}||^2 \, dV(\boldsymbol{r}) = \rho \int_V \boldsymbol{r}^T \boldsymbol{r} dV(\boldsymbol{r}). \tag{2.2}$$

The analog between moment of inertia and mass[1] can also be seen when comparing Newton's second law with its rotational counterpart, as is done below.

$$\begin{aligned} f &= ma, \\ \tau &= I\alpha, \end{aligned} \tag{2.3}$$

where

- f is force and τ is torque,
- m is mass and I is moment of inertia and
- a is acceleration and α is angular acceleration.

When the object is rotating around an arbitrary axis, the scalar moment of inertia cannot be used in 3D. Therefore, a symmetric positive semi-definite 3 by 3 matrix is used to represent the moment of inertia which is referred to as the *moment of inertia tensor*. This, however, is of no concern in 2D physics because the object must rotate around an axis parallel to the z-axis. Therefore, the scalar moment of inertia can still be used. This is a great simplification and improves both memory usage and computations.

When the freedom of choosing an arbitrary axis of rotation is removed, the parallel axis theorem[2] can be used to calculate the corresponding moment of inertia around a different, but still parallel, axis of rotation.

[1] Mass is, in fact, sometimes referred to as *linear inertia*.
[2] The parallel axis theorem is also called *Huygens-Steiner theorem*

More formally, the theorem states that if the moment of inertia around a certain axis I_a is known, then the moment of inertia around a different axis I_b parallel to I_a is

$$I^b = I^a + mr^2, \tag{2.4}$$

where m is the mass of the object and r is the shortest distance between the axes.

2.1.2 Cross-product

The cross-product is not unambiguously defined in two dimensions. Two different approaches can be taken here depending on what the purpose of the cross-product is, in a given case. One purposed could be to get an orthogonal (or perpendicular) vector, in which case one could simply turn the given vector 90° clockwise[3].

$$(a_y, -a_x) = Cross_{perp}(\boldsymbol{a}). \tag{2.5}$$

Another purpose could be to measure a quantity such as the angular velocity. In this case, the magnitude of the cross-product in 3D is still the same as the magnitude of the sought velocity and can therefore be used. The direction of the axis can be ignored as it is already known to be parallel to the z-axis. Taking this approach, one can simply take the cross product of the two vectors and setting the z component to zero. Using Cofactor expansion

$$a \times b = \begin{vmatrix} a_y & 0 \\ b_y & 0 \end{vmatrix} \boldsymbol{i} - \begin{vmatrix} a_x & 0 \\ b_x & 0 \end{vmatrix} \boldsymbol{j} + \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix} \boldsymbol{k} = (a_x b_y - a_y b_x)\boldsymbol{k}. \tag{2.6}$$

Therefore, the definition of the magnitude of the cross product in 2D becomes

$$a_x b_y - a_y b_x = Cross_{magnitude}(\boldsymbol{a}, \boldsymbol{b}). \tag{2.7}$$

2.2 Multivariate Calculus

A very brief introduction to some relevant subjects in multivariate calculus is presented in this section. This is by no means complete and interested readers are referred to Adams [2006] which is the source of this section, if not otherwise stated.

[3]Only convention dictate that is should be turned *clockwise*. There is no physical meaning about that choice.

2.2.1 Vector Calculus

Scalar Field

A *scalar field* is a function which takes a point in space and returns a *scalar*. For example, in a 2D case this would be

$$a = f(x, y), \tag{2.8}$$

where x and y represents a point in 2D space and a is the returned scalar. This could be the temperature at the point (x, y). The rate of change of a scalar field is contained in the n first partial derivatives, where n is the number of dimensions of the function (in this case $n = 2$). This is compactly described using the *gradient* of f. The gradient of a function of two variables is shown below. The gradient is defined in the same way for functions of more than two variables, i.e.

$$\nabla f(x, y) = \frac{\partial f}{\partial x}\boldsymbol{i} + \frac{\partial f}{\partial y}\boldsymbol{j}, \tag{2.9}$$

where $\boldsymbol{i}$ and $\boldsymbol{j}$ are the unit base vectors spanning the two dimensional space.

This function points in the direction (in the plane) where to move if one wants to maximize a. If a is temperature, then the gradient points in the direction of highest temperature (locally).

Vector Field

A *vector field* is a function which takes a point in space and returns a *vector*. For example, in a 2D case this could be

$$\boldsymbol{a} = F(x, y) = F_1(x, y)\boldsymbol{i} + F_2(x, y)\boldsymbol{j}, \tag{2.10}$$

where x and y represents a point in 2D space, $\boldsymbol{a}$ is the returned vector and $F_1(x, y)$ as well as $F_2(x, y)$ are scalar functions which gives the magnitude of the $\boldsymbol{i}$ and $\boldsymbol{j}$ component of $\boldsymbol{a}$. The resulting vector $\boldsymbol{a}$ could be the flow velocity of a fluid at the point (x, y).

The rate of change of this field is contained in the n^2 first partial derivatives, where n is the number of dimensions of the function (in this case $n = 2$). This generally cannot be as compactly written as the gradient of a scalar field where ∇ is simply multiplied by a scalar function ($\nabla f(\boldsymbol{x})$). In the same way as multiplication is extended into the dot and cross product, the gradient is extended to two operations, the *divergence* and *curl*. These are written as

$$\begin{aligned} \text{Divergence:} \quad \operatorname{div} F &= \nabla \cdot F = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} \\ \text{Curl:} \quad \operatorname{curl} F &= \nabla \times F = \frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y} \end{aligned} \tag{2.11}$$

Note that the cross product is not defined unambiguously in two dimensions, see Section 2.1.2. The curl in two dimensions is the magnitude of the three dimensional cross-product when the third dimension is set to zero. For the curl in three dimensions, the full cross-product is used.

$$\nabla \times F(x, y, z) = \left(\frac{\partial F_3}{\partial x} - \frac{\partial F_2}{\partial y}\right) \boldsymbol{i} + \left(\frac{\partial F_1}{\partial x} - \frac{\partial F_3}{\partial y}\right) \boldsymbol{j} + \left(\frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}\right) \boldsymbol{k}. \tag{2.12}$$

Also note that the divergence of F is always a scalar field but the curl is a scalar field *only* in two dimensions. In three dimensions, the curl is another vector field. This is very intuitive if one is familiar with the properties of the dot and the cross product.

The divergence can be described as the rate at which the vector field *diverges* or *spreads away* from a given point. It measures the quantity of *flux* emanating from a given point in the vector field[4]. Flux is the process of flowing (especially outwards), or the rate of flow. If there is a flux source at point P and no sinks, then the divergence at that point is equal to the *source strength* according to the *Divergence Theorem*. This means, that if there is *not* a source at point P, nor a sink, then the quantity of flux going "into" P is the same as the quantity "emanating from" P and the divergence at P is therefore zero.

This can be used in the context of fluids. For example, if a quantity (such as mass) should be conserved, then the inward flux must equal the outward flux i.e. the divergence at all points (except sources and sinks, if any) must be zero.

Substantial Derivative

The position of ∇ in the definition of divergence is very important. It can be placed *after* F resulting in

$$F \cdot \nabla = F_1 \frac{\partial}{\partial x} + F_2 \frac{\partial}{\partial y}. \tag{2.13}$$

The divergence is an operator that operates on some input. $F \cdot \nabla$ does not change that fact i.e. it is also an operator. The only difference is that the divergence of the input is scaled by the components of F. This new operator is called the *scalar differential operator* and is used in the construction of the *substantial operator* used to define the *substantial derivative* or the *material derivative*.

[4]More specifically it is the limit of the flux per unit volume (or area in 2D) out of the surface of an infinitesimally small sphere (or circle in 2D) which is centered at the given point.

Consider the scalar field of the temperature of a moving particle in a fluid, $a_{\text{temp}} = T(x, y)$. Also assume that the position of the particle is changing over time and that the temperature also changes over time even though the particle might be still. Then the temperature at time t is $T(x(t), y(t), t)$. The rate of change in temperature is given by the total derivative of T with respect to time

$$\begin{aligned} \frac{dT}{dt} &= \frac{\partial T}{\partial t} + \frac{\partial T}{\partial x}\frac{dx}{dt} + \frac{\partial T}{\partial y}\frac{dy}{dt} \\ &= \frac{\partial T}{\partial t} + v_1\frac{\partial T}{\partial x} + v_2\frac{\partial T}{\partial y}, \end{aligned} \tag{2.14}$$

where v_1 and v_2 are the linearly independent components of the velocity vector field $\boldsymbol{v}$.

The substantial derivative is a way of writing this in a more compact and dimensionless way. Using Eq. 2.13 one can rewrite Eq. 2.14 as

$$\frac{\partial T}{\partial t} + v_1\frac{\partial T}{\partial x} + v_2\frac{\partial T}{\partial y} = \frac{\partial T}{\partial t} + \boldsymbol{v} \cdot (\nabla \cdot T). \tag{2.15}$$

Equation 2.15 can be written as an operator times T, where the operator would be

$$\frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla. \tag{2.16}$$

The *Substantial Derivative* (or *Material Derivative*) can now be defined as the operator

$$\frac{D}{Dt} \stackrel{\text{def}}{=} \frac{\partial}{\partial t} + \boldsymbol{v} \cdot \nabla \tag{2.17}$$

where $\boldsymbol{v}$ is the velocity vector field. In this equation, the first term (on the right) represent change over time, or *local derivative*, whereas the second term represent changes respect to position, or *convective derivative*.

2.3 Numeric Integration

The flow of time, as we believe, is continuous. There is no time-unit which cannot be subdivided in smaller parts, or steps, i.e. the time-unit is infinitesimal (infinitely small). Obviously, computers cannot handle infinitely many updates per second so when simulating physics with computers, time must be discretized.

Most real-time games have an updated frequency of 60 Hz but in some cases this is not enough for the underlying physics. The physics might need to be updated at 120 Hz. The choice is context dependant, but higher frequency results in a closer approximation of infinitesimal time-steps which might be believed to give a more accurate simulation. In

many cases this is true, but it is also very important to look at *how* the objects are moved through space over time.

Numeric integration has been subject to much research where stability and accuracy has been thoroughly analyzed. Here, only a brief introduction and an intuitive derivation of some common options are described. For a more in-depth study of numeric integration in computational physics, see e.g. Erleben et al. [2005], Lacoursière [2007].

The teachings of fundamental physics tells that the acceleration of an object is given by $\boldsymbol{a} = m^{-1}\boldsymbol{f}$, and that this can be integrated to give velocity $\boldsymbol{v}$. Furthermore, the velocity can be integrated to give the position $\boldsymbol{x}$. Assuming that mass is constant, this yields

$$\boldsymbol{v} = \int_0^t \boldsymbol{a}(t)dt, \tag{2.18}$$

$$\boldsymbol{x} = \int_0^t \boldsymbol{v}(t)dt. \tag{2.19}$$

An integral is the area under the curve of consideration which can be approximated using a discreet number of rectangles, the Riemann sum, see Fig. 2.1. As the width of the rectangles approaches zero, the sum of all areas will approach the true integral.

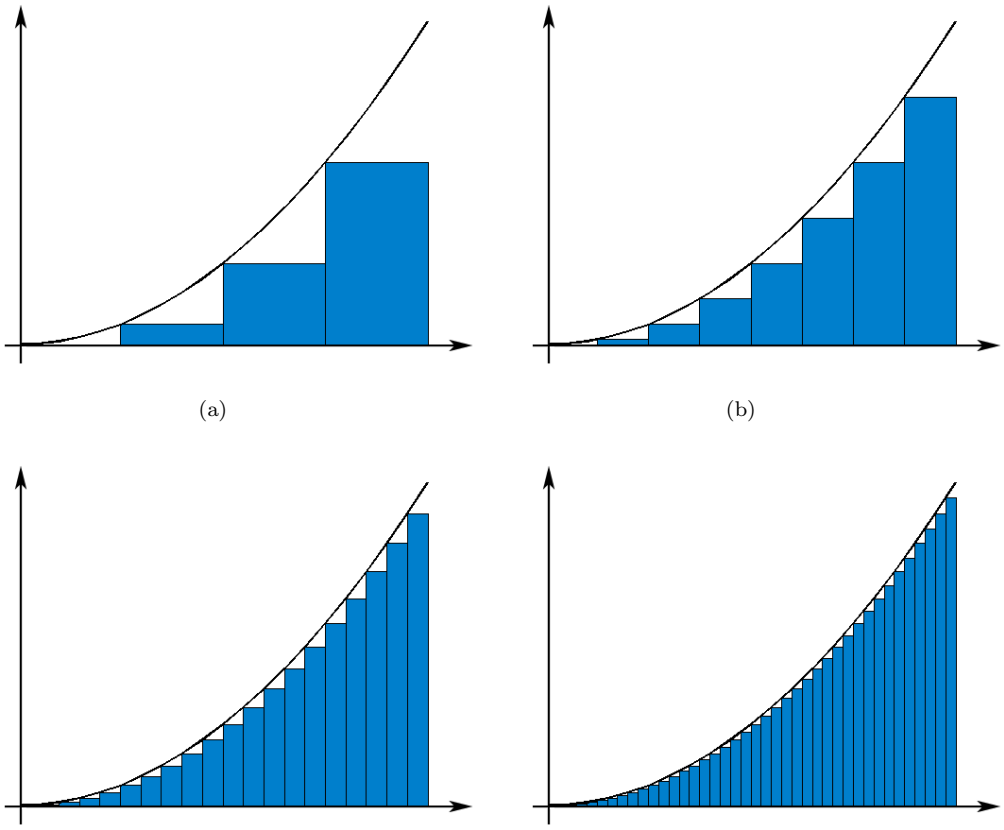
The velocity integral is the area under the curve $\boldsymbol{a}(t)$. In this case, the time-step is the width of the rectangles and $\boldsymbol{a}(t)$ is the height at time t, thus giving the area $\Delta t \cdot \boldsymbol{a}(t)$. The velocity at a given time is the sum of all previous areas plus the current. The same principle can be applied to position, giving

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{v}^{(t)} + \boldsymbol{a}^{(t)}\Delta t, \tag{2.20}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \boldsymbol{v}^{(t)}\Delta t. \tag{2.21}$$

There are multiple ways of choosing the "height" of the rectangles, i.e. where the rectangles should intersect the curve, see Fig. 2.2. This choice will directly affect the stability and accuracy of the method. The approximated area in Fig. 2.2c are larger than 2.2a so the different choices obviously give different results. The method of Left Riemann sum, Fig. 2.2a, will underestimate monotonically increasing functions while overestimate monotonically decreasing functions. The right sum in Fig. 2.2c will do the opposite. The middle sum will not systematically over or underestimate.

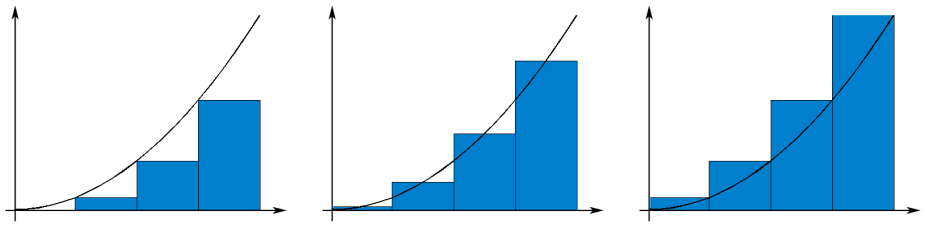
The velocity update in Eq. 2.20 would correspond to a left sum because it only uses the acceleration at the current time.



(a) (b)

(c) (d)

Figure 2.1: *The integral, or area under a curve can be approximated by summing the area of rectangles placed under the curve according to some scheme. The approximation becomes more accurate when using several smaller rectangles (b) versus using few large ones (a).*



(a) Left sum (b) Mid sum (c) Right sum

Figure 2.2: *Three different ways of placing rectangles whose area should approximate the integral of the curve.*

The question of which point, or height, to choose, can more formally be stated as choosing parameters in a linear interpolation scheme between time t and $t+1$. This can be written as

$$\begin{aligned} \boldsymbol{a}^{new} &= \alpha \boldsymbol{a}^{(t)} + (1-\alpha)\boldsymbol{a}^{(t+1)}, \quad 0 \leq \alpha \leq 1, \\ \boldsymbol{v}^{new} &= \beta \boldsymbol{v}^{(t)} + (1-\beta)\boldsymbol{v}^{(t+1)}, \quad 0 \leq \beta \leq 1. \end{aligned} \tag{2.22}$$

With this, the integration step can be rewritten as

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{v}^{(t)} + \boldsymbol{a}^{new}\Delta t, \tag{2.23}$$
$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \boldsymbol{v}^{new}\Delta t. \tag{2.24}$$

Using $\alpha = \beta = 1$ gives the *Explicit Euler* (or forward Euler) method, Eq. 2.20 and 2.21. This method is known to be unstable because errors in both position and velocity accumulate over time, [Erleben et al., 2005].

A choice of $\alpha = \beta = 0$ results in *Implicit Euler* (or backward Euler) with stepping scheme

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{v}^{(t)} + \boldsymbol{a}^{(t+1)}\Delta t, \tag{2.25}$$
$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \boldsymbol{v}^{(t+1)}\Delta t. \tag{2.26}$$

This choice uses only future values, i.e. values at the next time-step. However, the acceleration often depends on velocity (and maybe position), i.e. $\boldsymbol{a}^{(t+1)} = \boldsymbol{a}(\boldsymbol{v}^{(t+1)})$. This means that in order to calculate $\boldsymbol{v}^{(t+1)}$, the answer must be known in advance. This is why it is referred to as implicit, and can be troublesome to solve. However, with numerical methods, the solution can be approximated. Implicit Euler has good numerical stability but is dissipative, i.e. it causes the total energy to decrease. This might be fine in some cases, because real world systems are dissipative. For instance, in a mechanical system energy is often "lost" as heat due to friction. However, this numerical damping might become too big if large time-steps are used [Lacoursière, 2010, Servin, 2010].

A third choice is $\alpha = 1, \beta = 0$ which is referred to as *Semi-implicit Euler*, also called symplectic Euler, semi-explicit Euler, Euler-Cromer, and Newton-Størmer-Verlet (NSV)[5]. This method gives equivalent results to Leap-Frog and Velocity Verlet, but they are computed in a slightly different way [Servin, 2010]. This is a common method due to its computational simplicity and good stability properties. However, it is not unconditionally stable, the time-step must be kept below $1/\omega$, i.e. the time-step must be kept less than the highest frequency of oscillation inherit in the system.

[5] `http://en.wikipedia.org/wiki/Semi-implicit_Euler_method`

2.4 Constraints

Constraints are of great importance in the simulation of physics. They contain information and "rules" about where an object can be (or how fast it can move). A constraint can, for example, make sure that two objects do not penetrate each other. In some sense, constraints approximates all the forces between molecular structures that keep an object together and other objects apart.

This section begins with a mathematical description of how a function can be extremized subject to a constraint. This method, called *The method of Lagrange Multipliers*, is not limited to physics nor geometric constraints, but used in general optimization problems. The information found in this section relies heavily on Adams [2006].

The discussion is then steered back on the subject of simulated mechanical systems, introducing *holonomic* and *non-holonomic* constraints in Section 2.4.2 and Section 2.4.3 respectively. Finally, a short section on how they are integrated over time.

2.4.1 Lagrange Multipliers

In mathematical optimization problems, there exists a method called *Lagrange Multipliers*. This is useful for calculating an extreme point of a function subject to a constraint. For instance

$$\begin{aligned} \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) = c. \end{aligned} \tag{2.27}$$

The constraint g, could also be an inequality constraint, i.e. $g(x,y) \geq c$. However, here only equality constraints are considered. In the above case, f and g are three dimensional functions. These can be projected down to a two dimensional plane, showing the contour of the three dimensional structure, much like a map shows the contour of a hill. The functions above generally shows different contours which could intersect. However, if the tangent to the contours of f and g are parallel, then following the contours of g will not change the value of f, i.e. walking at a "contour line" of a hill, the traveler's height will remain the same. If the tangents are parallel, then the *gradient* of the functions are also parallel. This means that one gradient is a scalar multiple of the other, i.e.

$$\nabla f = -\lambda \nabla g, \tag{2.28}$$

where λ is some scalar. This multiplier is called the Lagrange multiplier. With it, the Lagrangian function could be defined as

$$\Lambda(x,y,\lambda) = f(x,y) + \lambda(g(x,y) - c). \tag{2.29}$$

When trying to find extreme points of a function, the derivative can be used. This because points where the derivative is zero (critical points) represent a local extreme point, e.g. maximum or minimum. In the same manner, the points of interest is where the gradient of $\Lambda(x, y, \lambda)$ is zero. Therefore, the equation

$$\nabla\Lambda(x, y, \lambda) = 0, \tag{2.30}$$

should be solved. Note that this equation also implies that $g(x, y) = c$.

One might wonder if f and the constraint function g (or functions) always share at least one common parallel tangent. The answer is no. The functions must be smooth in the neighborhood of the extreme values. Also, the extreme values must not occur on endpoints, or edges, of the domain of f and g. This method can therefore not guarantee that a solution exists, it can only provide means of finding a solution already known to exist [Adams, 2006].

It is possible to define the Lagrangian function for a constrained problem in dimension d, i.e. given the constrained problem

$$\begin{aligned} &\text{extremize} && f(\boldsymbol{x}) \\ &\text{subject to} && g(\boldsymbol{x}) = c, \end{aligned} \tag{2.31}$$

where $\boldsymbol{x} \in \mathbb{R}^d$, the Lagrangian function is

$$\Lambda(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda(g(\boldsymbol{x}) - c). \tag{2.32}$$

Another possibility is to have multiple constraints on the same function. In such case

$$\begin{aligned} &\text{extremize} && f(\boldsymbol{x}) \\ &\text{subject to} && g(\boldsymbol{x}) = c \text{ and} \\ & && h(\boldsymbol{x}) = a. \end{aligned} \tag{2.33}$$

Then $\nabla f(\boldsymbol{x}) = -\lambda\nabla g(\boldsymbol{x}) - \mu\nabla h(\boldsymbol{x})$ and the Lagrangian function is

$$\Lambda(\boldsymbol{x}, \lambda, \mu) = f(\boldsymbol{x}) + \lambda(g(\boldsymbol{x}) - c) + \mu(h(\boldsymbol{x}) - a), \tag{2.34}$$

which could be further generalized.

2.4.2 Holonomic Constraints

When only the position is constrained, e.g. "object a must be d distance units from the origin", the constraint is called *holonomic*. In general, the motion of an object under a

holonomic constraint is limited to a hypersurface, or constraint surface, of valid positions. This constraint surface is defined as

$$\boldsymbol{g}(\boldsymbol{x}) = 0, \tag{2.35}$$

where $\boldsymbol{x}$ is the position of the constrained object and $\boldsymbol{g}$ is one or more constraint functions, i.e. $\boldsymbol{g} = [g_1, g_2, ..., g_n]^T$.

Taking the time derivative of this function gives the valid velocities or the *kinematic constraint*, i.e. the velocities that make sure that the position of the object satisfies the constraint. The time derivative, using the chain rule, is

$$\mathbf{0} = \dot{\boldsymbol{g}}(\boldsymbol{x}) = \frac{\partial(\boldsymbol{g})}{\partial(\boldsymbol{x}^T)}\frac{d\boldsymbol{x}}{dt} = \boldsymbol{G}\boldsymbol{v}, \tag{2.36}$$

where $\boldsymbol{G}$ is the Jacobian matrix[6].

The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function. In fact, row i in this matrix is the gradient of $\boldsymbol{g}_i(\boldsymbol{x})$. The gradient of a function f represents the steepest slope of the tangent plane at a given point. The Jacobian, therefore, describes the orientation of the tangent plane at a given point. If body i is "outside" the constraint surface, the quickest way to return to $\boldsymbol{g}_i(\boldsymbol{x}) = 0$ is by following the negative gradient which always gives the direction of fastest decrease of a function.

This is a regular strategy when trying to minimize (or maximize) a given function. Methods such as the *steepest descent*, sometimes referred to as *gradient descent*, arise in all sorts of applications [Wang, 2008].

To concretize, consider a two dimensional space where $g(\boldsymbol{x}) = f(\boldsymbol{x}) = y$, i.e. *the body under this constraint is only allowed to move on the horizontal x-axis that goes through the origin.* If the body is at position (4, 3) the constraint violation would be $g(\boldsymbol{x}) = 3$. The direction of steepest descent is in the normal direction of the line $y = 0$. Actually the direction of steepest descent is always in normal direction to the constraint surface and is given by $\boldsymbol{G}^T$ in the general case.

This means that to keep the position of the object on the constraint surface, a force should be applied in the normal direction (or orthogonal) to the constraint surface. The magnitude of the force should be just enough to move it back. From this moment on this magnitude is labeled λ. The following statement summarizes it all. To move a body back to a non-violating position, a force $\boldsymbol{f} = \boldsymbol{G}^T\lambda$ should be applied.

Due to the fact that the constraint force is orthogonal to the constraint surface, it does no physical work which, in turn, means that it neither adds nor removes energy.

[6]Sometimes J is used for the Jacobian matrix.

2.4.3 Non-Holonomic Constraints

There are also *non-holonomic* constraints. They are constraints on the velocity of the object, (if the constraint cannot be integrated to a holonomic constraint), i.e.

$$0 = g(\boldsymbol{v}). \tag{2.37}$$

These are, for example, used to model frictional contacts whose magnitude depend on the velocity and therefore cannot be integrated to a holonomic constraint.

2.4.4 Time-integration with Constraints

Newton's second law is the most useful law when simulating the motion of a body. It states that the force acting on a body is equal to its mass times its acceleration,

$$\boldsymbol{f} = m\boldsymbol{a} = m\dot{\boldsymbol{v}}. \tag{2.38}$$

In the case of constrained mechanical systems (with multiple bodies), the constraint force is added to this equation which becomes

$$\boldsymbol{M}\dot{\boldsymbol{v}} = \boldsymbol{f} + \boldsymbol{G}^T\boldsymbol{\lambda}, \tag{2.39}$$

where $\boldsymbol{M}$ is the matrix of body masses, $\boldsymbol{G}$ is the Jacobian matrix and $\boldsymbol{\lambda}$ is the vector of *Lagrange multipliers*.

2.5 Spook

Spook is a numerical integrator for constraint systems [Lacoursière, 2007]. It is used to find the Lagrange multipliers, or rather a perturbation of them. Important to note is that the perturbations actually has physical origin as it relaxes the unphysical rigidity assumption. It does this by adding a regularization to the matrix diagonal which makes it easier to solve for iterative solvers. It also contain dissipation parameters which improves stabilization of the constraints.

Spook is usually written on the form,

$$(\boldsymbol{G}\boldsymbol{M}^{-1}\boldsymbol{G}^T + \boldsymbol{\Sigma})\boldsymbol{\lambda} = -\frac{4}{\Delta t}\boldsymbol{\Xi}\boldsymbol{g} + (\boldsymbol{\Xi} - \boldsymbol{I})\boldsymbol{G}\boldsymbol{v} - \boldsymbol{G}\boldsymbol{M}^{-1}\Delta t\boldsymbol{f}, \tag{2.40}$$

called the Schur complement form where the matrix on the left hand side $S_\epsilon = \boldsymbol{G}\boldsymbol{M}^{-1}\boldsymbol{G}^T + \boldsymbol{\Sigma}$ is called the Schur complement matrix. Here, $\boldsymbol{G}$ is the Jacobian, $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{\Sigma}$

is the diagonal regularization matrix, $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers, $\boldsymbol{\Xi}$ is a diagonal matrix which increase stability, $\boldsymbol{g}$ is the vector of constraint violations, $\boldsymbol{v}$ is the vector of body velocities and $\boldsymbol{f}$ is the vector of body forces.

For particles, the mass matrix $\boldsymbol{M}$ is strictly diagonal. Additionally, if uniform mass can be assumed, the mass matrix can be reduced to a scalar m. There are also damping and regularization parameters, τ and ϵ, which can be set to uniform values. Also, by defining the auxiliary scalar,

$$\xi = \frac{1}{1 + 4\frac{\tau}{\Delta t}}. \tag{2.41}$$

one can define $\boldsymbol{\Xi}$ and $\boldsymbol{\Sigma}$ as

$$\begin{aligned} \boldsymbol{\Xi} &= \xi \boldsymbol{I} \\ \boldsymbol{\Sigma} &= \frac{4\epsilon}{\Delta t^2 \xi} \boldsymbol{I} \end{aligned} \tag{2.42}$$

That is, they can be written as a scalar times the identity matrix. With these notes in mind, Eq. 2.40 can be rewritten. Start by writing Eq. 2.40 explicitly,

$$\left(\frac{1}{m} \boldsymbol{G}\boldsymbol{G}^T + \frac{4\epsilon}{\Delta t^2 \left(1 + 4\frac{\tau}{\Delta t}\right)} \boldsymbol{I} \right) \boldsymbol{\lambda} =$$
$$-\frac{4}{\Delta t \left(1 + 4\frac{\tau}{\Delta t}\right)} \boldsymbol{g} + \left(\frac{1}{\left(1 + 4\frac{\tau}{\Delta t}\right)} - 1 \right) \boldsymbol{G}\boldsymbol{v} - \frac{\Delta t}{m} \boldsymbol{G}\boldsymbol{f}.$$

Introducing several auxiliary constant parameters

$$d = \frac{\tau}{\Delta t}, \tag{2.43}$$
$$k = \frac{1}{\epsilon}, \tag{2.44}$$
$$a = \frac{4}{\Delta t(1 + 4d)}, \tag{2.45}$$
$$b = \frac{4d}{1 + 4d}, \tag{2.46}$$
$$e = \frac{4}{\Delta t^2 k(1 + 4d)}. \tag{2.47}$$

The constant k can now be seen as the spring constant and the relaxation d as the number of time-steps needed to remedy the constraint violation. Also note that

$$\begin{aligned}\frac{1}{1+4d}-1 &= \frac{1}{1+4d}-\frac{1+4d}{1+4d} = \\ \frac{1-(1+4d)}{1+4d} &= \\ -\frac{4d}{1+4d} &= -b.\end{aligned} \tag{2.48}$$

With these new parameters, a slightly less general form of Eq. 2.40 can be formulated as

$$\left(\frac{1}{m}\boldsymbol{G}\boldsymbol{G}^T + e\boldsymbol{I}\right)\boldsymbol{\lambda} = -a\boldsymbol{g} - b\boldsymbol{G}\boldsymbol{v} - \frac{\Delta t}{m}\boldsymbol{G}\boldsymbol{f}, \tag{2.49}$$

where $m^{-1}\boldsymbol{G}^T\boldsymbol{\lambda}$ is the sought impulse, i.e. the instantaneous velocity change. It therefore has units of velocity. Because of this, $m^{-1}\boldsymbol{G}\boldsymbol{G}^T\boldsymbol{\lambda}$ *also* has units of velocity. Clearly, the three terms on the right hand side also have this dimension and units and can therefore be seen as contributions to the constraint impulse.

The resulting constraint impulse must counteract the three terms on the right hand side. The term $a\boldsymbol{g}$ says that it must counteract the constraint violation due to *position*. This term is zero only when the body is *on* the constraint surface. The second term $b\boldsymbol{G}\boldsymbol{v}$ is the violation due to velocity, specifically the normal component of velocity. Finally $\frac{\Delta t}{m}\boldsymbol{G}\boldsymbol{f}$ is the violation due to the normal component of force. The constraint impulse must be large enough to counteract position violation, normal velocities and normal forces.

When $\boldsymbol{\lambda}$ is found, the velocities of the next time-step can be calculated as

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{v}^{(t)} + m^{-1}\boldsymbol{G}^T\boldsymbol{\lambda}^{(t)} + \frac{1}{\Delta t}m^{-1}\boldsymbol{f}^{(t)}. \tag{2.50}$$

2.6 Solvers

To solve a system of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.51}$$

one can either use direct or iterative methods. The direct methods try to solve the matrix equation exactly (within machine precision) usually using factorizations of $\boldsymbol{A}$, e.g. Qr or Lu [Strang, 1986]. However, when the matrices becomes large and sparse, the direct methods might not be fast enough for a given application. In some cases, direct methods do still compete in speed and wins in accuracy and absolutely have a place in real-time physic simulation [Lacoursière et al., 2010]. For a good direct solver, see e.g. Li [2005].

Usually though, developers turn to the iterative methods for approximations to the solution which are deemed "good enough". This is especially true for developers of real-time physics engines for games where speed is of outmost importance and the number of objects simulated are quite large.

The information in this section is retrieved from Strang [1986] and the video lecture by Strang [2006], unless otherwise stated.

2.6.1 Iterative Methods

There are many iterative methods proposed over the years. Iterative methods usually chooses a *preconditioner* $\boldsymbol{P}$ which is close to $\boldsymbol{A}$ but a lot simpler, i.e. $\boldsymbol{P} \approx \boldsymbol{A}$.

One can categorize iterative methods in three categories, i.e.

- Pure iterations or stationary iterations - does the same thing, over and over.
- Multigrid - solves the system on multiple grids of mixed granularity and interpolates the values when going from one grid to another. First smooths the problem with pure iterations (Jacobi or Gauss-Seidel) on fine grid and moves to coarse (smaller) grid.
- Krylov methods (Conjugate Gradient).

Choosing Preconditioner

Begin with the system of equations $\boldsymbol{Ax} = \boldsymbol{b}$. This can be written as

$$\begin{aligned} (\boldsymbol{I} + \boldsymbol{A} - \boldsymbol{I})\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{Ix} &= \boldsymbol{Ix} - \boldsymbol{Ax} + \boldsymbol{b}, \\ \boldsymbol{x} &= (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}, \end{aligned} \tag{2.52}$$

where $\boldsymbol{I}$ is the identity matrix. Let $\boldsymbol{P}$ be a matrix close to $\boldsymbol{A}$, but not too close. A choice of $\boldsymbol{A}$ as preconditioner would, in some sense, be perfect but it has already been established that the original system should *not* be solved, but a pertubed one.

Following the same logic as Eq. 2.52, one can write $\boldsymbol{Ax} = \boldsymbol{b}$ as

$$\begin{aligned} (\boldsymbol{P} + \boldsymbol{A} - \boldsymbol{P})\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{Px} &= \boldsymbol{Px} - \boldsymbol{Ax} + \boldsymbol{b}, \\ \boldsymbol{Px} &= (\boldsymbol{P} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}. \end{aligned} \tag{2.53}$$

Hence, the iteration becomes

$$\boldsymbol{P}\boldsymbol{x}^{(k+1)} = (\boldsymbol{P} - \boldsymbol{A})\boldsymbol{x}^{(k)} + \boldsymbol{b}, \tag{2.54}$$

where $\boldsymbol{x}^{(k+1)}$ (at iteration $k+1$) is a closer approximation of the true solution x than $\boldsymbol{x}^{(k)}$ was (hopefully).

Jacobi

One choice of $\boldsymbol{P}$ would be to take the diagonal of $\boldsymbol{A}$. This choice is called the *Jacobi* method. Starting from 2.53, the Jacobi method can be written as

$$\begin{aligned} \boldsymbol{D}\boldsymbol{x} &= (\boldsymbol{D}-\boldsymbol{A})\boldsymbol{x}+\boldsymbol{b}, \\ \boldsymbol{D}\boldsymbol{x} &= -(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}+\boldsymbol{b}, \\ \boldsymbol{x} &= \boldsymbol{D}^{-1}(\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}), \end{aligned} \tag{2.55}$$

where $\boldsymbol{L}$ is the strictly lower triangular part and $\boldsymbol{U}$ is the strictly upper triangular part of $\boldsymbol{A}$. This can be formulated as the iteration

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{D}^{-1}(\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}^{(k)}). \tag{2.56}$$

Jacobi is guaranteed to convergence if the matrix is *diagonally dominant*. This means that the greatest absolute value of all the numbers on a row must be on the diagonal. One can sometimes permute $\boldsymbol{A}$ so that it becomes diagonally dominant.

Component Form of the Jacobi Method

Equation 2.56 can also be written in component form, i.e.

$$\boldsymbol{x}_i^{(k+1)} = \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} \boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}\right). \tag{2.57}$$

Gauss-Seidel

A second choice of $\boldsymbol{P}$ would be to take *both* the diagonal *and* the lower triangular part of $\boldsymbol{A}$. This results in the *Gauss-Seidel* method. Starting from 2.53 once again, the method can be written as

$$\begin{aligned} (\boldsymbol{D}+\boldsymbol{L})\boldsymbol{x} &= ((\boldsymbol{D}+\boldsymbol{L})-\boldsymbol{A})\boldsymbol{x}+\boldsymbol{b}, \\ (\boldsymbol{D}+\boldsymbol{L})\boldsymbol{x} &= -\boldsymbol{U}\boldsymbol{x}+\boldsymbol{b}, \\ \boldsymbol{x} &= (\boldsymbol{D}+\boldsymbol{L})^{-1}(\boldsymbol{b}-\boldsymbol{U}\boldsymbol{x}), \end{aligned} \tag{2.58}$$

where $\boldsymbol{L}$ is the strictly lower triangular part and $\boldsymbol{U}$ is the strictly upper triangular part of $\boldsymbol{A}$. This can be formulated as the iteration

$$\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} + \boldsymbol{L})^{-1}(\boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}^{(k)}). \tag{2.59}$$

This has several advantages over Jacobi. Because $\boldsymbol{P}$ now is triangular, Eq. 2.54 can be solved with back substitution which updates the result vector $\boldsymbol{x}$. As soon as it have a new component $\boldsymbol{x}_i^{(k+1)}$ it uses it to calculate the rest of the components $\boldsymbol{x}_j^{(k+1)}$ (where $j > i$). This requires less storage because only one vector is needed.

Gauss-Seidel is guaranteed to converge if the matrix is diagonally dominant or if it is symmetric positive definite.

Component Form of the Gauss-Seidel Method

The component form is a bit more involved than the component form of the Jacobi method. First, rewrite Eq. 2.58 as

$$\begin{aligned} (\boldsymbol{D} + \boldsymbol{L})\boldsymbol{x} &= ((\boldsymbol{D} + \boldsymbol{L}) - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}, \\ \boldsymbol{D}\boldsymbol{x} + \boldsymbol{L}\boldsymbol{x} &= -\boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}, \\ \boldsymbol{D}\boldsymbol{x} &= \boldsymbol{b} - \boldsymbol{U}\boldsymbol{x} - \boldsymbol{L}\boldsymbol{x}, \\ \boldsymbol{x} &= \boldsymbol{D}^{-1}(\boldsymbol{b} - \boldsymbol{L}\boldsymbol{x} - \boldsymbol{U}\boldsymbol{x}). \end{aligned} \tag{2.60}$$

Then the iteration scheme becomes,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{D}^{-1}\left(\boldsymbol{b} - \boldsymbol{L}\boldsymbol{x}^{(k+1)} - \boldsymbol{U}\boldsymbol{x}^{(k)}\right), \tag{2.61}$$

and the component form of Gauss-Seidel can then be written as

$$\boldsymbol{x}_i^{(k+1)} = \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{n}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}\right). \tag{2.62}$$

Sometimes, one is more interested in the *difference* between next $\boldsymbol{x}$ and the current, i.e. $\Delta\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$. If this is the case, the old value of $\boldsymbol{x}$ can be included in the summation, giving

$$\begin{aligned} \Delta\boldsymbol{x}_i^{(k+1)} &= \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \textstyle\sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{n}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}\right) - \boldsymbol{x}_i^{(k)}, \\ \Delta\boldsymbol{x}_i^{(k+1)} &= \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \textstyle\sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{n}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}\right) - \dfrac{\boldsymbol{A}_{ii}\boldsymbol{x}_i^{(k)}}{\boldsymbol{A}_{ii}}, \\ \Delta\boldsymbol{x}_i^{(k+1)} &= \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \textstyle\sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i+1}^{n}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)} - \boldsymbol{A}_{ii}\boldsymbol{x}_i^{(k)}\right), \\ \Delta\boldsymbol{x}_i^{(k+1)} &= \frac{1}{\boldsymbol{A}_{ii}}\left(\boldsymbol{b}_i - \textstyle\sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j=i}^{n}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}\right). \end{aligned} \tag{2.63}$$

Other Choices

There are also a group of methods called over-relaxation methods. These uses weights to modify the above choices. One is called *Successive over-relaxation* or Sor.

Another choice is to use *Incomplete* Lu or Ilu. Here, $\boldsymbol{L}$ and $\boldsymbol{U}$ stands for the lower and upper triangular parts of $\boldsymbol{A}$. In Ilu P is chosen as $\boldsymbol{P} = \boldsymbol{L}_{approx}\boldsymbol{U}_{approx} \approx \boldsymbol{A}$ where the approximation comes from ignoring the *fill-in*. Fill-in often arises when factoring. Non-zeros can appear in spaces where $\boldsymbol{A}$ itself is zero. These empty spaces are "filled-in" with non-zeros causing $\boldsymbol{L}$ and $\boldsymbol{U}$ to become more dense. The idea of Ilu then becomes; *Only keep the non-zeros in spaces which corresponds to non-zeros in* $\boldsymbol{A}$*, don't allow fill-in.* A variant of this is to have a tolerance. If a fill-in is large, one might keep it.

Convergence

When does $\boldsymbol{x}^{(k)}$ approach the solution as k grows large, i.e. when does the iteration converge? One can formulate an error function as

$$\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}, \tag{2.64}$$

where $\boldsymbol{x}$ is the true solution. When inserting this in Eq. 2.54, the error equation can be formulated as

$$\boldsymbol{P}\boldsymbol{e}^{(k+1)} = (\boldsymbol{P} - \boldsymbol{A})\boldsymbol{e}^{(k)}. \tag{2.65}$$

Multiply this with $\boldsymbol{P}^{-1}$ to get

$$\boldsymbol{e}^{(k+1)} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})\boldsymbol{e}^{(k)}. \tag{2.66}$$

Thus, at each iteration, the error gets multiplied with the matrix

$$\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A}). \tag{2.67}$$

Note again that if $\boldsymbol{A}$ is chosen as preconditioner, then $\boldsymbol{I} - (\boldsymbol{A}^{-1}\boldsymbol{A}) = 0$ and the error disappears. It is this matrix $\boldsymbol{M}$ that controls the convergence. After k steps the error is $\boldsymbol{e}^{(k)} = \boldsymbol{M}^k\boldsymbol{e}^{(0)}$ meaning that $\boldsymbol{e}^{(0)}$ is effectively multiplied by the eigenvalues of $\boldsymbol{M}$. Therefore, it is the largest eigenvalue that matters most, and should be bound.

The convergence criterion therefore becomes

$$\forall\lambda \in \texttt{eig}(\boldsymbol{M}) : [|\lambda| < 1], \tag{2.68}$$

where `eig` returns all the eigenvalues of a given matrix. Considering the *spectral radius*, i.e. $\rho(\boldsymbol{M}) = \texttt{max}(\texttt{eig}(\boldsymbol{M}))$ one can rephrase the criterion to

$$|\rho(\boldsymbol{M})| < 1. \tag{2.69}$$

2.7 Collision Handling

Collision detection is a huge research area and a large problem when trying to simulate mechanical systems [Ericson, 2005]. An even larger problem is to *handle* the found collision in a physically plausible way.

In theory, one could simply apply the brute force strategy and check collisions by checking every body in the system against every other, i.e. an $O(n^2)$ operation. It does not take many bodies before this strategy becomes impractical. Even using Newton's third law[7] effectively halving the number of checks, it is still $O(n^2)$.

Many engines breaks down the problem in multiple phases, usually two or three. Some sort of global detection is first applied which is usually called *Broad-phase Collision Culling*. The goal here is to try and separate parts of the simulated world which cannot possibly collide. Pairs of bodies which has great distance between them are *culled* (not considered further) from the detection procedure.

After this phase, one can either do middle-phase culling or go directly to a *Narrow-phase Collision Detection*. The middle phase, if any, usually just applies another broad-phase algorithm with more information and detail.

The narrow-phase checks, in detail, if any collisions has occurred. Here, the geometries of the bodies are compared in order to find contacts. If it finds any it computes relevant information such as penetration depth and contact normals. The handling of the collision is often done later by a separate piece software.

In the following sections, some different algorithms are introduced and discussed. It start with the *Sweep and Prune* (or Sort and Sweep) and goes on to describe Spatial Hashing and similar techniques. Distance fields and distance maps are also discussed.

The information in this section is retrieved from Ericson [2005] unless otherwise stated.

2.7.1 Sweep and Prune

Sweep and prune is a method in which Axis Aligned Bounding Boxes (AABBs) are used.[8] These boxes are enclosing geometries which represents the boundaries of a given entity such as a body or a system of bodies.

[7] If body a exerts a force on body b, then b exerts a force on a that is equal in magnitude but in opposite direction.

[8] Actually, they should be called *Axis Aligned Bounding Rectangles* (AABR) in this 2D case but boxes are the conventional name and henceforth used throughout the text.

The algorithm starts by projecting all the Aabb's down on each axis (x and y). This produces an interval per Aabb on these axis with a low and high endpoint. All endpoints are sorted and parsed. If a low endpoint belonging to body a is found, then the high endpoint of a must follow, otherwise a's projection on that axis overlap with the projection of another object (body b), and a potential overlap between a and b is found. If a and b's projection on the other axis also overlap, their Aabb's must intersect. Therefore, narrow-phase detection between the bodies are required. Or, if middle phase is used, one could compare their *Oriented Bounding Boxes* (Obb) before going to narrow-phase.

This is not a very good algorithm if it is used improperly, but it gives a chance to use temporal coherence. This means that it takes advantage of the fact that most bodies do not travel a great distance from one time-step to the next, probably just a very short distance or none at all. The list of projected endpoints could therefore be stored. This will, however, invalidate the order of the list which will therefore need sorting in every time-step. For this, an algorithm such as *bubble sort* or *insertion sort* can be used which is very fast on nearly sorted lists with a time complexity of $O(n)$ [Ericson, 2005].

2.7.2 Grids

A common method, especially when dealing with uniformly sized object, is to place them in a grid of cells. This will make neighboring cells known and only a few cells needs to be searched in order to find all neighbors of a given particle. When dealing with neighbors and particles, one can define the *neighborhood* of particle p as all particles within the *radius of interaction*, see Fig. 2.3b.

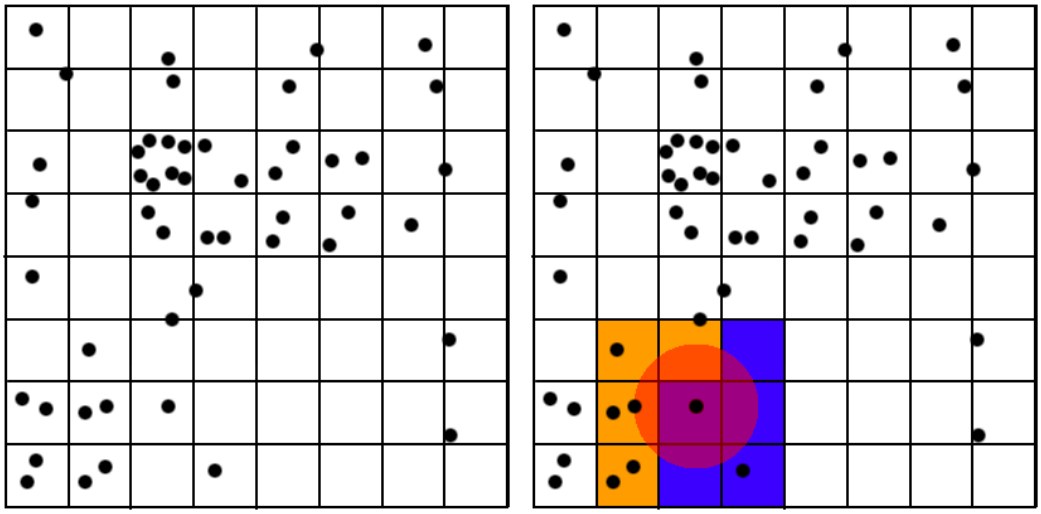
There are multiple ways of forming a grid. Here, three approaches will get a short presentation describing their basic ideas.

Explicit Grid

The most intuitive and easy approach is to create an explicit grid, i.e. all cells are explicitly created. This will introduce a rather large memory footprint because, in most cases, cells that are not used are still created and stored, see Fig. 2.3a. In the case of uniformly sized particles with some radius of interaction, red circle in Fig. 2.3b, the cell size can be chosen to equal this radius.

The grid is usually created with a preset size. This implies that the application must know the limits of the fluid, i.e. all places it can be. However, the modulo operation can be used to insert particles that are outside the limits.

Often, it is a good idea to apply Newton's third law to physics simulation. This makes it possible to only search 5 cells instead of 9 in the 2d case, see Fig. 2.3b. Note, however, that if only 5 cells are searched, the particle will not know about its other neighbors. The plan is to let *them* interact. This means that it *must* be clearly defined which particle should interact with which in all cases. In Fig. 2.3b, the particle in the middle of the red circle only searches the blue cells, i.e. from top right to bottom. However, the case where the particles are in the same cell must be handled as a special case. Here, one could choose



(a) Space is divided into a grid of cells.

(b) The neighborhood of a particle.

Figure 2.3: *An explicit grid (a) is created to store the particles. (b) The particle in the center of the red circle need only check for neighbors in the five blue cells on the right and bottom, for example.*

many solutions, for example the particle with shortest distance to the origin should do the interactions.

A slightly different version is to store the particles in a staggered grid. This structure stores the necessary values, such as position and velocity, in separate grids. This can increase caching and is often used in fluid simulations [Stam, 2003, Braley and Sandu, 2009].

Spatial Hashing

Spatial Hashing is an *implicit* formulation of the grid Teschner et al. [2003]. The particles positions are used to calculate a key,

$$hash(x, y) = (xp_1 \ \textbf{xor} \ yp_2) \ \textbf{mod} \ n \tag{2.70}$$

where p_1 and p_2 are two large prime numbers and n is the size of the hash table. This key is mapped to a row, or cell, in the hash table, where the particle is stored. This means that an often sparse two dimensional structure is mapped to a dense one dimensional structure. This often results in a smaller memory footprint than the use of explicit grid.

Different strategies and variants exists. For example, the grid can be emptied at each time-step *or* one can use a time-stamp to know if a cell is outdated. When a particle is about to be inserted, it checks if the cell is outdated. If so, the cell is first emptied, the time-stamp updated and the particles is inserted in the empty cell. Two insertion strategies could be to

either insert a given particle *only* in the cell with key equal to Eq. 2.70 *or* to insert the given particle in their neighboring cells too. The first choice stores less data, but the neighboring cells must be searched to find all neighbors. In the second choice, the cell that a particle is mapped to will contain the entire neighborhood of said particle. It is also possible to only store or search in 5 cells instead of 9 in the 2D case, as discussed above.

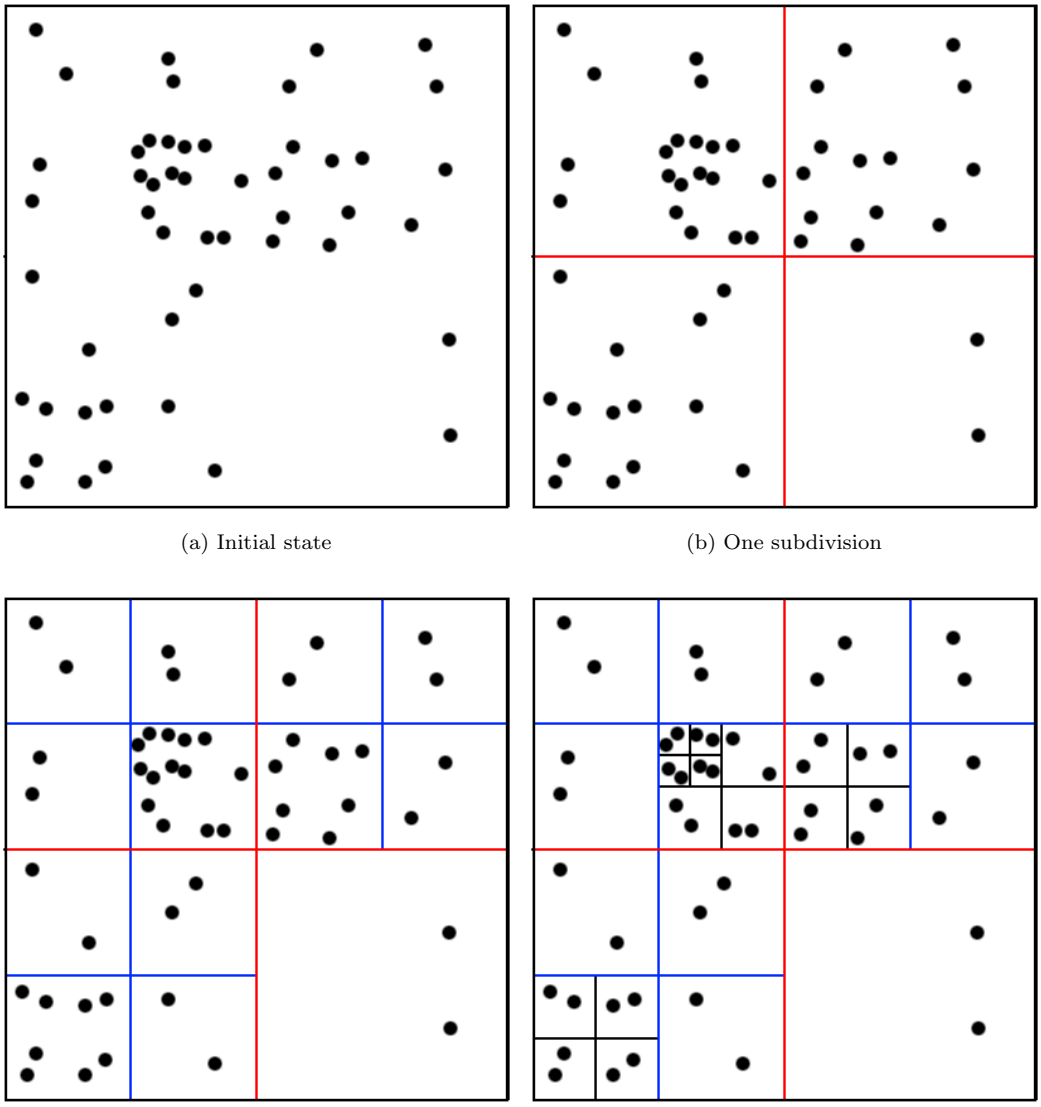
Quadtree (Octree)

The grid could also contain cells of different sizes. In the case of 2D, the space can be subdivided into four cells if the space contain more particles than some set threshold. Each of the new four cells could be further subdivided in four until the number of particles in each cell is less than the threshold, see Fig 2.4.

2.7.3 Signed Distance Maps

Signed distance maps are used to implicitly represent an objects collision geometry. In the context of fluid simulations, they are used to handle collisions between particles and the boundary.

A signed distance map is a discretized version of a *signed distance field*. A signed distance field is like a vector field, but returns the distance and the normal to the closest edge. The discretized version renders the object to a grid of some size, storing normals and distances from each cell's center to its closest edge. The particles positions are then used to fetch the cell containing the particle position and the information in there is used to handle the collision.



(a) Initial state

(b) One subdivision

(c) Two subdivision

(d) Process done

Figure 2.4: *Construction of a quad-tree. The grid in (a) is subdivided in four regions to give (b). Regions with more particles than a given threshold, in this case two, are further subdivided (c). This process is continued until all regions contain at most two particles.*

Chapter 3

Fluid Mechanics

Fluid mechanics is the study of fluids and is a sub-discipline of *continuum mechanics* i.e. it ignores the fact that matter is made up of a discrete number of particles and instead models matter on a macroscopic level or as a *continuum*. *Computational Fluid Dynamics* is the field of solving fluid mechanics problem and is therefore the field of interest in this thesis.

The chapter begins by describing the fundamental equations of fluid motion, the *Navier-Stokes equations*, Section 3.1. It then moves from theory to applications, starting with a group of methods called *Eulerian*, or *grid*-based methods, in Section 3.2. Section 3.3 describes another group of methods called *Lagrangian* or *particle*-based methods. This latter is the group of interest for this thesis.

The last chapter gives an in-depth description of one class of particle-based methods that is used throughout the thesis, called *Smoothed Particle Hydrodynamics*, found in Section 3.4.

3.1 Navier-Stokes Equations

The information in this section is gathered from Erleben et al. [2005] and Eberly [2010][1], unless otherwise stated.

The fundamental equation of fluid motion has been named after Claude-Louis Navier and George Gabriel Stokes who developed them in the early 19th century. This section will describe these equations, which assumptions are made and how the form used is derived.

Henceforth, the incompressibility assumption is also made, i.e. that water cannot be compressed. If this were true, sound could not be propagated through water and is therefore physically false. However, the amount of compression is so small that this assumption still makes sense and simplifies the computations.

Fluid, as well as all material, consists of molecules and atoms, i.e. a discrete number of

[1] Note that the first edition of Eberly [2010] lack the chapter on fluid dynamics.

particles that interact through forces. However, molecular scale aside, fluid can be assumed to be a continuum. This means that fluid quantities such as pressure, density and velocity are differentiable continuous functions. This is important as it gives an opportunity to derive fluid motion equations.

Important to note is that these equations describe a *model* of fluids. As such it does not exactly describe the world. Even less so when adding computers to solve them using discretized time. Also the Navier-Stokes equations are not fully understood which is why Clay Mathematics Institute has announced a one million dollar prize for proving the "existence and smoothness of Navier-Stokes solutions"[2].

3.1.1 Derivation

As with many physical equations describing motion, they stem from Newton's second law and so does the Navier-Stokes equations. Roots are also found in the conservation of momentum, mass and energy.

Newton's second law of motion ($m\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}$) can be reformulated for when the velocity is not a vector but a vector *field* $\boldsymbol{v}$. Using the substantial derivative from Eq. 2.17, this becomes

$$\rho\frac{D}{Dt}\boldsymbol{v} = \boldsymbol{f}, \tag{3.1}$$

where ρ is the density and $\boldsymbol{f}$ is the force acting on the fluid per unit volume. This force is referred to as *force density*.

Euler connected the gradient of pressure p in a fluid to the internal force. Pressure is a scalar function, taking a position and returning the pressure at that point. The *gradient* of a scalar field is a *vector field*. Substituting the force density of Eq. 3.1 for the negative gradient of pressure, the result becomes

$$\rho\frac{D}{Dt}\boldsymbol{v} = -\nabla p. \tag{3.2}$$

Euler also used the conservation of mass in order to state that the divergence of the vector field $\boldsymbol{v}$ must be zero. This means that in any small region of an incompressible fluid, the amount of fluid entering the region is exactly equal to the amount leaving the region, i.e.

$$\nabla \cdot \boldsymbol{v} = 0. \tag{3.3}$$

Eq. 3.2 and Eq. 3.3 is collectively known as the Euler's equations. However, these equations disregards friction between the water molecules. Here is where Navier and Stokes enter. Their version of Eq. 3.2 extends the right hand side with the addition of friction [Erleben et al., 2005].

[2] `http://www.claymath.org/millennium/`

Stress The stress in continuum mechanics is the internal force acting within a (deformable) body as a reaction to the external force acting on it. This is how friction behave. Friction is a reaction to external force and would be zero if there were no external force, as such it is a non-linear force.

The stress is measured in units of force per unit area N/m^2 which is the same unit as pascal Pa, the unit for pressure. In other words, stress has the same *dimension* as pressure.

A stress tensor contains the information about the stress and can be written as the sum of two stress tensor components which are called the hydrostatic and the deviatoric stress tensor. The hydrostatic component tends to change the volume of the fluid while the deviatoric tends to distort it.

When stress is added to Eq. 3.2, one derives the general form of the Navier-Stokes equations, which is

$$\rho \frac{D}{Dt}\boldsymbol{v} = -\nabla p + \nabla \cdot \mathbb{T} + \boldsymbol{f}. \tag{3.4}$$

Here $\mathbb{T}$ is the deviatoric stress tensor and ∇p is the pressure gradient and comes from the hydrostatic stress tensor[3]. The form of the deviatoric stress tensor is not known and requires that some hypothesis about the fluid is made. Throughout this paper, the fluids are assumed to be *Newtonian incompressible fluids* and will therefore not deal with other cases.

A Newtonian fluid has a linear relationship between shear stress and the *rate of strain* (or strain rate). The constant of proportionality here is the coefficient of viscosity. This means that if the shear stress τ increases, the rate of strain $\dot{\epsilon}$ also increases. This is written

$$\tau = \mu\dot{\epsilon} = \mu\frac{d\boldsymbol{v}}{dt}, \tag{3.5}$$

where $\dfrac{d\boldsymbol{v}}{dt}$ is the velocity gradient perpendicular to the direction of shear.

This, together with the following assumptions, give a simplified version of the Navier-Stokes equations, see Eq. 3.6 and 3.7.

- Viscosity μ is constant.
- The fluid is *isotropic*, i.e. the physical properties of the fluid has the same values when measured in different directions.
- When the fluid is at rest, then $\nabla \cdot \mathbb{T} = 0$, i.e. when at rest, the deviatoric stress disappears but the fluid is still subject to pressure due to gravity. This is similar to friction. There is no friction if the body is at rest.
- Including the conservation statement from Eq. 3.3.

[3]The hydrostatic stress tensor is $-pI$ where I is the identity matrix. However, only the *gradient* of pressure matters.

$$\rho \frac{D}{Dt}\boldsymbol{v} = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \boldsymbol{f}, \tag{3.6}$$

$$\nabla \cdot \boldsymbol{v} = 0. \tag{3.7}$$

The deviatoric stress tensor is now known and is the force due to viscosity. If the magnitude of this deviatoric stress tensor dominates the hydrostatic, the fluid is called a *viscous fluid*, i.e. $||\nabla p||_2 \ll ||\mu \nabla^2 \boldsymbol{v}||_2$. In the opposite case, the fluid is called *inviscid* [Erleben et al., 2005].

3.1.2 Convective Acceleration

A solution to the Navier-Stokes equations is a velocity field that, given a point in the fluid, tells the velocity of the fluid that passes through that point. This also means that the velocity field can be constant but the acceleration of the particle at the given point might be non-zero.

This effect of *time independent acceleration* of the particles in a fluid with respect to position is called *convective acceleration*. This might feel awkward and non-intuitive, but if the distinction is made between a point moving with the flow versus a static point in space, it might be clearer.

A *point moving with the flow*, like a leaf on a stream, might experience deceleration when the stream meets a lake, even though the flow itself is steady and *time independent*. A *static point in space*, however, will *not* experience any change in velocity over time, if the flow is steady. This acceleration of points moving with the flow is *convective acceleration* and is described by the non-linear term

$$\boldsymbol{v} \cdot \nabla \boldsymbol{v}. \tag{3.8}$$

This term is the second part of the material derivative, Eq. 2.17, used in Eq. 3.4 above.

3.2 Eulerian (or Grid-based) Methods

One way of simulating the Navier-Stokes equations is to superimpose a *grid* over the fluid region. This, in order to sample field quantities, such as velocity and density, at static points in the fluid, for example at the center of each cell [Braley and Sandu, 2009]. Such methods are referred to as *Eulerian methods*.

Advection is the key operation in Eulerian methods [West, 2007]. Advection refers to the transfer of heat or matter by the flow of a fluid. The grid sampling points are fixed which means that when a field quantity is advected, the position it was moved to is probably not exactly on another sampling point. Therefore, the field quantity must be interpolated to neighboring sampling points.

To distinguish between sampling points and the resulting location of a field quantity, one can refer to the latter as particles which are advected through the fluid. This is just conceptual, but is sometimes referred to as *Semi-Lagrangian advection*.

Another option is to follow Stam [2003] which does the opposite. His approach is to follow the *negative* velocity vector of every grid point in order to find the "particles" which would be *advected exactly to* a grid sampling point. This implies that the particles to advect must be evaluated by interpolating the neighboring sampling points. This is referred to as *backtracing advection* or *backward advection*.

This only works if the velocity field is mass conserving. Stam [2003] solves this by using the Hodge decomposition, i.e. every velocity field is the sum of a mass conserving field and a gradient field. The gradient field can be found by solving a Poisson equation and the mass conserving field can be found by subtracting the gradient field from the current velocity field to get the new, mass conserving, velocity field.

There are many different strategies around. West [2007] use both forward and backward advection and maintains mass conservation by making sure that when field quantities are moved, the amount added to the new point is subtracted from where it came.

Diffusion is also a vital part. This is the gradient of divergence, i.e. the rate of change in divergence over time. It is solved with a Gauss-Seidel relaxation scheme in Stam [2003].

One large problem with Eulerian methods is that the space where the fluid is allowed to be must be predetermined. This might not be a problem if the application is a wind tunnel for example. However, in games where the user is allowed to interact and perhaps move fluid from one place to another, it can be a problem. One solution could be to make the grid span all space that the player can move in. However, the algorithm is equally expensive if no fluid is in the grid as if the grid is full which diminishes the solutions effectiveness on small volumes of fluid.

3.3 Lagrangian (Particle-based) Methods

The Navier-Stokes equations for incompressible fluid, restated here for convenience,

$$\rho \frac{D}{Dt}\boldsymbol{v} = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \boldsymbol{f}, \tag{3.9}$$

$$\nabla \cdot \boldsymbol{v} = 0, \tag{3.10}$$

can be simplified when using the Lagrangian formulation instead of the Eulerian.

Because the number of particles as well as their mass is constant, the conservation of mass is automatically upheld when using particles [Müller et al., 2003]. This leaves only the statement about the conservation of momentum (Eq. 3.9). The left hand side of this equation can be written

$$\rho \frac{D}{Dt}\boldsymbol{v} = \rho\left(\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\nabla\boldsymbol{v}\right). \tag{3.11}$$

Because the particles are convected (moved) *with* the fluid the dreaded non-linear convective acceleration term $\boldsymbol{v}\cdot\nabla\boldsymbol{v}$ can be omitted from Eq. 3.11 [Müller et al., 2003]. Also, dividing the right hand side of Eq. 3.9 with ρ yields

$$\frac{\partial \boldsymbol{v}}{\partial t} = \frac{-\nabla p}{\rho} + \frac{\mu\nabla^2\boldsymbol{v}}{\rho} + \frac{\boldsymbol{f}}{\rho}. \tag{3.12}$$

This is now in a very neat form where the time derivative of the velocity field is calculated by summing three different acceleration contributions,

$$\boldsymbol{a}_{pressure} = \frac{-\nabla p}{\rho}, \tag{3.13}$$

$$\boldsymbol{a}_{viscosity} = \frac{\mu\nabla^2\boldsymbol{v}}{\rho}, \tag{3.14}$$

$$\boldsymbol{a}_{external} = \frac{\boldsymbol{f}}{\rho}. \tag{3.15}$$

Recall that $\boldsymbol{f}$ is force *density*, i.e. $\boldsymbol{f} = \rho\boldsymbol{a}$. If the only external force was gravity, then $\boldsymbol{f} = \rho\boldsymbol{g}$ and $\boldsymbol{a}_{external} = \boldsymbol{g}$.

3.4 Smoothed Particle Hydrodynamics (SPH)

Smoothed Particle Hydrodynamic or Sph in short, is a Lagrangian method for simulating fluids. It was introduced by Gingold and Monaghan [1977] and independently by Lucy [1977] for simulation of astrophysical problems, e.g. the flow of interstellar gas. However, the formulation is general enough for any fluid flow.

The information in this section is retrieved from Müller et al. [2003], Kelager [2006], Bodin et al. [2011] and Monaghan [2005] unless otherwise stated.

The idea is to define fluid quantities at the location of the particles and then interpolate these quantities over the entire fluid. This will effectively *smooth* the quantities over the fluid in order to approximate the real quantities. The interpolation scheme uses a weighted sum of the contributions from all particles, including itself, taking the general form

$$a(\boldsymbol{r}_i) = \sum_j^n m_j \frac{a_j}{\rho_j} W(\boldsymbol{r}_i - \boldsymbol{r}_j, h), \tag{3.16}$$

where i is the index of the interpolation point, or particle, under consideration. The sum is over all n particles in the system, a is a scalar quantity, $\boldsymbol{r}$ is the position, m and ρ is

the mass and the density, respectively. $W(\boldsymbol{r}, h)$ is the weight function, or kernel, which is further described below.

In practice, it would not be feasible to interpolate over all the particles as it would result in an $O(n^2)$ algorithm. Instead, the quantities are only interpolated in the *neighborhood* of a particle[4], i.e. all the particles within the *smoothing length*, denoted h, from the particle. This gives an improved complexity of $O(nm)$ where m is the average number of neighbors, given that the neighbors are already found. The total complexity is therefore $O(nm) + O_{nfind}$ where O_{nfind} is the complexity for finding the neighborhood (usually between $O(n)$ to $O(n\log(n))$

3.4.1 Kernel Function

The kernel function is the key to a good interpolation. Many different functions can be used but it must follow some prerequisites. It must strike it's peak value when the given vector $\boldsymbol{r}$ is of length zero. Then it monotonically *decreases* as the length of $\boldsymbol{r}$ *increases*, eventually returning zero when distance is equal to, or higher than, the smoothing length h. In other words it makes sure that far away neighbors gets less than close neighbors. More formally

$$\begin{aligned} W(\boldsymbol{r}, h) : \mathbb{R}^{d+1} \to \mathbb{R} \quad & \text{for } d\text{-dimensional vector } \boldsymbol{r}, \\ W(\boldsymbol{r}, h) \neq 0 \quad \text{iff} \quad 0 \leq ||r|| \leq h \quad & \text{non-zero only inside the smoothing length,} \\ \forall \boldsymbol{r} \in \mathbb{R}^d (W(\boldsymbol{r}_1, h) \geq W(\boldsymbol{r}_2, h) \text{ iff } ||\boldsymbol{r}1|| \leq ||\boldsymbol{r}_2|| \quad & \text{monotonically decreasing.} \end{aligned} \tag{3.17}$$

Kernel functions should also be even, normalized and differentiable, i.e.

$$\begin{aligned} W(\boldsymbol{r}, h) = W(-\boldsymbol{r}, h) \quad & \text{(even)}, \\ \int W(\boldsymbol{r}, h) d\boldsymbol{r} = 1 \quad & \text{(normalized)}. \end{aligned} \tag{3.18}$$

The kernels used in this thesis are explained in Appendix A. Here, popular 3D kernels are redefined for 2D.

[4]The neighborhood also includes the particle under consideration as neighbor.

3.4.2 Field Quantities

Density

Using Eq. 3.16, the interpolation of the density field quantity can now be formulated as

$$\rho_i = \sum_j m_j \frac{\rho_j}{\rho_j} W(\boldsymbol{r}_{ji}, h) = \sum_j m_j W(\boldsymbol{r}_{ji}, h). \tag{3.19}$$

Pressure

The expressions for the force due to pressure is not as easily constructed. First, the pressure must be computed. For this, one usually uses either of the three formulas

$$p = k\rho, \tag{3.20}$$
$$p = k(\rho - \rho_0), \tag{3.21}$$
$$p = c_s^2(\rho - \rho_0), \tag{3.22}$$

where k is the idealized gas constant, c_s is the speed of sound in the fluid and ρ_0 is the reference density. These are all based on the speed of sound, or the propagation of pressure waves. Eq. 3.21 is preferred over Eq. 3.20 according to Müller et al. [2003], due to use of reference density but says nothing about 3.22. Bodin et al. [2011] mentions that the pressure can be approximated with the Tait equation of state

$$p = \frac{p_0 c_s^2}{\gamma} \left(\left(\frac{\rho_i}{\rho_0} \right)^\gamma - 1 \right). \tag{3.23}$$

From this, Eq. 3.22 is derived when setting $\gamma = 1$ which gives models that are referred to as *pseudo-compressible* Sph. A choice of $\gamma = 7$ gives models that are weakly incompressible. This latter choice will obviously give a more incompressible fluid model but the time-step would have to be prohibitively small. This basically says that the above pressure models are all crude approximations that has to be made to keep the time-step reasonably large.

Pressure Force

To get the interpolated value of the pressure force, first note that the pressure term in Eq. 3.9 would generate

$$-\nabla p(\boldsymbol{r}_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W_{pressure}(\boldsymbol{r}_{ji}, h), \tag{3.24}$$

then note that the pressure force is equal to the acceleration (see Eq. 3.13) times mass. This means that the pressure force is

$$\begin{aligned} \boldsymbol{f}_i^{pressure} &= \frac{-m_i \nabla p(\boldsymbol{r}_i)}{\rho_i} &= -\frac{m_i}{\rho_i} \sum_j m_j \frac{p_j}{\rho_j} \nabla W_{pressure}(\boldsymbol{r}_{ji}, h) \\ &= -\sum_j m_i m_j \frac{p_j}{\rho_i \rho_j} \nabla W_{pressure}(\boldsymbol{r}_{ji}, h). \end{aligned} \tag{3.25}$$

This formulation is a bit problematic though, because it is not symmetric. Quoting [Müller et al., 2003]

> *It is important to realize that* Sph *holds some inherent problems. When using* Sph *to derive fluid equations for particles, these equations are not guaranteed to satisfy certain physical principals such as symmetry of forces and conservation of momentum.*

In Colin et al. [2006] they examine two other options of obtaining an approximation of the derivatives of a function except the above approach of using the derivatives of the smoothing kernel, which they call "basic gradient approximation formula" (or Bgaf). This derivative is exact but the function is approximate which can give slightly inaccurate results.

The two other formulae are referred to by Colin et al. [2006] as "difference gradient approximation formula" (or Dgaf)[5],

$$\nabla f_i = \sum_j m_j \frac{f_j - f_i}{\rho_i} \nabla W(\boldsymbol{r}_{ji}, h), \tag{3.26}$$

and "symmetric gradient approximation formula" (or Sgaf)[6],

$$\nabla f_i = \sum_j \rho_i m_j \left(\frac{f_i}{\rho_i^2} + \frac{f_j}{\rho_j^2} \right) \nabla W(\boldsymbol{r}_{ji}, h). \tag{3.27}$$

The Dgaf approach turned out to offer the most accurate differential of the three, according to Colin et al. [2006] but is still not symmetric in accordance to Newton's third law. The Sgaf gave worse accuracy but is still often referred to and used, e.g. in Grahn [2008].

Another approach proposed by Müller et al. [2003] is to simply take the geometric mean of the two pressure values in the current pair, i.e.

$$-\sum_j m_i m_j \frac{p_i + p_j}{2\rho_i \rho_j} \nabla W_{pressure}(\boldsymbol{r}_{ji}, h). \tag{3.28}$$

[5]The Dgaf is derived from the derivative of a *product*, $\nabla(\rho f) = \rho \nabla f + f \nabla \rho$.

[6]The Sgaf is derived from the derivative of a *quotient*, $\nabla \left(\frac{f}{\rho} \right) = \frac{\rho \nabla f - f \nabla \rho}{\rho^2}$.

Müller et al. [2003] claims that this approach gives good stability and performance. Because the article by Müller et al. [2003] is the basis for one of the methods in this paper, this formulation is henceforth used.

Viscosity force

Now turning to viscosity. Colin et al. [2006] gives the following formula for the Dgaf approach for the Laplacian

$$\nabla^2 f_i = \sum_j m_j \frac{f_j - f_i}{\rho_i} \left(\nabla^2 W(\boldsymbol{r}_{ji}, h) - \frac{2}{\rho_i} \nabla W(\boldsymbol{r}_{ji}, h) \cdot \nabla \rho_i \right). \tag{3.29}$$

Because ρ_i should equal ρ_0 at all times (in incompressible fluids) and becuase ρ_0 is constant then $\nabla \rho_i = 0$. Therefore, Eq. 3.29 can be written

$$\nabla^2 f_i = \sum_j m_j \frac{f_j - f_i}{\rho_i} \nabla^2 W(\boldsymbol{r}_{ji}, h). \tag{3.30}$$

With this in mind now looking at the viscosity term in Eq. 3.9 together with Eq. 3.30 results in

$$\boldsymbol{f}_i^{viscosity} = \frac{m_i \mu \nabla^2 \boldsymbol{v}(\boldsymbol{r}_i)}{\rho_i} = \mu \textstyle\sum_j m_i m_j \dfrac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_i^2} \nabla^2 W_{viscosity}(\boldsymbol{r}_{ji}, h). \tag{3.31}$$

This is now very easy to symmetrize. Simply change $\rho_i \rho_i$ to $\rho_i \rho_j$, because these two values should be equal in an incompressible fluid, resulting in

$$\boldsymbol{f}_i^{viscosity} = \mu \sum_j m_i m_j \frac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_i \rho_j} \nabla^2 W_{viscosity}(\boldsymbol{r}_{ji}, h). \tag{3.32}$$

Again, this is the choice made in Müller et al. [2003] and many other papers. The intuition of Eq. 3.32 is that it is only *relative* motion that causes friction. There is no friction between objects traveling at the same velocity. Therefore, the calculations of $f_{viscosity}$ is from particles i's moving frame of reference.

The Smoothed Particle Hydrodynamics method with the above choices is summarized in Algorithm 3.1. Note that the forces are symmetric so it would probably be beneficial to update the neighbours forces in accordance to Newton's third law. Cases where this is not the obvious approach is when implementing for some accelerating hardware such as Gpu, or some other parallel system.

Algorithm 3.1 *Basic algorithm for Smoothed Particle Hydrodynamics*

for all particles i **do**
  for all neighbors j of i **do**
    $\rho_i \leftarrow \rho_i + m_j W(\boldsymbol{r}_{ji}, h)$
  end for
  $p_i = c_s^2(\rho_i - \rho_0)$
end for
for all particles i **do**
  for all neighbors j of i **do**

$$\boldsymbol{f}_i^{pressure} \leftarrow \boldsymbol{f}_i^{pressure} - m_i m_j \frac{p_i + p_j}{2\rho_i \rho_j} \nabla W(\boldsymbol{r}_{ji}, h)$$

$$\boldsymbol{f}_i^{viscosity} \leftarrow \boldsymbol{f}_i^{viscosity} + \mu m_i m_j \frac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_i \rho_j} \nabla^2 W(\boldsymbol{r}_{ji}, h)$$

  end for
  $\boldsymbol{f}_i \leftarrow \boldsymbol{f}_i^{pressure} + \boldsymbol{f}_i^{viscosity} + \boldsymbol{f}_i^{external}$
  $\boldsymbol{a}_i \leftarrow \dfrac{\boldsymbol{f}_i}{m_i}$
end for

3.4.3 Parameters

Reference Density

In real life, the density of water is 1000 kg/m^3. However, in two dimensions the unit *cubic meter* is undefined and hence the concept of density. There is no "mass per unit square" because there is no such thing as point masses. One could possibly argue that a 2D plane, with sides x and y, is the limit of the corresponding 3D cube when letting the z-component approach zero. Then the volume, for all value of x and y, approaches zero, see Eq 3.33, leaving no fluid and the concept of density vanishes as well.

$$\lim_{z \to 0} xyz = 0. \tag{3.33}$$

However, the density of the fluid in the above 3D cube is constant as long as there is fluid left in the cube. So the limit as z goes to zero is

$$\lim_{z \to 0} \rho = \rho. \tag{3.34}$$

This might be a reason to conclude that the density of water in two dimensions is still 1000 kg/m^3, whatever that means.

Throughout this paper, the mapping from three dimensions to two dimensions has been discussed. In most cases, it is simply a matter of removing one dimension and it is not strange nor unexpected that such reductions are not in accordance with physical laws. The fluid particles are simulated as point masses with an influence radius covering a circle *area*

in two dimensions. When multiplying such area with density, mass should result. However, units involving m^3 will give mass per meter,

$$1m^2 \cdot 1000 \text{ kg/m}^3 = 1000 \text{ kg/m}. \tag{3.35}$$

Certainly, one can just reduce the dimensionality of the density and say that it is 1000 kg/m^2, based on the fact that the vector pointing in direction of z vanishes in the limit in Eq. 3.34, leading to rank deficiency and inability to span the three dimensional space.

However, questions arise. What does it mean when the dimensionality is reduced? What is the resulting behavior? One square meter of water would weigh 1000 kg, the same as one cubic meter. To move both one square meter and one cubic meter of fluid, assuming the fluid is contained in a massless container, the same amount of force would have to be applied to both fluids in order to move them an equal distance. In a 2D game environment, a 3D world is often imagine and then projected to simplify. A choice of 1000 kg/m^2 would entail that all fluid bodies have one meter "inward width" (along the z-axis) in this 3D world. A choice of say 100 kg/m^2 would instead suggest that all fluid are one *deci*meter wide and much easier to move. Which choice to make is unclear, it might even be a good idea to introduce such width as a parameter.

In this context ropes, trees and characters should be able to interact with the fluid, these are *not* usually one meter wide, rather in the magnitude of decimeters so a density of 100 kg/m^2 is chosen.

Smoothing Length

Assuming for a moment that all particles are rigid circles of radius r. Then, the smoothing length must be at least $2r$ for the kernels to be able to have an effect on neighboring particles, and exactly $2r$ to repulse only contacting circles. In this case, the self contribution of density should equal the reference density. To see this, assume that the smoothing length is $h = 2r$. Then the area covered by *one* particle is given by

$$a = r^2\pi = \left(\frac{h}{2}\right)^2 \pi, \tag{3.36}$$

where r is the radius of a particle. Given the area, the mass of *one* particle is

$$m = a\rho_0, \tag{3.37}$$

where ρ_0 is the reference density. Inserting Eq. 3.36 in Eq. 3.37 to eliminated area one gets

$$m = \left(\frac{h}{2}\right)^2 \pi\rho_0 \Rightarrow \tag{3.38}$$

$$h = 2\sqrt{\frac{m}{\pi\rho_0}}. \tag{3.39}$$

Consider Eq. 3.19 with self contribution only[7] when using the two dimensional poly6 kernel (Eq. A.8),

$$\rho = mW^{2D}_{poly6}(0,h) = m\frac{4}{\pi h^8}h^6 = \frac{4m}{\pi h^2}. \tag{3.40}$$

Inserting Eq. 3.39 in Eq. 3.40 gives

$$\rho = \frac{4m}{\pi\left(2\sqrt{\frac{m}{\pi\rho_0}}\right)^2} = \frac{4m}{\left(\frac{4m}{\rho_0}\right)} = \rho_0. \tag{3.41}$$

Eq. 3.39 gives an equation for the smoothing length in two dimensions in the case of using rigid circles as particles. However, using rigid circles, the simulation would look more like sand or a pile of small balls.

In any case, the self contribution of one lone particle *must not be larger* than the reference density. If this was larger, then every point in the simulated fluid would always have higher density than the reference density regardless of pressure, viscosity and other forces. This is self-contradictory and not physical.

This leaves the case where the self contribution is less than the reference density. This would cause nearby particles to be attracted so that they each sum up to the reference density, see Fig 3.1. In some methods, this can result in a tendency to build small "clusters", i.e. groups of particles which together contribute with enough density so that groups are no longer attracted to each other, see Fig. 3.2.

The calculation of the smoothing length can be changed a bit, to incorporate the amount of self-contribution. This is done by modifying Eq. 3.39 to

$$h = \sqrt{\frac{4m}{\pi\beta\rho_0}}, \tag{3.42}$$

where $0 < \beta \leq 1$. β in Eq. 3.42 can be seen as the percentage of the reference density that should equal the self contribution, i.e. 80 % ($\beta = 0.8$) would result in slight attraction between particles, while 100 % ($\beta = 1$) would result in no attraction (rigid circles).

[7] The summation obviously disappears and $r = 0$.

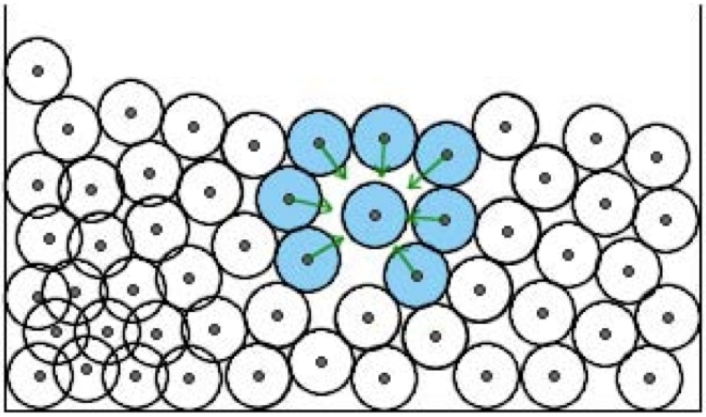


Figure 3.1: *Area of low density is highlighted and the force acts in the direction of the arrows in order to reach the reference density. This in the case that the self contribution is smaller than the reference density. Image from Kelager [2006]. Used with permission.*

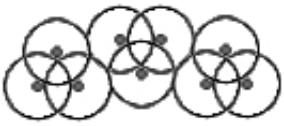


Figure 3.2: *Three small cluster-groups of three particles each. Each particle in a cluster gets enough density contribution from the cluster to sum up to the reference density. Thus, the attractive force vanishes* between *clusters.*

This result is based on the use of the poly 6 kernel for two dimensions. If another kernel is used, the result might need to be scaled. Because of this, Eq. 3.42 can be modified further to make it even more general. Consider

$$h = \sqrt{\frac{4mc}{\pi\beta\rho_0}}, \tag{3.43}$$

where c is some constant that is related to the normalization constant of the kernel used to calculate the density. In the above example c is obviously equal to 1.

Number of Neighbors

Some argue that the smoothing length should be chosen so that 15-20 neighbors are considered in three dimensions. There are probably many ways of mapping these values to two dimensions. However, a simple solution is to consider that x neighbors should be used in a one dimensions and that x^3 should be used in three. Then

$$\begin{aligned} x_{low} &= \sqrt[3]{15} \approx 2.46, \\ x_{high} &= \sqrt[3]{20} \approx 2.71. \end{aligned} \tag{3.44}$$

It seems reasonable that roughly two neighbors, and a bit of the third neighbor, should be used on a one dimensional line. For two dimensions,

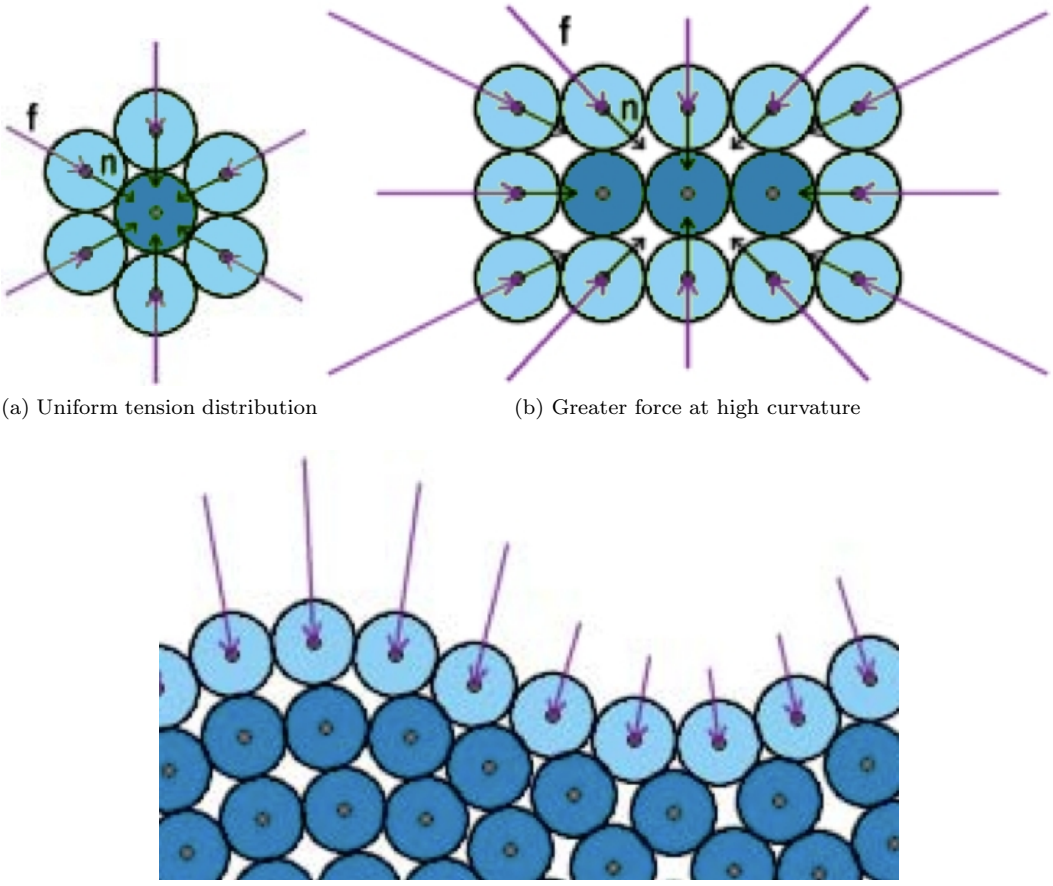
$$\begin{aligned} x_{low}^2 &= \sqrt[3]{15}^2 \approx 6.08, \\ x_{high}^2 &= \sqrt[3]{20}^2 \approx 7.37. \end{aligned} \tag{3.45}$$

This also seems reasonable. When executing informal experiments, $\beta = 1.0$ in Eq. 3.43 leads to an average around 6 neighbors when the fluid is at rest and $\beta = 0.6$ gives around 7 when the fluid is *not* subject to extreme settings, such as high pressure.

3.4.4 Surface Tension

In fluids, as well as solids, the molecules are subject to attractive forces from nearby molecules. Inside the fluid these attractive forces are symmetric, i.e. equal in all directions, and therefore cancels. However, near the surface they are no longer balanced. This difference is called *force due to surface tension* [Müller et al., 2003].

This force will try to minimize the curvature of the surface, i.e. it will try to enforce a minimum surface area by flattening the surface [Kelager, 2006]. This suggests that the direction is always in direction of the inward surface normal, i.e. towards the fluid, see Fig 3.3.



(a) Uniform tension distribution

(b) Greater force at high curvature

(c) Greater force at positive curvature

Figure 3.3: *The surface tension is in direction of the inward surface normal, i.e.* towards *the fluid. (a) The force is equal in magnitude and directed towards the center. (b) The force is greater where the curvature is large. (c) The force is greater where the curvature is positive, i.e. greater force on convex parts of the surface rather than the concave parts. Image from Kelager [2006]. Used with permission.*

The surface can be tracked using a field quantity, usually referred to as a color field [Müller et al., 2003]. The value of this color field is set to 1 only at the particles locations and zero everywhere else. This field quantity can then be smoothed using the Sph idea i.e. in accordance with Eq. 3.16,

$$c(\boldsymbol{r}_i) = \sum_j^n m_j \frac{c_j}{\rho_j} W(\boldsymbol{r}_i - \boldsymbol{r}_j, h) = \sum_j^n \frac{m_j}{\rho_j} W(\boldsymbol{r}_i - \boldsymbol{r}_j, h). \tag{3.46}$$

The surface tension model used in Müller et al. [2003] is initiated by noting that the gradient of the smoothed color field gives the unnormalized surface normal,

$$\boldsymbol{n}_i = \nabla c_i = \sum_j^n \frac{m_j}{\rho_j} W(\boldsymbol{r}_{ji}, h). \tag{3.47}$$

The Gaussian curvature of the surface is measured by the divergence of $\hat{\boldsymbol{n}}$, i.e.

$$\kappa = \frac{\nabla \boldsymbol{n}}{|\boldsymbol{n}|} = \frac{-\nabla^2 c}{|\boldsymbol{n}|}. \tag{3.48}$$

It is negated to get the correct sign, i.e. positive curvature for convex volumes. With the above, one can formulate *surface traction* as

$$\boldsymbol{t}^{surface} = \sigma \kappa \hat{\boldsymbol{n}}. \tag{3.49}$$

Surface traction is the force density per unit area acting on the surface of a fluid. This force should only be applied on the surface particles of the simulated fluid. This could be done in a variety of ways but Müller et al. [2003] suggests multiplying Eq. 3.49 with the length of the (unnormalized) normal $|\boldsymbol{n}|$. This is only non-zero near the surface because elsewhere the pairwise normals are symmetric and its sum is zero. This is not quite true in reality, though. Partly due to numerical errors when $|\boldsymbol{n}|$ is small but there is also risk of a slight non-symmetry in neighbor distribution when simulating fluids with Sph. This, because practical time-steps give rise to compressible fluids with higher distribution of neighbors in some areas. Therefore, some threshold values must be chosen. For example, only apply forces if $|\boldsymbol{n}| > \epsilon$ for some value of ϵ. As extension to this, it might be a good idea to also check the number of neighbors. Only apply said force if the particle has less than the desired number of neighbors *and* $|\boldsymbol{n}| > \epsilon$.

The force density for surface tension can now be written,

$$\boldsymbol{f}^{surface} = -\sigma \nabla^2 c \hat{\boldsymbol{n}}, \tag{3.50}$$

or as regular force on a given particle i

$$\boldsymbol{f}_i^{surface} = -\frac{m_i}{\rho_i}\sigma\nabla^2 c_i \hat{\boldsymbol{n}}_i. \tag{3.51}$$

Note that this is a crude model. It is asymmetric and therefore goes against Newton's third law. Real surface tension arise when fluids interact e.g. when water is in contact with air. In such interfaces the air molecules are *reacting* to the force due to surface tension *acting* on the surface of the water.

3.4.5 The Boundary Problem

Boundary problem is the problem of making boundaries impervious. This introduces irregularities in the smoothing area of the kernels causing errors in the calculated field quantities such as density. In other words, a fluid particle in a corner will have less neighbors than a particle in the midst of fluid. Many suggested solutions exist. Perhaps the easiest is to project the particles that are outside the boundary back inside.

Another approach is to use fixed particles at the boundaries making sure that particles near boundaries still get sufficiently many neighbors. How to distribute these particles are not trivial in arbitrary cases. Some use several layers, or mirror the neighborhood of near boundary particles in the tangent of the boundary [Yildiz et al., 2008]. There are many solutions and versions. The boundary problem *does not* exclude moving bodies.

Chapter 4

SPH Methods

Three related methods of simulating fluids have been implemented and compared to each other. This, with the goal to learn about different aspects of particle-based fluid simulation and shed some light on similarities and differences in order to hopefully see what constitutes a "good" fluid simulation.

4.1 The Chosen Methods

The three methods are based on the papers by Müller et al. [2003], Clavet et al. [2005] and Bodin et al. [2011].

Müller et al. [2003] is chosen for its importance in interactive fluid simulation. Their work is cited by many and some or all of their proposed set of kernels are often found in a multitude of articles and other contexts.

Clavet et al. [2005] is chosen because of their different approach to SPH. One example is the use of so called "near" density, as extension to regular density, in order to avoid clustering but yet keep an implicit surface tension. Handling density in such way is clearly not physical but highly efficient and easy to implement. Therefore, variations of it is often seen on forums and unofficial implementations [Madams, 2010]. The article puts its main focus on viscoelastic substances, i.e. substances that stick on objects, such as goo and phlegm. However, to ease comparison, the special methods concerning viscoelasticity are removed.

Another different approach to SPH is taken in Bodin et al. [2011] which, again, is why it was chosen. From a theoretical view, it is more advanced than the other two methods. However, it is still rather intuitive and easy to overview. Bodin et al. [2011] manage to compactly describe and solve the complete system where most others scatter the problems and solve them individually. It also promises incompressible SPH fluid simulation at relatively small time-step. This is also the only paper that describe a 2D implementation with accompanying remarks.

Table 4.1: *Key comparison features of the methods, except neighbor search and boundary detected included for completeness.*

	Müller	Clavet	Bodin
Kernels	W_{poly6} for density, W_{spiky} for pressure and $W_{\text{viscosity}}$ for viscosity	Quadratic spike $(1 - r/h)^2$ and $(1 - r/h)^3$ for near density. Linear $(1 - r/h)$ for viscosity. These are all used unnormalized.	Not specified
Pressure	$p_i = k(\rho_i - \rho_0)$	$p_i = k(\rho_i - \rho_0)$ and $p_i^{\text{near}} = k^{\text{near}}\rho_i^{\text{near}}$ for near pressure	Not needed
Surface tension	Calculated explicitly with color field	Falls out naturally due to use of two densities and two pressures	Not used
Boundary collisions	Projection of position and reflection of velocity	Tailor-made. Impulse-based three step procedure. Project if needed.	Density Field
Integration	Leap-frog	Prediction-relaxation	Spook
Neighbor search	Staggered Grid	Spatial Hashing	Sweep-and-prune
Boundary detection	Not specified	Signed Distance Fields	Signed Distance Fields

4.2 Overview of Original Methods

Table 4.1 show the different choices that the authors have taken. Some of more importance than others. For example, neighbor search and boundary detection has no effect other than performance[1].

4.3 Overview of Modified Methods

The methods in the papers are slightly modified and interpreted to fit the intended purpose. Table 4.2 is similar to table 4.1 but with the modifications shown.

[1] Performance is, of course, of importance but it poses no visible effect on the simulation.

Table 4.2: *Key comparison features of the modified methods. The gray cells represent changes from table 4.1.*

	The Müller method	The Clavet method	The Bodin method
Kernels	W_{poly6} for density, W_{spiky} for pressure and $W_{\text{viscosity}}$ for viscosity	Hybrid kernels, see App. A.3	W_{poly6} for density, W_{cons} for constraint direction
Pressure	$p_i = k(\rho_i - \rho_0)$	$p_i = k(\rho_i - \rho_0)$ and $p_i^{\text{near}} = k^{\text{near}}\rho_i^{\text{near}}$ for near pressure	Not needed
Surface tension	Calculated explicitly with color field	Falls out naturally due to use of two densities and two pressures	Not used
Boundary collisions	Impulse-based. Project if needed.	Impulse-based. Project if needed.	Density Field
Integration	Semi-implicit Euler	Prediction-relaxation	Spook
Neighbor search	Staggered Grid	Staggered Grid	Staggered Grid
Boundary detection	Grid Map	Grid Map	Grid Map

4.4 Description of Modified Methods

4.4.1 The Müller Method

The accelerations from the interaction between fluid particles using the Müller method is calculated using Algorithm 4.1. This algorithm is basically the same as the base SPH method, found in Algorithm 3.1. However, it updates both the current particles *and* its neighbors. If a parallel version is used, it could either iterate over *all* neighbors or use separate buffers for the pressure and viscosity accelerations and then add the contributions from all threads.

Algorithm 4.1 *Calculation of new accelerations for the particles with the Müller method.*

for all particles i **do**
  for all neighbors j of i $(j > i)$ **do**
    $\rho_{\text{temp}} \leftarrow mW^{2D}_{poly6}(\boldsymbol{r}_{ji}, h)$
    $\rho_i \leftarrow \rho_i + \rho_{\text{temp}}$
    $\rho_j \leftarrow \rho_j + \rho_{\text{temp}}$
  end for
  $p_i \leftarrow c_s^2(\rho_i - \rho_0)$
end for
for all particles i **do**
  for all neighbors j of i $(j > i)$ **do**
    $\boldsymbol{p}_{\text{temp}} \leftarrow m\dfrac{p_i + p_j}{2\rho_i\rho_j}\nabla W^{2D}_{spiky}(\boldsymbol{r}_{ji}, h)$
    $\boldsymbol{a}_i^{pressure} \leftarrow \boldsymbol{a}_i^{pressure} - \boldsymbol{p}_{\text{temp}}$
    $\boldsymbol{a}_j^{pressure} \leftarrow \boldsymbol{a}_j^{pressure} + \boldsymbol{p}_{\text{temp}}$
    $\boldsymbol{\mu}_{\text{temp}} \leftarrow \mu m\dfrac{\boldsymbol{v}_j - \boldsymbol{v}_i}{\rho_i\rho_j}\nabla^2 W^{2D}_{viscosity}(\boldsymbol{r}_{ji}, h)$
    $\boldsymbol{a}_i^{viscosity} \leftarrow \boldsymbol{a}_i^{viscosity} + \boldsymbol{\mu}_{\text{temp}}$
    $\boldsymbol{a}_j^{viscosity} \leftarrow \boldsymbol{a}_j^{viscosity} - \boldsymbol{\mu}_{\text{temp}}$
  end for
  $\boldsymbol{a}_i \leftarrow \boldsymbol{a}_i^{pressure} + \boldsymbol{a}_i^{viscosity} + \boldsymbol{a}_i^{external}$
end for

Note that if uniform mass can be assumed then it would be beneficial to exclude it from the density calculations and make the appropriate changes where the mass would cancel. However, if this approach is used, the value of density is no longer the actual particles density but must, of course, be scaled by mass.

4.4.2 The Clavet Method

The method presented in Clavet et al. [2005] is no doubt the most involved method. Values have been introduced without any clear physical connection. For example, they use an additional density value which they call near density. They give no physical justification, but claim that this value would remedy clustering of particles while maintaining a relatively low time-step. They have chosen the path of cool visual effect rather than physical accuracy.

Important to note, however, is that some physical laws such as the conservation of linear and angular momentum are upheld.

The modified method is a reduction of their complete work. They use springs and extra functions for viscosity all of which is unnecessary to simulate water. Surface tension emerges from their core method and it can be tuned to give rather viscous flow too. Therefore, only the *double density relaxation* algorithm from their paper as well as their method of integration and collision handling is implemented.

Algorithm 4.2 updates all particles to the next time-step. It has some similarities to the base Sph method, in Algorithm 3.1 but adds the calculations of near density and near pressure. It also updates the predicted position of the particles instead of calculating an acceleration contribution. This position together with the position from one time-step back is then used to calculate the velocity. This process is referred to as *Prediction-relaxation scheme* in Clavet et al. [2005].

Algorithm 4.2 *Updates the particles from time $t-1$ to t with the Clavet method.*

for all particles i **do**
  $\boldsymbol{v}_i^{(t)} \leftarrow \boldsymbol{v}_i^{(t-1)} + \boldsymbol{g}\Delta t$
  $\boldsymbol{x}_i^{(t)} \leftarrow \boldsymbol{x}_i^{(t-1)} + \boldsymbol{v}_i^{(t)}\Delta t$
  $\rho_i \leftarrow 0$
  $\rho_i^{near} \leftarrow 0$
end for
for all particles i **do**
  for all neighbors j of i **do**
    $\rho_i \leftarrow \rho_i + W^{clavet}(\boldsymbol{r}_{ji}, h)$
    $\rho_i^{near} \leftarrow \rho_i^{near} + W^{near}(\boldsymbol{r}_{ji}, h)$
  end for
  $p_i \leftarrow k(\rho_i - \rho_0)$
  $p_i^{near} \leftarrow k^{\text{near}}\rho_i^{near}$
  $\boldsymbol{\Delta x} \leftarrow \boldsymbol{0}$
  for all neighbors j of i **do**
    $\boldsymbol{d} \leftarrow \Delta t^2(p_i \nabla W^{clavet}(\boldsymbol{r}_{ji}, h) + p_i^{near}\nabla W^{near}(\boldsymbol{r}_{ji}, h))\hat{\boldsymbol{r}}$
    $\boldsymbol{x}_j^{(t)} \leftarrow \boldsymbol{x}_j^{(t)} + \frac{\boldsymbol{d}}{2}$
    $\boldsymbol{\Delta x} \leftarrow \boldsymbol{\Delta x} - \frac{\boldsymbol{d}}{2}$
  end for
  $\boldsymbol{x}_i^{(t)} \leftarrow \boldsymbol{x}_i^{(t)} + \boldsymbol{\Delta x}$
end for
Resolve collisions
for all particles i **do**
  $\boldsymbol{v}_i^{(t)} \leftarrow \frac{\boldsymbol{x}_i^{(t)} - \boldsymbol{x}_i^{(t-1)}}{\Delta t}$
end for

The relevant parameters presented in Clavet et al. [2005] are presented in Table 4.3.

It is hard to use some of these parameter as is. Some values are not specified such as gravity which makes one draw the conclusion that a value around 9.81 m/s^2 is used. However,

Table 4.3: *The relevant parameters presented in Clavet et al. [2005].*

Parameter	Value
Time step	1 (where the time unit is 1/30 second)
Reference density	10
k	0.004
k^{near}	0.01

experiments showed that a value of gravity around 0.001 must be used instead to give a reasonable stable simulation. Also, Clavet et al. [2005] claims that their parameters such as reference density and k are pre-scaled with the normalization constant of the used kernel. However, the parameters can be recomputed by extracting the normalization constant. For the interested reader, this computation is done in Appendix A.2. The current implementation have recomputed the constant of gravity and the time-step *only* in order to work well with Box2D. However, the scalar of the spring constants is only internal and cancels and is therefore left out. The reference density is tricky. The standard reference density of 100 kg/m^2 is used when interacting with Box2D and when calculating the smoothing length. However, when used in the algorithm, another "reference density" is used, tuned at around the value above.

4.4.3 The Bodin Method

The idea of the final method is to constrain the density to the reference density in the same way that a body can be constrained to follow a curve. It might be easier to visualize a body constrained to a curve because it is inherently geometric, but mathematically, the formulations are basically the same.

The constraint is formulated as

$$g = \frac{\rho}{\rho_0} - 1 = 0. \tag{4.1}$$

The formulation makes the density ρ, constrained to the reference density ρ_0. Note the similarities to Tait's equation of state, Eq. 3.23.

In the context of Sph, this corresponds to one constraint per particle, i.e. the i:th particle has constraint g_i. The density of the i:th particle is calculated with Eq. 3.19 restated here for convenience,

$$\rho_i = \sum_j m_j W(\boldsymbol{r}_{ij}, h). \tag{4.2}$$

This gives

$$g_i = \frac{\rho_i}{\rho_0} - 1 = \frac{1}{\rho_0} \sum_j m_j W(\boldsymbol{r}_{ij}, h) - 1. \tag{4.3}$$

Taking the time derivative (using the chain rule) of this yields the kinematic constraint[2],

$$\begin{aligned} \dot{g}_i &= \frac{d}{dt}\left(\frac{1}{\rho_0}\textstyle\sum_j m_j W(\boldsymbol{r}_{ij}, h) - 1\right) \\ &= \frac{1}{\rho_0}\textstyle\sum_j m_j \dfrac{\partial W(\boldsymbol{r}_{ij}, h)}{\partial \boldsymbol{r}_{ij}} \dfrac{\partial \boldsymbol{r}_{ij}}{\partial t} \\ &= \frac{1}{\rho_0}\textstyle\sum_j m_j \nabla W(\boldsymbol{r}_{ij}, h) \cdot \boldsymbol{v}_{ij} \end{aligned} \tag{4.4}$$

Following Eq. 2.36 the kinetic constraint is separated into the Jacobian times the velocity, i.e. $\dot{\boldsymbol{g}} = \boldsymbol{G}\boldsymbol{v}$. In order to separate the parts of $\dot{\boldsymbol{g}}$, first notice that

$$\begin{aligned} \dot{g}_i &= \frac{1}{\rho_0}\textstyle\sum_j m_j \nabla W(\boldsymbol{r}_{ij}, h)^T(\boldsymbol{v}_i - \boldsymbol{v}_j) \\ &= \left(\frac{1}{\rho_0}\textstyle\sum_j m_j \nabla W(\boldsymbol{r}_{ij}, h)^T\right)\boldsymbol{v}_i - \frac{1}{\rho_0}\textstyle\sum_j m_j \nabla W(\boldsymbol{r}_{ij}, h)^T \boldsymbol{v}_j \end{aligned} \tag{4.5}$$

This means that the diagonal of the Jacobian is the term multiplying $\boldsymbol{v}_i$ i.e.

$$\boldsymbol{G}_{ii} = \frac{1}{\rho_0} \sum_j m_j \nabla W(\boldsymbol{r}_{ij}, h)^T \tag{4.6}$$

and the rest of the Jacobian is the term multiplying $\boldsymbol{v}_j$ i.e.

$$\boldsymbol{G}_{ij} = -\frac{1}{\rho_0} m_j \nabla W(\boldsymbol{r}_{ij}, h)^T \tag{4.7}$$

where $j \neq i$. In this thesis, the poly6 kernel in 2D (W^{2D}_{poly6}) is used as W when calculating the constraint. However, a slight modification of its gradient is used in the same spirit as Nilsson [2009]. The notation for this is ∇W^{2D}_{poly6h}. Notice the h in the name, this is because $\nabla W^{2D}_{poly6h} = h\nabla W^{2D}_{poly6}$. These are explained in Appendix A.5.

Algorithm 4.3 is paraphrased from *Algorithm 7.2.4 - Narrow phase kernel*, [Nilsson, 2009] and is used to calculate the densities, the Jacobian, the constraint violations as well as the diagonal elements of the Schur complement matrix ($\boldsymbol{G}\boldsymbol{M}^{-1}\boldsymbol{G}^T + \boldsymbol{\Sigma}$) from Eq. 2.40.

[2] In Bodin et al. [2011] they separate the scalar part from $\hat{\boldsymbol{r}}$ and write the constraint as $\dot{g}_i = \frac{1}{\rho_0}\sum_j m_j f_{ij} \boldsymbol{r}_{ij}^T \boldsymbol{v}_{ij}$.

Algorithm 4.3 *Calculation of the densities, the Jacobian, the constraint violations as well as the diagonal elements of the Schur complement matrix* ($\boldsymbol{G}\boldsymbol{M}^{-1}\boldsymbol{G}^T + \boldsymbol{\Sigma}$) *from Eq. 2.40. Paraphrased from* Algorithm 7.2.4 - Narrow phase kernel, *[Nilsson, 2009]*

for all particles i **do**
  $\boldsymbol{d} \leftarrow \boldsymbol{0}$
  $\Phi_i \leftarrow 0$
  $\rho_i \leftarrow m_i W^{2D}_{poly6}(\boldsymbol{0}, h)$
  for all neighbors j of i $(j \neq i)$ **do**
    $r^2 \leftarrow \boldsymbol{r}_{ij}^T \boldsymbol{r}_{ij}$
    if $r^2 < h^2$ **then**
      $\rho_i \leftarrow \rho_i + m_j W^{2D}_{poly6}(\boldsymbol{r}_{ij}, h)$ // Eq. 3.19.
      $\boldsymbol{b} \leftarrow -\frac{m_j}{\rho_0} \nabla W^{2D}_{poly6h}(\boldsymbol{r}_{ij}, h)^T$ // Calculate the off-diagonal, $\boldsymbol{G}_{ij}$. Eq. 4.7.
      $\boldsymbol{d} \leftarrow \boldsymbol{d} - \boldsymbol{b}$ // Eq. 4.6.
      $\Phi_i \leftarrow \Phi_i + \boldsymbol{b}^T \boldsymbol{b}$
      $\boldsymbol{G}_{ij} \leftarrow \boldsymbol{b}$
    end if
  end for
  for all rigid neighbors k of i **do**
    $r^2 \leftarrow \boldsymbol{r}_{ik}^T \boldsymbol{r}_{ik}$
    if $r^2 < h^2$ **then**
      $\rho_i \leftarrow \rho_i + m_i W^{2D}_{poly6}(\boldsymbol{r}_{ik}, h)$ // Density field boundary condition.
      $\boldsymbol{b} \leftarrow -\frac{m_i}{\rho_0} \nabla W^{2D}_{poly6h}(\boldsymbol{r}_{ik}, h)^T$
      $\boldsymbol{d} \leftarrow \boldsymbol{d} - \boldsymbol{b}$
      $\Phi_i \leftarrow \Phi_i + \boldsymbol{b}^T \boldsymbol{b}$
      $\boldsymbol{G}^{rigid}_{ik} \leftarrow \boldsymbol{b}$
    end if
  end for
  $\boldsymbol{G}_{ii} \leftarrow \boldsymbol{d}$ // Set the summed elements as the diagonal.
  if $\Phi_i \neq 0$ **then**
    $\Phi_i \leftarrow \frac{\Phi_i + \boldsymbol{d}^T \boldsymbol{d}}{m_i} + \Sigma_i$
    $\Phi_i \leftarrow \frac{1}{\Phi_i}$
  else
    $\Phi_i \leftarrow \infty$
  end if
  $g_i \leftarrow \frac{\rho_i}{\rho_0} - 1$
end for

Box2d solves the movement of the rigid bodies *separate* from the fluid implementation. Because of this, the implementation separates the parts of the Jacobian that corresponds to particles from the part corresponding to rigid bodies. This last part is called $\boldsymbol{G}^{rigid}$, i.e. the interaction between particle i and rigid body k is store in $\boldsymbol{G}_{ik}^{rigid}$. Solutions such as this is expected when integration between two separate systems should be made.

The initialization of Gauss-Seidel is shown in Algorithm 4.4 while the actual iterations are found in Algorithm 4.5. Here $\boldsymbol{c}$ is introduced as

$$\boldsymbol{c} = -a\boldsymbol{g} + \boldsymbol{\Xi}\boldsymbol{G}\boldsymbol{v}, \tag{4.8}$$

where a is found in Eq. 2.45, while the rest is as defined in Eq. 2.40.

Algorithm 4.4 *The initialization of the Gauss-Seidel iteration solver to solve Eq. 2.40. See* Algorithm VII.1*, [Bodin et al., 2011]*

for all particles i **do**
  $\boldsymbol{\lambda}_i \leftarrow 0$
  $\alpha \leftarrow \boldsymbol{G}_{ii}\boldsymbol{v}_i$
  for all neighbors j of i $(j \neq i)$ **do**
    $\alpha \leftarrow \alpha + \boldsymbol{G}_{ij}\boldsymbol{v}_j$
  end for
  for all rigid neighbors k of i **do**
    // Get the velocity of the rigid body at the collision point, in Box2d:
    $\boldsymbol{v}_k^{rigid} \leftarrow$ `bodies[k].GetLinearVelocityFromWorldPoint(collisionPoint);`
    $\alpha \leftarrow \alpha + \boldsymbol{G}_{ik}^{rigid}\boldsymbol{v}_k^{rigid}$
  end for
  $\boldsymbol{c}_i \leftarrow -a\boldsymbol{g}_i + \boldsymbol{\Xi}_{ii}\alpha$
end for

Algorithm 4.5 *The Gauss-Seidel iteration solver to solve Eq. 2.40. See* Algorithm VII.1*, [Bodin et al., 2011]*

repeat
  for all particles i **do**
    // Φ_i is defined in Algorithm 4.3
    if $\Phi_i \neq \infty$ **then**
      $\boldsymbol{r}_i \leftarrow -\boldsymbol{c}_i + e\boldsymbol{\lambda}_i$ // e is defined in Eq. 2.47

      for all neighbors j of i **do**
        $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \boldsymbol{G}_{ij}\boldsymbol{v}_j$
      end for
      for all rigid neighbors k of i **do**
        // Get the velocity of the rigid body at the collision point, in Box2D:
        $\boldsymbol{v}_k^{rigid} \leftarrow$ `bodies[k].GetLinearVelocityFromWorldPoint(collisionPoint);`
        $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \boldsymbol{G}_{ik}^{rigid}\boldsymbol{v}_k^{rigid}$
      end for

      $\Delta\boldsymbol{\lambda}_i \leftarrow -\boldsymbol{r}_i\Phi_i$
      $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i + \Delta\boldsymbol{\lambda}_i$

      for all neighbors j of i **do**
        $\boldsymbol{v}_j \leftarrow \boldsymbol{v}_j + \boldsymbol{G}_{ij}m_j^{-1}\Delta\boldsymbol{\lambda}_i$
      end for
      for all rigid neighbors k of i **do**
        $I \leftarrow \boldsymbol{G}_{ik}^{rigid}\Delta\boldsymbol{\lambda}_i$
        // Apply linear impulse I at the collision point of the rigid body.
        // In Box2D this is written:
        `bodies[k].ApplyLinearImpulse(I, collisionPoint);`
      end for
    end if
  end for
until Residual is small or iterations exceed max

Chapter 5

Implementation

Box2d is a small 2d mechanics engine for rigid bodies. However, it is still a complex piece of software. For a complete manual, see Catto [2010]. However, a short introduction and summary is presented below.

Box2d has a world filled with bodies. Each of these bodies have a number of fixtures of different shapes. For simplicity, only one fixture and shape is used for all bodies in this application. The world object keeps track of a list of bodies. These are updated each frame, and moved accordingly. Bodies that do not move are put to sleep, and non-contacting sets of contacting bodies are solved independently in what is called an island. The contacts are solved in a similar manner as spook, i.e. with constraints and an iterative solver.

The contribution of this thesis adds a list of particle systems to the world class, in a similar manner as bodies. These are updated by the world but not seen by any other entity in Box2d. As such, it is not part of any island. The emitter on the other hand is a full-blown body. It can be moved in the same manner as all the other bodies. However, the emission is still active even though the body might be asleep.

Box2d supports Aabb *queries* or *region queries*, i.e. all fixtures within a given rectangle can be fetched. This is exploited when the fluid needs to find its boundaries. Therefore, an Aabb is fitted around the fluid at each time-step. Then, this Aabb is sent to the world together with a callback which will be called with every fixture in the world that intersects the given Aabb. Each fixture is then tested against the fluid.

This is done with a method inspired by distance maps, however, it only stores pointers to the closest edge without the distance. It also does not store the middle part of the fixture, if it is large. This map is created each time a *new* fixture is found to be within the fluid Aabb and stores it in the fixture, i.e. the map is only created *at most once* for each fixture. However, it is created in body coordinates and must therefore be transformed to world coordinates each frame that it intersects the fluid. Each particle in the fluid is first tested against the fixtures Aabb and then tested against the map. This latter test will result in the closest edge or no edge if the particle is too far away. If an edge is sufficiently close, the appropriate boundary condition is applied depending on simulated method.

Chapter 6

Experiments

A classic benchmark test for SPH methods is the "dam break" simulation. In this benchmark, a pillar of water is constructed. Then, one wall is removed letting the fluid flow and splash around. This, to test performance and how well it behaves. If nothing else is mentioned, the parameters in Table 6.1 are used for the Müller method and the Bodin method while the parameters in Table 6.2 are used for the Clavet method.

6.1 Experiment 1 - Theoretical Correctness

The purpose of the first experiment is to test how well the methods can simulate water. This experiment is divided into two parts, to test incompressibility and energy. These are carried out by constructing a pillar of water filled with a fixed number of particles. 1000 particles for the Incompressibility Test and 1000 as well as 2000 particles for the Stability and Energy Test. The width of the pillar is 3 meters for the Incompressibility Test and 4 meters for the Stability and Energy Test. During the simulation, relevant parameters are extracted and plotted.

Table 6.1: *The parameters used in the Müller method and the Clavet method.*

Parameters	value
Nr particles	1000
Mass	2 kg
Gravity	9.82 m / s^2
SCP	0.65
Time step Δt	1 / 120 s
Reference density ρ_0	100 kg / m^2
Smoothing length h	$2\sqrt{\frac{2}{\pi 100}} \approx 0.16$ m

Table 6.2: *The parameters used in the Clavet method with hybrid kernels.*

Parameters	value
Nr particles	1000
Mass	-
Gravity	9.82 m / s^2
Time step Δt	1 / 120 s
Reference density ρ_0	5
k	0.0004
k^{near}	0.05
Smoothing length h	0.2 m

6.2 Experiment 1.1 - The Incompressibility Test

Water is almost incompressible. The quality of a water simulation should therefore depend on the ability to uphold the incompressibility assumption. Therefore, a pillar of water is constructed and filled so that the mass density of the particles can be plotted as a function of depth.

Also, the self contribution percentage is varied to show the effect that the smoothing length has on compressibility. Finally, one configuration from each method is chosen. The configurations are weigh-offs between incompressibility, number of neighbors and visual artifacts. These are then run using half the original time-step, i.e. $\Delta t = 1/240$ s. This, in order to see the effect that the time-step has on incompressibility.

6.2.1 The Müller Method

When the self contribution percentage (Scp) is varied, the compressibility of the fluid is affected. Therefore the speed of sound is maximized, i.e. set to the highest value that still gives a sufficiently stable simulation, for each value of Scp. The chosen configuration for testing the effect of time-step, is to fix Scp at 65 %.

6.2.2 The Clavet Method

The method based on Clavet et al. [2005] is more difficult to compare and evaluate. The parameters do not have clear consequences, for example, increasing k also increase the believed viscosity. Also, changing the reference density has a direct impact on viscosity. This, because the reference density only have an effect on the normal density, *not* the "near" density and therefore affects the simulation in a similar way to k. However, the equations have been modified by extracting the kernel normalization constants and by scaling constant parameters (see Appendix A.2). Still, this does little to ameliorate the problem of choosing suitable parameters, and does nothing about the discrete jump between near and normal density.

Also, the discussion concerning self contribution percentage in this context is moot. Which density should equal Scp $\cdot\rho_0$? Possibly the sum or maybe only the normal density because it is the only density which is affected. Whichever choice is made, the outcome is still not in accordance to the previous discussions because the near pressure calculation will exert a force that keeps two particles from achieving reference density. This means that a choice of 100 kg / m^2 could result in an average density much less than 100, e.g. 25 kg / m^2. Because of this, another approach is taken here. The actual value of the reference density is ignored and simply set to a value which gives visually good results. Then the density is measured as a function of depth. The difference between 100 and the minimum density measurement is added to the resulting density so that the minimum measured density is 100 kg / m^2, i.e. the plot is "moved" up so that it starts at 100 kg / m^2. Also, the spring constant k^{near} is set to a maximum value while maintaining a visually good simulation.

During experimentation, different kernels were tested. In the original method, *distance*, not squared distance, is used, i.e. the original method involves an expensive square root. Therefore, a kernel using squared distance was tried instead, i.e.

$$1 - \frac{|\boldsymbol{r}|^2}{h^2} \quad \text{was used instead of} \quad 1 - \frac{|\boldsymbol{r}|}{h}. \tag{6.1}$$

This is referred to as *Squared kernel* throughout this thesis and a full specification is found in Appendix A.4. Also a hybrid version, using the squared core in Eq. 6.1 but with the normalization of $\boldsymbol{r}$ kept from the original kernel gradient. This is henceforth called *Hybrid kernel*. Using the method by Clavet et al. [2005] with another kernel has also been tried by Madams [2010] who reported an increase in incompressibility. The chosen configuration for testing the effect of time-step, is to use the hybrid kernel.

6.2.3 The Bodin Method

The Bodin method does not have any "context based" parameters, i.e. parameters that need to be changed because of Scp. Therefore, the parameters in Table 6.1 are used for all simulations in the Incompressibility Test. The chosen configuration for testing the effect of time-step, is to fix Scp at 65 %.

6.3 Experiment 1.2 - The Stability and Energy Test

Stability is extremely important in gaming environments. Without stability, it would not matter how fast something can be simulated. Therefore, the kinetic energy of the water in the pillar is measured and plotted as visual aid, in order to show the convergence from a moving to a resting state.

In Müller and Clavet methods, the different spring constants governing pressure was maximized when simulating the pillar of water. This, by definition, leaves the method on the verge of instability. If the fluid is subject to high external pressure, e.g. rigid bodies falling in or the fluid falls towards the ground, the simulation is likely destroyed. Also, these spring constants must be changed each time the initial condition of the simulation changes, e.g. the number of particles is changed, in order to maintain incompressibility. The Bodin method on the other hand do not have pressure related spring constants. In fact, many parameters e.g. number of particles and mass can be changed without affecting the stability.

This experiment tries to shed light on these issues by measuring the energy in the simulation and how long it takes for a pillar of water to settle. The same cases used in Section 6.2 are used here, except that the pillar is now four meters wide and 2000 particles are *also* tested except the case with 1000 particles.

The time it takes for the particles to reach a resting state is dependant on *how* the fluid particles are *placed* in the pillar. If they are placed in a way such that all forces are symmetric and equal, the particles would immediately be at rest. In all experiments, the particles are placed in a grid pattered with Scp times the smoothing length distance units apart. This will give the particles a rather high initial kinetic energy, which is needed in order to see how fast they converge to a resting state. However, because the way they are placed matters and the way that they handle Scp differs, the different times the methods take to converge should only be compared with caution.

6.4 Experiment 2 - The Performance Test

Performance is of great importance in methods for game environments. Often more important than accuracy. Therefore, the performance will be measured during a dam-break simulation *and* a pillar of water simulation for 500, 1000, 1500 and 2000 particles. The simulations are run over 10000 updates, or $10000\Delta t \approx 167$ s with 60 Hz or around 83 s with 120 Hz. The average times per update, in milliseconds, is then reported.

Because the methods all use the same collision detection, data storage and utility tools the measured times should mirror the theoretical time consumption of each method. Also, the structures of the methods are similar. However, the results should still be taken lightly.

The choice of parameters for the different methods will affect performance in a number of different ways, but each set of parameters is chosen to give a stable simulation free from artifacts.

The performance of the Müller method and the Clavet method is not *directly* affected by their parameters. Of course, if the spring constants are reduced they will have more neighbors and performance takes a hit. However, there is no parameter that gives an improved simulation while having a negative effect on performance. However, the Bodin method can increase the number of iterations of the solver in order to minimized the error. Bodin et al. [2011] uses 5–15 iterations while this implementation uses 15 iterations at all times. Therefore, the experiment is repeated with 10 iterations in order to see the effect on performance.

6.5 Experiment 3 - The Rigid Body Test

In a game environment, the fluid will come in contact with moving rigid objects. The movement of these should be affected as well as affect the fluid, i.e. they should be *two-way coupled*. To test the implementations, two scenarios are created. First, five rigid bodies of different density will be dropped in the fluid, and the behavior is recorded. Secondly, fluid will be poured onto the same rigid bodies. In a real fluid, Archimedes's principle tells that a body of lower density than the fluid will float and bodies of higher density will sink.

Chapter 7

Results

This chapter will present the results of the experiments listed in Chapter 6.

7.1 Experiment 1.1 - The Incompressibility Test

7.1.1 The Müller Method

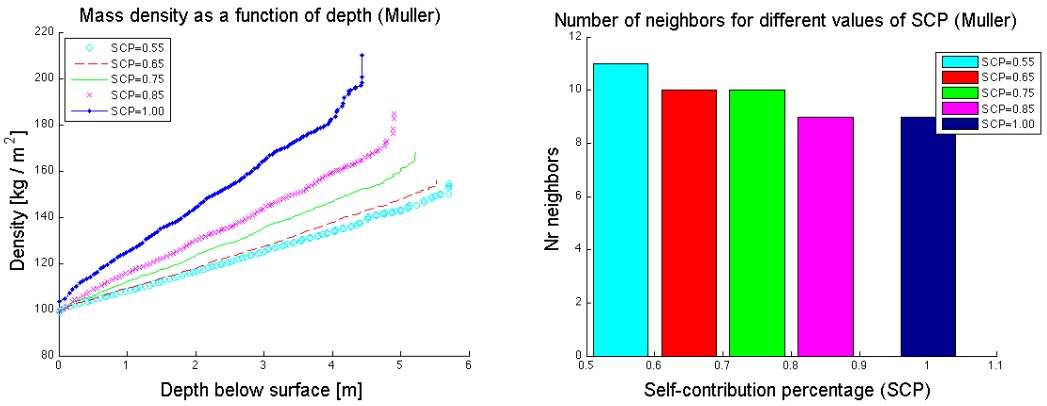
The mass density as a function of depth is plotted in Fig. 7.1a. Note that higher self-contribution percentage (Scp), i.e. the parameter related to smoothing length, has an effect on the compressibility of this method. If no attraction between particle is used, i.e. a lone particle's density is equal to the reference density, (Scp = 1), then the fluid is highly compressible. The total depth of the fluid is around 4 meters as compared to 5.5 meters when using Scp $\approx$ 0.6. At first glance, the graph suggests that the fluid would get less compressible if Scp decreased. However, note that Scp = 0.55 gives roughly the same level of compressibility as Scp = 0.65.

Fig. 7.1a also shows that mass density increases roughly linear with depth. It means that only the surface of the fluid has the correct density, and in this case the surface is only a small fraction of the total number of particles. Also note that the density has a sharp increase in density towards the bottom of the pillar, i.e. near the boundary. The boundary technique for this method is a projection technique, i.e. it *lifts* the particles up from the boundary.

Another important point to note is that as Scp is *decreased* the number of neighbors *increases*, see Fig 7.1b. Fig. 7.2 shows the difference in compressibility when halving the time-step.

The visual appearance of the simulation shows that large values of Scp gives the simulation a greater similarity to sand, while a low value introduces visible artifacts and jitters.

Fig. 7.2 shows the simulation with two time-steps, 1 / 120 s and 1 / 240 s, in order to show



(a) (b)

Figure 7.1: *Pillar of water experiment for the Müller method. (a) Shows the mass density as a function of depth for different values of* Scp*. (b) Shows the maximum number of neighbors in the pillar for different values of* Scp*. A given value of* Scp *has the same color in both figures.*

the effect of time-step on incompressibility. Because Scp = 65 % gave roughly the same incompressibility as Scp = 55 % but managed to keep the number of neighbors lower, the value of Scp = 65 % is used in this time-step test. Note, in Fig. 7.2, the massive decrease in compressibility when using the halved time-step.

7.1.2 The Clavet Method

When executing this experiment, the spring constant k^{near} is set to a maximum value while maintaining a visually good simulation. However, the height of the pillar causes great pressure which in turn causes some artifacts to emerge. The particles at the bottom tend to move, almost flow, as if the fluid is *boiling*.

The Hybrid and Squared kernels gave much more stability which allowed an increase of k^{near}, which, in turn, lead to an increase of incompressibility. Fig. 7.3 shows a comparison between the three kernels.

The number of neighbors are roughly the same, but reduced slightly for the Squared kernels, see Fig. 7.3b. Both new kernels managed to reduced the "boiling"-artifact but they did not remove it completely. The artifact was severely reduced when using the Squared kernels.

When using the Squared kernels the visual appearance of the simulation looks like a hybrid between water and sand. Note that this is basically the same phenomenon that arise when using Scp equal to 1 in Müller and Bodin methods. The simulated fluid might not flatten out as it should but keep small piles of particles. Fig. 7.4 shows the difference in compressibility when halving the time-step.

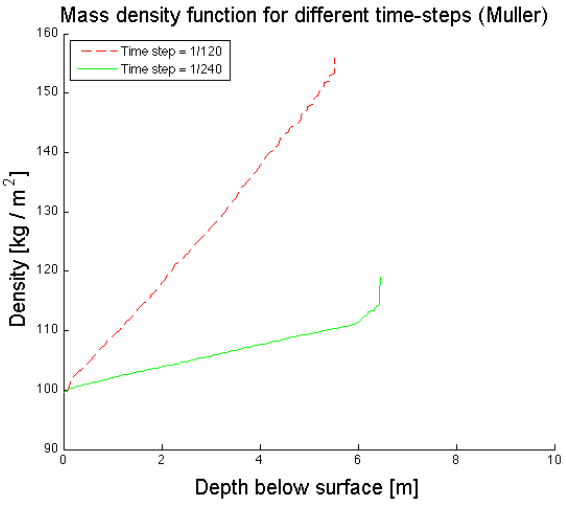
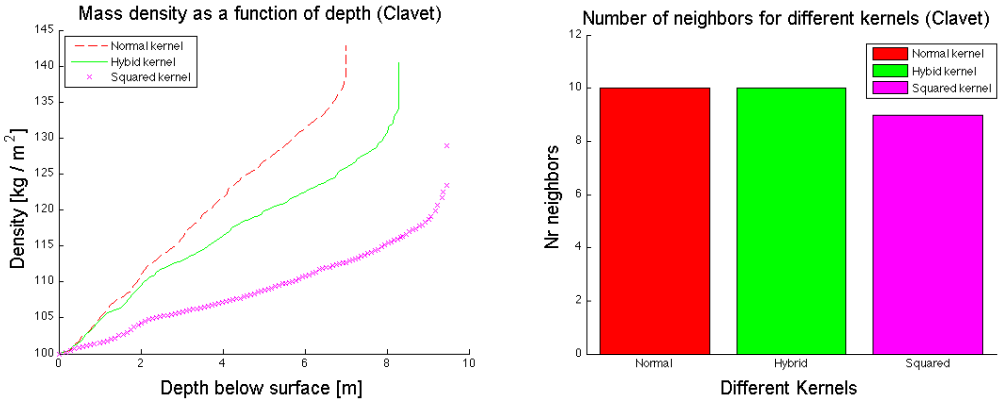


Figure 7.2: *The Müller method simulated in a pillar of water experiment. A comparison is then made between two chosen configurations, one with time-step 1 / 120 s, (red dashed line), and one with 1 / 240 s (green filled line). Note the massive decrease in compressibility when using a time-step of 1 / 240 s.*



(a) (b)

Figure 7.3: *Pillar of water experiment for the Clavet method. (a) Shows the mass density as a function of depth when using the normal kernels (red dashed line), hybrid kernels (green filled line) and squared kernels (purple crosses). (b) Shows the maximum number of neighbors in the pillar when using the normal kernels versus the squared kernels. A given choice of kernels has the same color in both figures.*

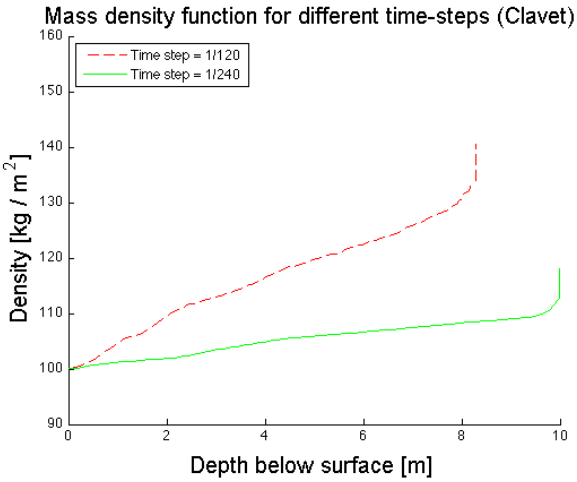
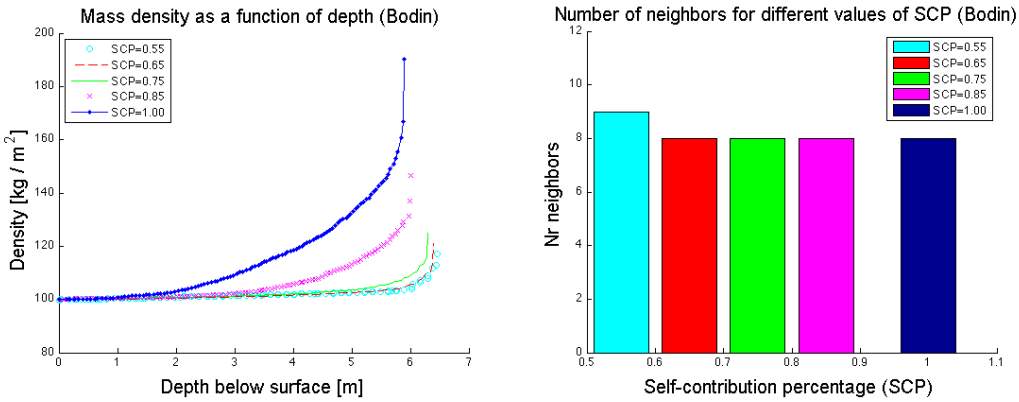


Figure 7.4: *The Clavet method simulated in a pillar of water experiment. A comparison is then made between two chosen configurations, one with time-step 1 / 120 s, (red dashed line), and one with 1 / 240 s (green filled line).*



(a) (b)

Figure 7.5: *Pillar of water experiment for the Bodin method. (a) Shows the mass density as a function of depth for different values of* Scp*. (b) Shows the maximum number of neighbors in the pillar for different values of* Scp*. A given value of* Scp *has the same color in both figures.*

7.1.3 The Bodin Method

For the Bodin method, the mass density as a function of depth is plotted in Fig. 7.5a. Note the non-linearity between mass density and depth. As seen in Fig. 7.5a, this depth is increased as Scp decreases giving the lowest compressibility around Scp = 0.65. Below this, not much happens with the incompressibility.

Fig. 7.5b shows that increased smoothing length does not give a much larger neighborhood. In fact, the number of neighbors is close to constant. Fig. 7.6 shows the difference in compressibility when halving the time-step.

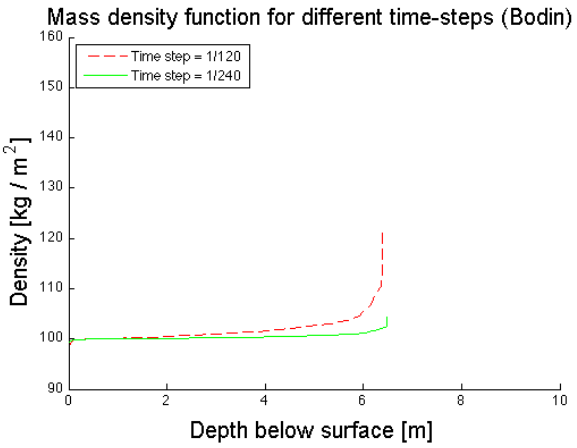
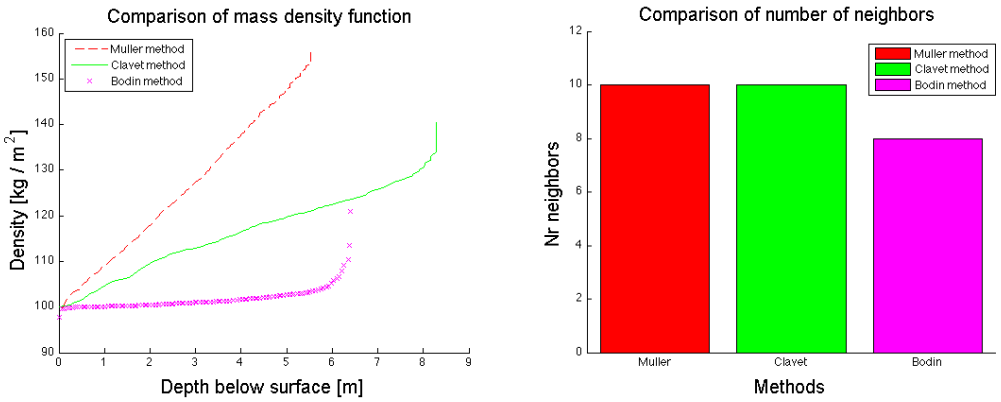


Figure 7.6: *The Bodin method simulated in a pillar of water experiment. A comparison is then made between two chosen configurations, one with time-step 1 / 120 s, (red dashed line), and one with 1 / 240 s (green filled line).*



(a) (b)

Figure 7.7: *A comparison between one chosen configuration from all three methods in a pillar of water experiment. (a) The mass density as a function of depth for the different methods. Dashed red line is the Müller method. Filled green line is the Clavet method and purple crosses for the Bodin method. (b) The maximum number of neighbors in the pillar for the different methods. A given method has the same color in both figures.*

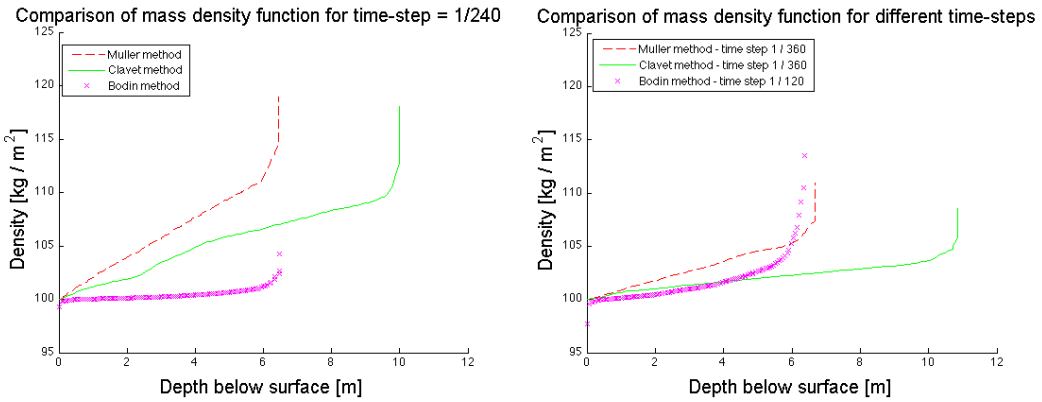
7.1.4 Comparison

The methods are compared in Fig. 7.7a where the density as a function of depth is shown using only *one* version of the respective methods. Take heed, though, not to compare the resulting depth of the Clavet method with the others, only look at how the density is distributed depending on depth.

The maximum number of neighbors, found at the bottom of the pillar for the different methods are shown in Fig. 7.7b. This result mirrors the result in Fig. 7.7a because the number of neighbors is directly related to the amount of compression.

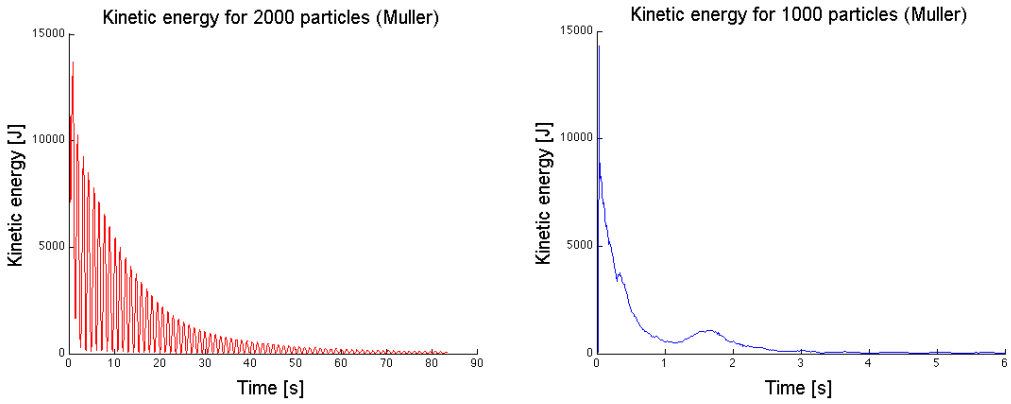
All methods are compared with itself using two time-steps in Fig. 7.2, 7.4 and 7.6. Also, the method are compared with each other when using the halved time-step, see Fig. 7.8a. Note, that the density error decreases giving a more incompressible fluid. Also, the Müller and Clavet methods increases notably in volume.

If all methods should be tuned to give roughly the same incompressibility, the time-step could be varied individually for each method. In Fig. 7.8b the time-step is reduced to 1/360 s for the Müller and Clavet methods while the original time-step of 1/120 s is used for the Bodin method.



(a) (b)

Figure 7.8: *All three methods are simulated in a pillar of water experiment. (a) The three methods are compared when using a time-step equal to 1 / 240 s. (b) The three methods are compared when using different time-steps, in order to achieve roughly the same incompressibility. Here, the Müller and Clavet methods uses a time-step equal to 1 / 360 s while the Bodin method uses 1 / 120 s.*



(a) (b)

Figure 7.9: *The kinetic energy of a fluid in a pillar-of-water experiment with 2000 particles in (a), and 1000 in (b), simulated with the Müller method. (a) The first 90 seconds of simulation. The kinetic energy is reduced in an exponential fashion, oscillating between high and low kinetic energy. (b) The first 6 seconds of simulation. The kinetic energy is reduced rather fast.*

7.2 Experiment 1.2 - The Stability and Energy Test

7.2.1 The Müller Method

First, the kinetic energy of the fluid is measured in a pillar of water with 2000 particles using the Müller method. The result of this is plotted in Fig. 7.9a.

From this figure, notice that it takes a *very* long time for the fluid to converge to rest. In Fig. 7.10, the oscillations in kinetic energy is clear. Note, that the period is roughly 1.2 seconds but the period of visual oscillation is twice that, i.e. 2.4 seconds. This, because the fluid in the pillar is at temporary rest, i.e. kinetic energy is zero, at *two* locations, both high and low.

Fig. 7.9b show the result when repeating the exact same experiments but reducing the number of particles to 1000. When doing this, the time of convergence is heavily reduced.

7.2.2 The Clavet Method

The kinetic energy of the fluid is measured in a pillar of water with 2000 particles using the Clavet method. The result of this is plotted in Fig. 7.11.

These figures, clearly indicate that the Clavet method is not guaranteed to converge. The movement of the fluid after the initial time, is not of oscillatory nature, but a kind of flowing near the bottom of the pillar. This is referred to as the ”boiling”-artifact in Section 7.1.2.

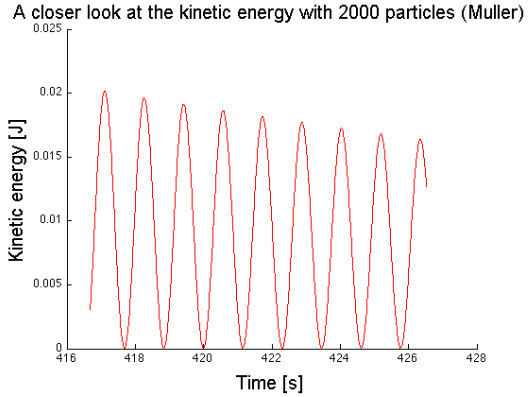
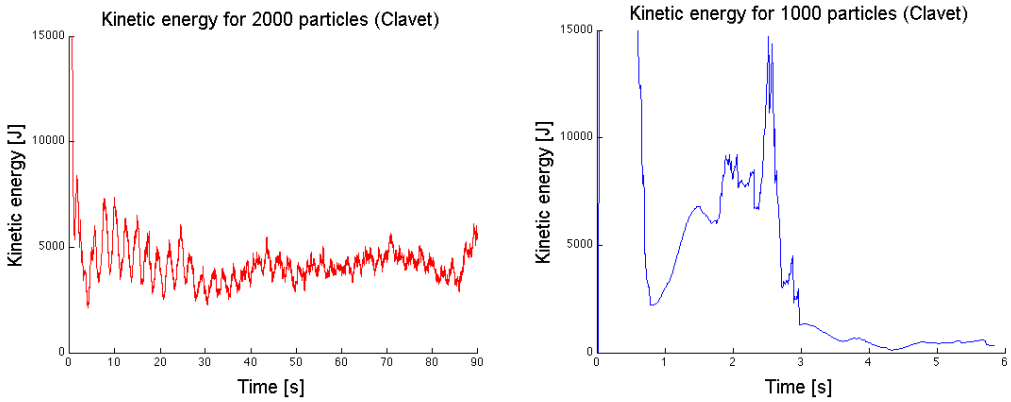
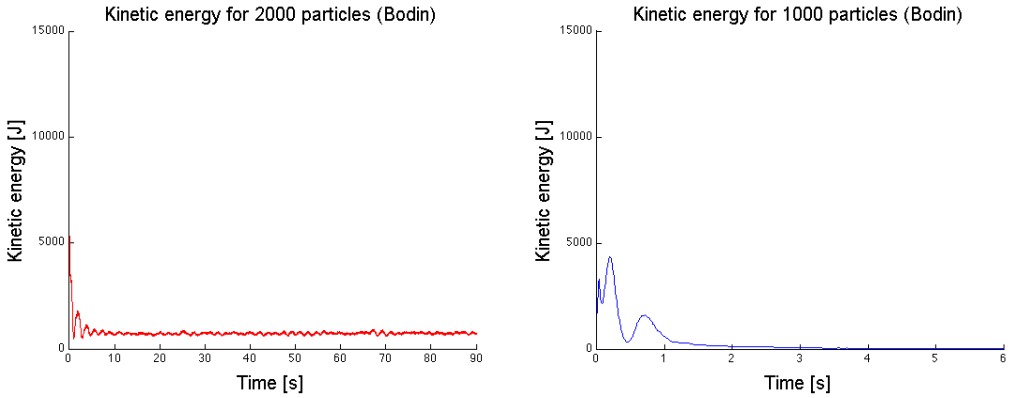


Figure 7.10: *A slice from Fig. 7.9a has been magnified in order to visualize the behavior of the fluid while it converges. Note that the period oscillation is roughly 1.2 seconds.*



(a) (b)

Figure 7.11: *The kinetic energy of a fluid in a pillar of water experiment with 2000 particles in (a) and 1000 particles in (b), simulated with the Clavet method. (a) The first 90 seconds of simulation. The kinetic energy is rapidly reduced in the beginning but it oscillates rapidly without convergence. (b) The first 6 seconds of simulation with 1000 particles. It seems to converge.*



(a) (b)

Figure 7.12: *The kinetic energy of a fluid in a pillar of water experiment with 2000 particles in (a) and 1000 particles in (b), simulated with the Bodin method. (a) The first 90 seconds of simulation. The kinetic energy is rapidly reduced in the beginning. Rough oscillations are then seen without convergence. (b) The first 6 seconds of simulation. The energy quickly stabilizes, without much visible oscillations.*

When simulating only a 1000 particles, this does not happen, and so it converges.

7.2.3 The Bodin Method

The kinetic energy of the fluid is measured in a pillar of water with 2000 particles using the Bodin method. The result of this is plotted in Fig. 7.12a.

As the Clavet method, this too is not guaranteed to converge. However, The movement of the fluid after the initial time is very different. This does not introduce flow (boiling) but vibrates around the boundary. Fig. 7.12a shows the kinetic energy which never seem to settle.

Fig. 7.12b shows the result when repeating the exact same experiments but reducing the number of particles to 1000. Here it quickly converges to low kinetic energy, without much visible oscillations in the graph. The visual result of the simulation show some oscillations. However, they sway from side to side, not down and up, much like water actually sway back and forth.

Table 7.1: *The time, in milliseconds, per physics update in a dam-break simulation using 500, 1000, 1500 and 2000 particles. Here, the Müller method and the Clavet method gives roughly the same execution time while the Bodin method takes a longer time.*

	500	1000	1500	2000
The Müller method	2.4	4.6	7.0	9.6
The Clavet method	2.4	4.7	7.3	9.9
The Bodin method	3.6	6.9	10.2	13.6

Table 7.2: *The time, in milliseconds, per physics update in a pillar simulation using 500, 1000, 1500 and 2000 particles. Here, the Müller method and the Clavet method gives roughly the same execution time while the Bodin method takes a little longer time.*

	500	1000	1500	2000
The Müller method	2.5	5.3	8.7	13.0
The Clavet method	2.7	5.4	8.5	11.9
The Bodin method	3.6	7.0	10.4	14.0

7.3 Experiment 2 - The Performance Test

The average times per update, in milliseconds, for the dam break is given in Table 7.1 and for the pillar simulation in Table 7.2. Note the large increase in execution-time between the dam-break simulation, Table 7.1 and the pillar simulation in Table 7.2 for the Müller and Clavet methods. This while the Bodin method reports only marginal increase in execution-time.

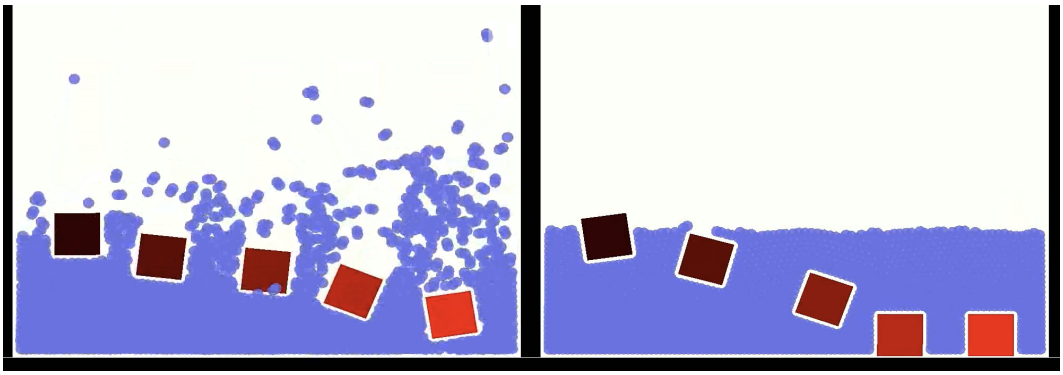
Repeating the Performance Test when *only* changing the number of iterations of the Bodin method to 10 gives the performance in Table 7.3 and 7.4. Note that the Bodin method has the lowest execution time for 2000 particles in Fig. 7.4.

Table 7.3: *The time, in milliseconds, per physics update in a dam-break simulation using 500, 1000, 1500 and 2000 particles. The number of iterations of the Bodin method is reduced.*

	500	1000	1500	2000
The Müller method	2.4	4.6	7.0	9.6
The Clavet method	2.4	4.7	7.3	9.9
The Bodin method	3.0	5.6	8.3	11.2

Table 7.4: *The time, in milliseconds, per physics update in a pillar simulation using 500, 1000, 1500 and 2000 particles. The number of iterations of the Bodin method is reduced.*

	500	1000	1500	2000
The Müller method	2.5	5.3	8.7	13.0
The Clavet method	2.7	5.4	8.5	11.9
The Bodin method	3.0	5.7	8.6	11.4

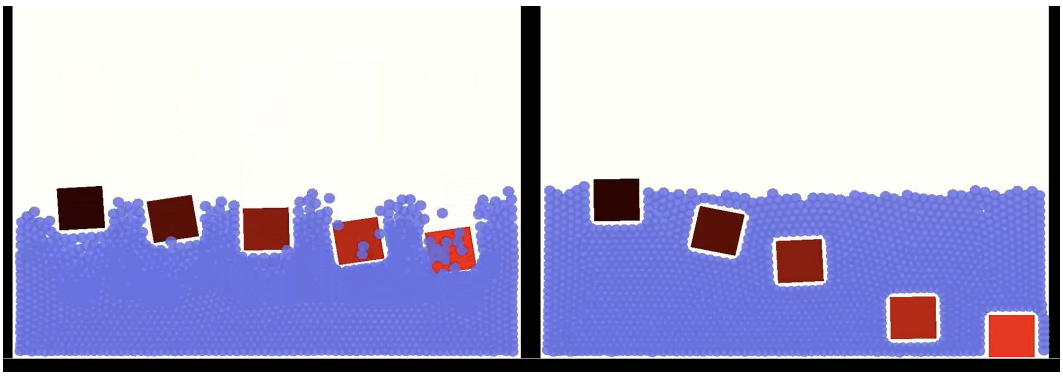


(a) Soon after the bodies has been dropped. (b) After the movement have converged.

Figure 7.13: *Five rigid bodies are dropped in fluid simulated with the Müller method. The higher the density of the body is, the brighter the color is. The darkest body has 50 kg / m^2, and the next darkest has 75 kg / m^2 and so on, in 25 kg / m^2 increments. Note that only the body with 100 / m^2 is neither on the bottom nor floating on the surface.*

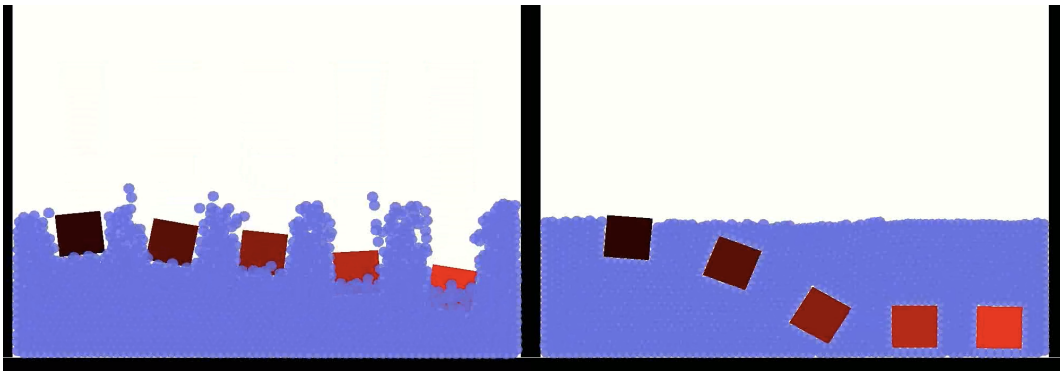
7.4 Experiment 3 - The Rigid Body Test

The results of the interaction test are difficult to measure. Therefore, the two scenarios are simulated and some screen-shots are shown here in order to give visual support for the discussion. These results are therefore more subjective than the others but note the importance of the figures showing the converged state of the interaction, usually shown to the right of an image pair.



(a) Soon after the bodies has been dropped. (b) After the movement have converged.

Figure 7.14: *Five rigid bodies are dropped in fluid simulated with the Clavet method. The higher the density of the body is, the brighter the color is. The darkest body has 50 kg / m^2, and the next darkest has 75 kg / m^2 and so on, in 25 kg / m^2 increments. Note that only the bodies with 150 and 50 kg / m^2 are on the bottom or floating on the surface.*



(a) Soon after the bodies has been dropped. (b) After the movement have converged.

Figure 7.15: *Five rigid bodies are dropped in fluid simulated with the Bodin method. The higher the density of the body is, the brighter the color is. The darkest body has 50 kg / m^2, and the next darkest has 75 kg / m^2 and so on, in 25 kg / m^2 increments. Note that only the body with 50 kg / m^2 is floating on the surface.*

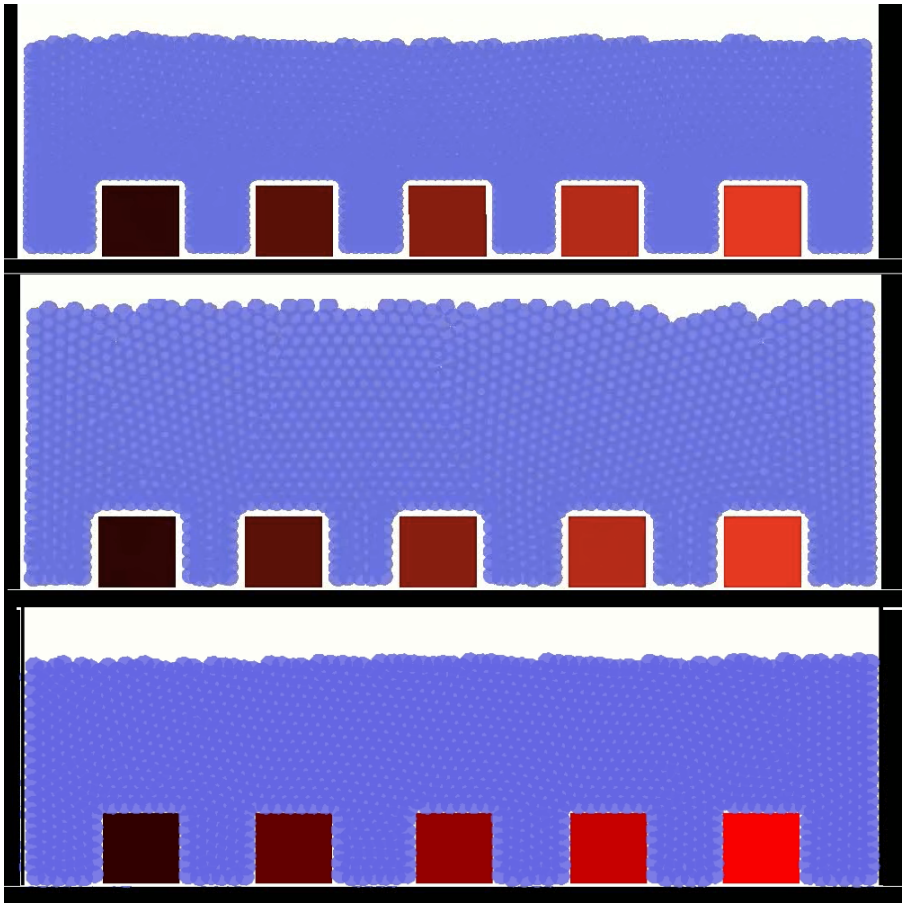


Figure 7.16: *Fluid, simulated with each method, the Müller method on top, the Clavet method in the middle and the Bodin method on the bottom, is dropped on five rigid bodies. The higher the density of the body is, the brighter the color is. The darkest body has 50 kg / m^2, and the next darkest has 75 kg / m^2 and so on, in 25 kg / m^2 increments. Note that all bodies in all cases stay at the bottom.*

Chapter 8

Discussion

This chapter discusses the results from Chapter 7.

8.1 Experiment 1.1 - The Incompressibility Test

8.1.1 The Müller Method

Consider the graph in Fig. 7.1. At first glance, the graph suggests that the fluid would get less compressible if Scp is decreased. However, note that Scp = 0.55 gives roughly the same level of compressibility as Scp = 0.65. This instead suggests that one cannot keep decreasing Scp to increase incompressibility.

Fig. 7.1a also shows that mass density increases roughly linear with depth. This is not good. It means that only the surface of the fluid has the correct density, and in this case the surface is only a small fraction of the total number of particles. The plot shows a sharp increase in density towards the bottom of the pillar. This is believed to be due to the boundary technique used, because at this point the particles experiences a non-regularity in the force exerted on them.

In these experiments, the speed of sound was varied. It is believed that this is what causes the increased level of incompressibility. This suggests that Scp "only" brings increased stability. However, similar results are achieved using the Bodin method, where a lower Scp gives increased incompressibility when *only* Scp is changed, i.e. independent of the speed of sound.

Another important point is to note that as Scp is *decreased* the number of neighbors *increases*, see Fig 7.1b. Therefore, it might be better to choose Scp = 0.65 than Scp = 0.55, for example.

8.1.2 The Clavet Method

The "boiling"-artifact suggest that the method is not suitable for simulating fluids subject to high pressure. However, note that the method in Clavet et al. [2005] uses explicit viscosity and springs which might alleviate the problem.

The Hybrid kernels as well as the Squared kernels gave much more stability which allowed an increase of k^{near}, which of course lead to an increase of incompressibility. The number of neighbors got reduced as well, because of less compressibility, see Fig. 7.3b. Also, both other kernels are more efficient as they avoid all square roots, except the hybrid which has kept the normalization. Both new kernels also managed to reduced the "boiling"-artifact but they did not remove it completely. The artifact was severely reduced when using the squared kernel.

However, the squared kernel is not better in all aspects. It allowed a massive increase in the spring constants which lead to a simulation looking like a hybrid between water and sand. This could possibly be ameliorated by the use of surface tension. When simulating fluid with extremely high viscosity e.g. flan, this is not a problem and this squared kernel would lend itself efficiently. However, in this context it is a problem.

8.1.3 The Bodin Method

The Bodin method is the only method that shows a non-linearity between mass density and depth which suggest a good approximation of the incompressibility of water up to a certain depth. A consequence of increased incompressibility is that the number of neighbors are more constant which is also seen in Fig. 7.5b.

8.1.4 Comparison

The Müller and Bodin methods are easily compared. They are based on solid theory and the discretization principles have been well studied. However, the Clavet method is not as easily compare with numbers. Still, the methods are compared in Fig. 7.7a where the density as a function of depth is shown using only *one* version of the respective methods.

It is clear from Fig. 7.7a that the Bodin method offers the greatest incompressibility of the three. Almost the *entire* fluid has close to correct density as opposed to the other where the density error is large only after 1 meter.

All methods benefit from a reduction in time-step. Especially the Müller method in particular reduces its density error substantially, see Fig. 7.2. The Clavet method also gains much stability and incompressibility, see Fig. 7.4. Both these methods increases notably in volume due to more incompressibility.

By comparing Fig. 7.7 with Fig. 7.8a, it could be believed that the Bodin method does not benefit much from an halved time-step. However, Fig. 7.6 reveals a rather large increase in incompressibility, especially near the bottom. If all methods should give, roughly the same incompressibility the time-step could be varied individually for each method. In Fig. 7.8 the

time-step is reduced to 1/360 s for Müller and Clavet methods while the original time-step of 1/120 s is used for the Bodin method.

8.2 Experiment 1.2 - The Stability and Energy Test

In the Müller method and the Clavet method, the maximized spring constants leaves the method on the verge of instability. These spring constants must also be changed each time the initial condition of the simulation changes, e.g. the number of particles is changed, in order to maintain a "good" simulation. This is time-consuming, frustrating and does not lend itself to game development efficiently. An option would be to keep the spring constants low. However, this would loose incompressibility and result in a fluid simulation that "bounces". Another option might be to dynamically set the spring constants. However, even though it has not been tried, it would likely be difficult.

8.2.1 The Müller Method

From Fig. 7.9, one should notice that it takes a *very* long time for the fluid to converge to rest. This is clearly visible and shatters the illusion of the simulation. This poor result is due to the immense pressure of 2000 heavy particles in a pillar.

Changing the number of simulated particles to 1000, the time of convergence is heavily reduced. This, together with the result of the incompressibility experiment, suggest that this method have great difficulty dealing with high pressure, but seems to work rather nicely on shallow volumes, i.e. less than a meter when using these parameters.

8.2.2 The Clavet Method

Fig. 7.11 seems to suggest that this method cannot handle high pressure. This points in the same direction as the results of the Müller method. However, the problems are fundamentally different. The Clavet method might not *ever* converge while the Müller method will, just taking an awful long time.

8.2.3 The Bodin Method

As the Clavet method, this too is not guaranteed to converge. However, The movement of the fluid after the initial time is very different. This does not introduce flow (boiling) but vibrates around the boundary. This is less of a problem because the rendering can smooth such vibration. Also, the number of iterations in the solver can be increased or another solver can be used in order to remedy the error. The error causing these vibrations have a well known source, as opposed to the Clavet method, and can therefore be tackled. Note, however, that increasing the number of iterations will affect performance.

When decreasing the number of particles, the kinetic energy quickly converge, see Fig. 7.12b. Visually, this is very pleasing. The kinetic energy is the result of rather realistic waves that sway back and forth, not from fluid that bounces up and down.

8.3 Experiment 2 - The Performance Test

Both the Müller method and the Clavet method report much higher run-time when simulating a pillar rather than a dam-break. This is due to their lack of maintaining incompressibility and their number of neighbors is much higher in the pillar. The Bodin method on the other hand manages to give roughly the same run-time for both kinds of simulations. The execution time of the Bodin method seems roughly unaffected by type of simulation while the others are heavily influenced by context.

It is believed that the Bodin method has most room for optimization. Faster solvers methods than Gauss-Seidel exists, e.g. Preconditioned Conjugate Gradient. However, such optimizations are beyond the scope of this text.

Important to note is that, if a certain incompressibility is a requirement, then the Bodin method would be much faster than the other two. For example, the result in Fig. 7.8 suggest that the Müller method and the Clavet method must be run 3 times per update to match the incompressibility of the Bodin method. This would make the Bodin method 2-4 times faster than the other two.

8.4 Experiment 3 - The Rigid Body Test

The first sub-experiment was to drop five rigid bodies, of different density, in the fluid. The converged state of all the methods, see Fig. 7.13 through 7.15, show that the light body floats while the heavy ones are close to, or at, the bottom. This is in line with Archimedes principle. However, all bodies (except the body with density equal to the reference density of the fluid) should either flow or sink. None of the methods fully support this. The Müller method is close, but the body with 75 kg / m^2 is considered too submerged for the result to be accepted.

The lack of buoyancy is further illustrated by Fig. 7.16. Here, the second scenario is recorded in which fluid is dropped *on* the bodies. None of the bodies floated to the surface with any method.

When the bodies are in this position, i.e. when one of their sides are in contact with the boundary, there is no upward force (except the normal force keeping it from not falling through the floor). There is force on the sides, from the pressure of the fluid, and most certainly from above where the weight of the fluid presses it down.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The purpose, or reason, for having fluids in games can vary. The game mechanics can depend heavily on the properties of the fluid. If the player should manipulate the flow by constructing pumps and pipes, the fluid *must* behave physically correct. Another reason could be to simply add a cool effect in which physically correctness is less necessary. In this section, some advantages and disadvantages with using these methods are presented.

The Müller method is the oldest method and it has good connection to physical quantities. However, the single largest problem is that in order to achieve good incompressibility, one must lower the time-step to values which would be unfeasible in a real-time application. If only small volumes should be simulated, however, one could run the fluid simulation code several times per game-update. This might be an acceptable solution, especially if the fluid could be simulated using accelerated hardware. However, the Bodin method would probably be a better bet. It gives similar result to the Müller method when run with a very low time-step and would therefore not need to be run several times per update. The left-over computational time could be used to increase the number of iterations of the solver instead, or maybe add more particles. The only real downside to this approach is that the Bodin method is rather hard to implement while implementing the Müller method is surprisingly easy. A simple version would only demand a couple of hours.

The strength of The Clavet method is two-fold. First, it is also easy to implement, not quite as easy as the Müller method but still. Secondly, it can easily be extended to give all sorts of different behaviors. In Clavet et al. [2005] they use explicit viscosity and springs to make it even more viscous. They also use a small variation in the collision handling which allows the fluid to stick on object and form drops. However, the down-sides are many. Lack of clear connection to physical quantities makes it difficult to integrate properly in an existing mechanics engine. It can also take time to tune for different scenarios, increasing development time. Even when the implementation is done it can still take many hours to achieve the sought behavior. If extremely viscous fluids are sought, like jello, the Squared kernels would lend them selfs efficiently, making it much more incompressible than the

Müller method.

The Bodin method is a rather new method with an interesting approach. Similar methods will probably have a clear place in game-physics in the time to come. The increased difficulty from the other two can be problematic. However, the method is intended to be run as part of a multi-physics engine in which the solver is central and already present. If one considers the Bodin method as an *extension* to a physics engine, i.e. integrated and merged with the physics engine, not an *addition*, i.e. along side an existing physics engine such as this case, then it would probably be easier to implement.

There is little room for the Bodin method to increase viscosity in the current implementation. Bodin et al. [2011] mentions that one could increase the damping time τ (or d), but this also increases the error. Another option they mention is to use traditional viscosity models, e.g. the one used by the Müller method and to add the resulting force as external force to spook. However, this would, off course, have a negative effect on performance. The last option they mention, and also the one they recommend, is to use *Rayleigh dissipation functions* to construct a viscosity constraint.

The Bodin method and the Clavet method are, in a way, opposite each other. The Bodin method has good physical properties but harder to extend. The Clavet method has poor physical properties but easy to extend.

Summary The Müller method is not really useful because it demands a too small time-step. The Clavet method is very flexible, fast and good with viscous fluids, but lack physical accuracy. The Bodin method has very nice physical accuracy and show great incompressibility at rather high time-step, but is rather difficult to implement and integrate in Box2d.

9.2 Future work

Physics in games will certainly continue to prosper. The level of abstraction is steadily rising as more and more developers use higher level languages e.g. Java, C# (with Xna), Flash. Soon, most physics engines will support fluids of different kinds.

However, it will be impossible to unify all different methods of simulating fluids. Therefore, it would be interesting to see how different methods could be used in unison. In Losasso et al. [2008], they use both level set methods *and* Sph in the same simulation.

Another area is *dynamic resolution in particle fluids*. This is researched in Adams et al. [2007], where they use larger particles in the middle of the fluid body where the size would not be noticeable.

List of Figures

List of Tables

List of Algorithms

References

Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26, July 2007. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1276377.1276437. URL `http://doi.acm.org/10.1145/1276377.1276437`.

Robert A. Adams. *Calculus - A Complete Course*. Person Education Limited, 6th edition, 2006.

Anthony Bedford, Wallace Fowler, and Yusof Ahmad. *Engineering Mechanics: Dynamics*. Prentice Hall, 5th edition, 2008.

Kenneth Bodin, Claude Lacoursiere, and Martin Servin. Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2011. ISSN 1077-2626. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.29.

Colin Braley and Adrian Sandu. Fluid simulation for computer graphics: A tutorial in grid based and particle based methods. Retrieved: 2011-04-17, 2009. URL `http://colinbraley.com/`.

Mark Carlson, Peter J. Mucha, R. Brooks Van Horn III, and Greg Turk. Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 167–174, New York, NY, USA, 2002. ACM. ISBN 1-58113-573-4. doi: http://doi.acm.org/10.1145/545261.545289. URL `http://doi.acm.org/10.1145/545261.545289`.

Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23:377–384, August 2004. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1015706.1015733. URL `http://doi.acm.org/10.1145/1015706.1015733`.

Erin Catto. *Box2D v2.1.0 User Manual*, 2010. URL `http://www.box2d.org/manual.html`. Retrieved: 2011-02-14.

Erin Catto. Iterative dynamics with temporal coherence. *White paper for Game Developer Conference*, 2005. URL `http://physics.hardwire.cz/mirror/IterativeDynamics.pdf`. Retrieved: 2011-02-14.

Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 219–228, New York, NY, USA, 2005.

ACM. ISBN 1-59593-198-8. doi: http://doi.acm.org/10.1145/1073368.1073400. URL http://doi.acm.org/10.1145/1073368.1073400.

Fabrice Colin, Richard Egli, and Feng Ying Lin. Computing a null divergence velocity field using smoothed particle hydrodynamics. *J. Comput. Phys.*, 217:680–692, September 2006. ISSN 0021-9991. doi: 10.1016/j.jcp.2006.01.021. URL http://portal.acm.org/citation.cfm?id=1217520.1217540.

Peter J. Cossins. *The Gravitational Instability and its Role in the Evolution of Protostellar and Protoplanetary Discs. Chapter 3: Smoothed Particle Hydrodynamics - Or: How I Learned to Stop Worrying and Love the Lagrangian.* PhD thesis, University of Leicester, 2010. arXiv:1007.1245v2 [astro-ph.IM].

Erwin Coumans. *Bullet 2.76 Physics SDK Manual*, 2010. URL http://www.bulletphysics.com/ftp/pub/test/physics/Bullet_User_Manual.pdf. Retrieved: 2011-02-15.

Emanuel Dahlberg. Electricity in a 2d mechanics simulator for education. Master's thesis, Umeå University, 2011.

David H. Eberly. *Game Physics*. Morgan Kaufmann Publishers, 2nd edition, 2010.

Christer Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, January 2005.

Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann. *Physics-Based Animation*. Charles River Media, Inc., 2005.

Emil Ernerfeldt. Phun - 2d physics sandbox. Master's thesis, Umeå University, 2009.

Robert A. Gingold and Joseph J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977.

Thomas T. Goldsmith Jr. and Estle Ray Mann. Cathode-ray tube amusement device. *United States Patent Office*, (2 455 992), 1948.

Andreas Grahn. Interactive simulation of contrast fluid using smoothed particle hydrodynamics. Master's thesis, Umeå University, 2008.

Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed Particle Hydrodynamics on GPUs. In *Proceedings of Computer Graphics International*, pages 63–70, 2007.

Kei Iwasaki, Hideyuki Uchida, and Tomoyuki Nishita Yoshinori Dobashi. Fast particle-based visual simulation of ice melting. *Computer Graphics Forum*, 29(7): 2215–2223, 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2010.01810.x. URL http://dx.doi.org/10.1111/j.1467-8659.2010.01810.x.

Micky Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics. Master's thesis, University of Copenhagen, January 2006.

Sho Kurose and Shigeo Takahashi. Constraint-based simulation of interactions between fluids and unconstrained rigid bodies. In *Proceedings of the 2009 Spring Conference on Computer Graphics*, SCCG '09, pages 181–188, New York, NY, USA, 2009. ACM. ISBN 978-1-4503-0769-7. doi: http://doi.acm.org/10.1145/1980462.1980498. URL http://doi.acm.org/10.1145/1980462.1980498.

Claude Lacoursière. Simple numerical methods for mechanical systems subject to strong forces and constraints, April 2010. URL http://www8.cs.umu.se/kurser/5DV058/VT10/lectures/cl-lecture-notes.pdf. Retrieved: 2011-04-14. Lecture notes.

Claude Lacoursière. *Ghosts and Machines: Regularized Variational Methods for Interactive Simulations of Multibodies with Dry Frictional Contacts*. PhD thesis, Umeå University, 2007.

Claude Lacoursière, Martin Servin, and Anders Backman. Fast and stable simulation of granular matter and machines. In *DEM5 - The Fifth International Conference on Discrete Element Methods*, August 2010.

Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.*, 31:302–325, September 2005. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/1089014.1089017. URL http://doi.acm.org/10.1145/1089014.1089017.

Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. Two-way coupled sph and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14:797–804, July 2008. ISSN 1077-2626. doi: 10.1109/TVCG.2008.37. URL http://portal.acm.org/citation.cfm?id=1373109.1373251.

Leon B. Lucy. A numerical approach to the testing of the fission hypothesis. *Monthly Notices of the Royal Astronomical Society*, 82(12):1013–1024, December 1977.

Tom Madams. Why my fluids don't flow, December 2010. URL http://imdoingitwrong.wordpress.com/2010/12/14/why-my-fluids-dont-flow/. Retrieved: 2011-04-14. Blog.

Ian Millington. *Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game*. Morgan Kaufmann Publishers, 2nd edition, 2010.

Joe Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8): 1703–1759, 2005.

Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5. URL http://portal.acm.org/citation.cfm?id=846276.846298.

Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25:809–836, December 2006.

Martin Nilsson. Constraint fluids on GPU. Master's thesis, Umeå University, 2009.

Brandon Pelfrey. An informal tutorial on smoothed particle hydrodynamics for interactive simulation. 2010. URL http://www.cs.clemson.edu/~bpelfre/sph_tutorial.pdf.

Martin Servin. Notes on discrete mechanics, March 2010. URL http://www8.cs.umu.se/kurser/5DV058/VT10/lectures/Notes_on_Discrete_Mechanics.pdf. Retrieved: 2011-04-14. Lecture notes.

Barbara Solenthaler and Renato Pajarola. Density contrast sph interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 211–218, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association. ISBN 978-3-905674-10-1. URL `http://portal.acm.org/citation.cfm?id=1632592.1632623`.

Jos Stam. Real-time fluid dynamics for games. *Proceedings of the Game Developer Conference*, March 2003.

Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 2nd edition, January 1986.

Gilbert Strang. Mathematical methods for engineers II (18.086) - lecture 15 - iterative methods and preconditioners, 2006. URL `http://ocw.mit.edu`.

Jie Tan and Xubo Yang. Physically-based fluid animation: A survey. *Science in China Series F Information Sciences*, 52(5):723–740, 2009.

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. *Proceedings of Vision, Modeling, and Visualization*, pages 47–54, November 2003.

Xu Wang. Method of steepest descent and its applications. Retrieved: 2011-04-14, November 2008. URL `http://sces.phys.utk.edu/~moreo/mm08/XuWangP571.pdf`.

Mick West. Practical fluid mechanics. *Game Developer magazine*, March 2007. URL `http://cowboyprogramming.com/2008/04/01/practical-fluid-mechanics/`. Retrieved: 2011-03-12.

Mehmet Yildiz, Rusty A. Rook, and Afzal Suleman. SPH with the multiple boundary tangent method. *International Journal for Numerical Methods in Engineering*, 9999(9999): n/a+, 2008. doi: 10.1002/nme.2458. URL `http://dx.doi.org/10.1002/nme.2458`.

Appendix A

Kernels

A.1 Kernels for the Müller method in 2D

Müller et al. [2003] suggested a kernel function on the following form

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

The scalar $\frac{315}{64\pi h^9}$ makes sure that the kernel is normalized. This value is derived by integrating the core part $(h^2 - r^2)^3$ over a sphere with radius h.

$$\frac{64\pi h^9}{315} = \int_V (h^2 - r^2)^3 d\boldsymbol{r} \tag{A.2}$$

This paper deals specifically with fluids in two dimension and should therefore derive another constant. The integration regards h as *constant*. As such the kernel function in 2D is a function of two *variables* ($z = f(x, y)$). It therefore defines a volume in 3D space and the sought normalization constant is one over the magnitude of this volume, $1/V$.

First assume that $y = 0$, then varying x between zero and h gives the plot in Fig. A.1. If the area under this curve is rotated around the z-axis it will trace out the sought volume.

Integral calculus states that the volume of the solid obtained by rotating the plane region $0 \leq z \leq f(x), 0 \leq a < x < b$ about the z-axis is (derived using *Cylindrical Shells*.)

$$V = 2\pi \int_a^b x f(x) dx \tag{A.3}$$

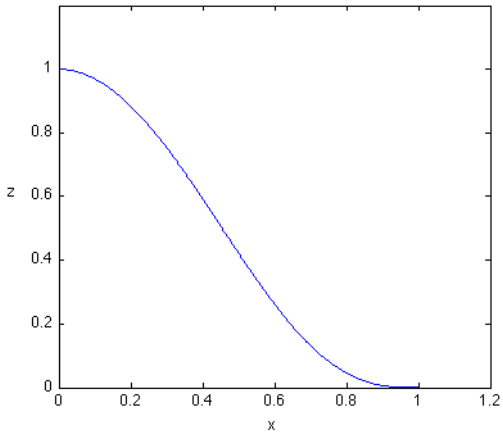


Figure A.1: *A cross section of the volume in* 3D *traced by the two dimensional poly 6 kernel function. The area of this swept around the z-axis gives the sought volume.*

In this case $a = 0, b = h$ and

$$V = 2\pi \int_0^h x(h^2 - x^2)^3 dx \tag{A.4}$$

Using the method of substitution, let $u = h^2 - x^2$ and $du = -2x\ dx$ then

$$\begin{aligned} V &= -\pi \int_{h^2}^0 u^3 du \\ &= -\pi \left[\frac{u^4}{4}\right]_{h^2}^0 \\ &= -\pi \left(0 - \frac{h^8}{4}\right) = \frac{\pi h^8}{4} \end{aligned} \tag{A.5}$$

(A.6)

Hence, the normalization constant for W_{poly6} in two dimensions is

$$\frac{1}{V} = \frac{4}{\pi h^8} \tag{A.7}$$

and W^{2D}_{poly6} can now be defined as

$$W^{2D}_{poly6}(\boldsymbol{r}, h) = \frac{4}{\pi h^8} \begin{cases} (h^2 - r^2)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.8}$$

Müller et al. [2003] also suggest using different kernels when smoothing different values, i.e. Eq. A.9 for pressure and Eq. A.10 for viscosity smoothing

$$W_{spiky}(\boldsymbol{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h-r)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.9}$$

$$W_{viscosity}(\boldsymbol{r}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.10}$$

Integrating these functions ignoring the 3D normalization constants yields

$$\begin{aligned} 2\pi \int_0^h x(h^3 - 3h^2x + 3hx^2 - x^3)dx &= \\ 2\pi \left[\frac{h^3x^2}{2} - h^2x^3 + \frac{3hx^4}{4} - \frac{x^5}{5} \right]_0^h &= \\ 2\pi \left(\frac{h^5}{2} - h^5 + \frac{3h^5}{4} - \frac{h^5}{5} \right) &= \frac{\pi h^5}{10} \end{aligned} \tag{A.11}$$

and

$$\begin{aligned} 2\pi \int_0^h x \left(-\frac{x^3}{2h^3} + \frac{x^2}{h^2} + \frac{h}{2x} - 1 \right) dx &= \\ 2\pi \left(-\frac{h^2}{10} + \frac{h^2}{4} + \frac{h^2}{2} - \frac{h^2}{2} \right) &= \frac{3\pi h^2}{10} \end{aligned} \tag{A.12}$$

With these, the two dimensional counterparts of the kernel functions in Eq. A.9 and Eq. A.10 can be defined.

$$W^{2D}_{spiky}(\boldsymbol{r}, h) = \frac{10}{\pi h^5} \begin{cases} (h-r)^3 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.13}$$

$$W^{2D}_{viscosity}(\boldsymbol{r}, h) = \frac{10}{3\pi h^2} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.14}$$

However, because the acceleration due to pressure (Eq. 3.13) involves a gradient and the viscosity acceleration (Eq. 3.14) involves a Laplacian, these kernels must be differentiated.

$$\nabla W^{2D}_{spiky}(\boldsymbol{r}, h) = -\frac{30}{\pi h^5} \hat{\boldsymbol{r}} \begin{cases} (h-r)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.15}$$

$$\nabla^2 W^{2D}_{viscosity}(\boldsymbol{r}, h) = \frac{10}{3\pi h^2} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.16}$$

The Laplacian of this in 3D Cartesian coordinates is

$$-\frac{6(-hr + r^2)}{h^3 r} = -\frac{6}{h^3}(r - h) \tag{A.17}$$

The Laplacian of this in 2D Cartesian coordinates is

$$\frac{h^4 + 8hr^3 - 9r^4}{2h^3 r^3} = \frac{h^4}{2h^3 r^3} + \frac{8hr^3}{2h^3 r^3} - \frac{9r^4}{2h^3 r^3} = \frac{h}{2r^3} + \frac{4}{h^2} - \frac{9r}{2h^3} \tag{A.18}$$

The Laplacian in 2D Cartesian coordinates does *not* follow the good properties of the Laplacian in 3D. It is not linear and tends to infinity as r approaches zero. Therefore, in the spirit of Müller et al. [2003] another kernel core is devised for this paper

$$-\frac{4r^3}{9h^3} + \frac{r^2}{h^2}. \tag{A.19}$$

The normalization constant of this is calculated from

$$\begin{aligned} 2\pi \int_0^h x \left(-\frac{4x^3}{9h^3} + \frac{x^2}{h^2} \right) dx &= \\ 2\pi \left(\left[\frac{x^4}{4h^2} \right]_0^h - \left[\frac{4x^5}{45h^3} \right]_0^h \right) &= \\ 2\pi \left(\frac{h^2}{4} - \frac{4h^2}{45} \right) &= \frac{29}{90}\pi h^2. \end{aligned} \tag{A.20}$$

(A.21)

The normalized kernel becomes

$$W^{2D}_{viscosity}(\boldsymbol{r}, h) = \frac{90}{29\pi h^2} \begin{cases} -\frac{4r^3}{9h^3} + \frac{r^2}{h^2} & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise.} \end{cases} \tag{A.22}$$

The Laplacian of the core in 2D Cartesian coordinates is

$$-\frac{-4hr + 4(r^2)}{h^3 r} = -\frac{4r(-h + r)}{h^3 r} = \frac{4(h - r)}{h^3}. \tag{A.23}$$

The Laplacian of the kernel then becomes

$$\nabla^2 W^{2D}_{viscosity}(\boldsymbol{r}, h) = \frac{360}{29\pi h^5} \begin{cases} (h-r) & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.24}$$

A.2 Kernels and Parameters for the Clavet method in 2D

First of all, the kernels used in the paper Clavet et al. [2005] when normalized for 2D coordinates are presented below.

$$W^{clavet}(\boldsymbol{r}, h) = \frac{6}{\pi h^2} \begin{cases} \left(1 - \frac{r}{h}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.25}$$

$$W^{near}(\boldsymbol{r}, h) = \frac{10}{\pi h^2} \begin{cases} \left(1 - \frac{r}{h}\right)^3 & \text{if } 0 < r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.26}$$

$$\nabla W^{clavet}(\boldsymbol{r}, h) = -\frac{12}{\pi h^3} \hat{\boldsymbol{r}} \begin{cases} \left(1 - \frac{r}{h}\right) & \text{if } 0 < r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.27}$$

$$\nabla W^{near}(\boldsymbol{r}, h) = -\frac{30}{\pi h^3} \hat{\boldsymbol{r}} \begin{cases} \left(1 - \frac{r}{h}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.28}$$

Gravity (or any acceleration) is integrated by

$$\begin{aligned} \boldsymbol{v}^{(t+1)} &= \boldsymbol{v}^{(t)} + \boldsymbol{g}\Delta t, \\ \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} + \boldsymbol{v}^{(t+1)}\Delta t. \end{aligned}$$

If gravity $\boldsymbol{g}$ should be α times larger, then the time-step Δt must be multiplied with $\frac{1}{\sqrt{\alpha}}$ to give the same position,

$$\boldsymbol{p}^{(t+1)} = \boldsymbol{p}^{(t)} + \boldsymbol{v}^{(t)} + (\boldsymbol{g}\alpha)\left(\frac{\Delta t}{\sqrt{\alpha}}\right)^2. \tag{A.29}$$

The time-step is used when calculating the displacement d too, so

$$d = \left(\frac{\Delta t}{\sqrt{\alpha}}\right)^2 (p\nabla W^{clavet} + p^{near}\nabla W^{near}),$$

where the pressure must now be scaled by α in order to give the same displacement,

$$\begin{aligned} p &= \alpha k(\rho - \rho_0), \\ p^{near} &= \alpha k^{\text{near}}(\rho^{near}). \end{aligned}$$

If the kernels are split in core parts ($\bar{W}^x$, $\nabla\bar{W}^y$) and normalization constants (a^x, b^y) (where both x and y can be *clavet* or *near*), then the density ρ can be written

$$\begin{aligned} \rho &= \rho a^{clavet}\bar{W}^{clavet}, \\ \rho^{near} &= \rho^{near} a^{near}\bar{W}^{near}. \end{aligned}$$

Let ρ_0 be scaled by a^{clavet}, then the pressure becomes

$$\begin{aligned} p &= \frac{\alpha k}{a^{clavet}}(a^{clavet}\rho - a^{clavet}\rho_0), \\ p^{near} &= \frac{\alpha k^{\text{near}}}{a^{near}}(a^{near}\rho^{near}). \end{aligned}$$

However, the pressure is multiplied with one more normalization constant in the calculation of displacement,

$$d = \left(\frac{\Delta t}{\sqrt{\alpha}}\right)^2 (pb^{clavet}\nabla\bar{W}^{clavet} + p^{near}b^{near}\nabla\bar{W}^{near}).$$

Hence, the pressure should instead be

$$\begin{aligned} p &= \frac{\alpha k}{a^{clavet}b^{clavet}}(a^{clavet}\rho - a^{clavet}\rho_0), \\ p^{near} &= \frac{\alpha k^{\text{near}}}{a^{near}b^{near}}(a^{near}\rho^{near}). \end{aligned}$$

Table A.1: *The parameters given in the original paper when modified by extraction of normalization constants.*

	Previous value	Changed to	Updated approximative values
Reference density, ρ_0	10 kg/m^2	Changed to	100 kg/m^2
Scaling factor, α	1	Changed to	9820
Gravity, g	0.001 m/s^2	αg	9.82 m/s^2
Smoothing radius, h	N/A	Use Eq. 3.42	$\sqrt{\frac{6}{\pi\beta\rho_0}} \approx 0.138$ m (if $\beta = 1$)
Time step, Δt	1 s	$\frac{1}{\sqrt{\alpha}}$	$\frac{1}{\sqrt{9820}} \approx \frac{1}{99}$ s
k	0.004 N/m	$\frac{\alpha k}{a^{clavet} b^{clavet}}$	$\frac{9820 \cdot 0.004}{\frac{6}{\pi h^2}\frac{-12}{\pi h^3}} \approx -0.00027$
k^{near}	0.01 N/m	$\frac{\alpha k^{\text{near}}}{a^{near} b^{near}}$	$\frac{9820 \cdot 0.01}{\frac{10}{\pi h^2}\frac{-30}{\pi h^3}} \approx -0.00016$

to give the same displacement. Applying these results would given the parameters in Table A.1

The approximative values are calculated with some assumptions such as $\beta = 1$ in the calculation of the smoothing length h and that this h is used when calculating k and k^{near}. The value of the spring constants are still parameters set by the user, table A.1 just shows a mapping of the parameters specified in Clavet et al. [2005] to comparable parameters.

A.3 Hybrid Kernels for the Clavet method

$$W^{clavet}(\boldsymbol{r}, h) = \frac{6}{\pi h^2} \begin{cases} \left(1 - \frac{r^2}{h^2}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.30}$$

$$W^{near}(\boldsymbol{r}, h) = \frac{10}{\pi h^2} \begin{cases} \left(1 - \frac{r^2}{h^2}\right)^3 & \text{if } 0 < r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{A.31}$$

$$\nabla W^{clavet}(\boldsymbol{r},h) = -\frac{12}{\pi h^3}\hat{\boldsymbol{r}}\begin{cases}\left(1-\frac{r^2}{h^2}\right) & \text{if } 0 < r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.32}$$

$$\nabla W^{near}(\boldsymbol{r},h) = -\frac{30}{\pi h^3}\hat{\boldsymbol{r}}\begin{cases}\left(1-\frac{r^2}{h^2}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.33}$$

A.4 Squared Kernels for the Clavet method

$$W^{clavet}(\boldsymbol{r},h) = \frac{3}{\pi h^2}\begin{cases}\left(1-\frac{r^2}{h^2}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.34}$$

$$W^{near}(\boldsymbol{r},h) = \frac{4}{\pi h^2}\begin{cases}\left(1-\frac{r^2}{h^2}\right)^3 & \text{if } 0 < r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.35}$$

$$\nabla W^{clavet}(\boldsymbol{r},h) = -\frac{12}{\pi h^4}\boldsymbol{r}\begin{cases}\left(1-\frac{r^2}{h^2}\right) & \text{if } 0 < r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.36}$$

$$\nabla W^{near}(\boldsymbol{r},h) = -\frac{24}{\pi h^4}\boldsymbol{r}\begin{cases}\left(1-\frac{r^2}{h^2}\right)^2 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise}\end{cases} \tag{A.37}$$

A.5 Kernels for the Bodin method in 2D

Bodin et al. [2011] does not specify which kernels they use. Therefore, the kernels from Nilsson [2009] is used instead. They use poly6 for density and a modified version of it's gradient to calculate the entries in the Jacobian matrix.

The poly 6 kernel and its gradient in 3D are

$$W_{poly6}(\boldsymbol{r},h) = \frac{315}{64\pi h^9}\begin{cases}(h^2-r^2)^3 & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise,}\end{cases} \tag{A.38}$$

$$\nabla W_{poly6}(\boldsymbol{r},h) = \frac{945}{32\pi h^9}\hat{\boldsymbol{r}}\begin{cases}(h^2-r^2)^2 & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise.}\end{cases} \tag{A.39}$$

The modified version in Nilsson [2009] reads

$$\nabla W_{poly6h}(\boldsymbol{r}, h) = \frac{945}{32\pi h^8}\hat{\boldsymbol{r}} \begin{cases} (h^2 - r^2)^2 & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise,} \end{cases} \tag{A.40}$$

i.e. the gradient of poly6 is scaled with the smoothing length h. It is therefore named *poly6h*. Nilsson [2009] claims that this has "better stability compared to the formal definition of the kernel gradient".

The gradient of the poly6 kernel in 2D is

$$\nabla W^{2D}_{poly6}(\boldsymbol{r}, h) = \frac{24}{\pi h^8}\hat{\boldsymbol{r}} \begin{cases} (h^2 - r^2)^2 & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise.} \end{cases} \tag{A.41}$$

If the same approach is used, i.e. to scale the formal kernel gradient by the smoothing length then,

$$\nabla W^{2D}_{poly6h}(\boldsymbol{r}, h) = \frac{24}{\pi h^7}\hat{\boldsymbol{r}} \begin{cases} (h^2 - r^2)^2 & \text{if } 0 \leq r \leq h, \\ 0 & \text{otherwise.} \end{cases} \tag{A.42}$$

This gives much better stability in this 2D context too and therefore used in the application.