Examensarbete
LITH-ITN-MT-EX--05/027--SE

Simulation and visualization of incompressible fluids using improved particle level sets

Andreas Söderström

2005-03-10

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings Universitet
601 74 Norrköping

LITH-ITN-MT-EX--05/027--SE

Simulation and visualization of incompressible fluids using improved particle level sets

Examensarbete utfört i medieteknik
vid Linköpings Tekniska Högskola, Campus Norrköping

Andreas Söderström

Handledare Ken Museth
Examinator Ken Museth

Norrköping 2005-03-10

Avdelning, Institution Division, Department Institutionen för teknik och naturvetenskap Department of Science and Technology	**Datum** Date **2005-03-10**

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ _______________	**Rapporttyp** Report category Examensarbete ☐ B-uppsats ☐ C-uppsats ☒ D-uppsats ☐ _______________	**ISBN** **ISRN** **LITH-ITN-MT-EX--05/027--SE** **Serietitel och serienummer** Title of series, numbering **ISSN**

URL för elektronisk version

Titel Title	Simulation and visualization of incompressible fluids using improved particle level sets
Författare Author	Andreas Söderström

Sammanfattning
Abstract

This thesis presents a system for free surface fluid simulation using improved particle level sets. The system is based on previous work in the field of fluid simulation and level sets but contains some novel features, most important of which is a method to guarantee volume conservation for the simulated fluid. A simple approach to simulated surface friction is also presented as well as a modification to the particle level set method designed to improve the capturing of thin fluid interfaces on boundary surfaces

Nyckelord
Keyword

computational fluid dynamics, level sets, particles, incompressible fluids, visualization, stable fluids

Simulation and visualization of incompressible fluids using improved particle level sets

Andreas Söderström

Supervisor: Professor Ken Museth

Linköping 2005-03-17

Abstract

A system for free surface fluid simulation using improved particle level sets is presented. The system is based on previous work in the field of fluid simulation and level sets but contains some novel features, most important of which is a method to guarantee volume conservation for the simulated fluid. A simple approach to simulated surface friction is also presented as well as a modification to the particle level set method designed to improve the capturing of thin fluid interfaces on boundary surfaces.

Contents

List of figures

1 Introduction

Though the field of computational fluid dynamics (CFD) has been around for quite some time the simulation of free surface flows for graphics applications is a recent addition. There has been some quite impressive work previously published in this field but there still are many things that can be improved upon or further developed. The aim of this thesis is to describe a system for free surface fluid simulation and visualization building on previous work in the field and to provide solutions to some of the problems associated with such a system. The system will be based on Eulerian methods using level sets to represent geometry. Our own implementation of Lagrangian marker particles as described by Enright [2] will be used to improve the ability of the level sets to preserve fine features. The particle system will contain a few novel features, one regarding the treatment of particles on thin surfaces in contact with external boundaries and one regarding the deletion of particles. A simple method for simulating surface friction is introduced as well as a novel feature that aims to eliminate volume loss during the simulation – a common problem for fluid simulation using Eulerian methods. The simulation software is intended to be able to handle complex geometry that is easy to model and use. This is achieved by using the closest point transform to allow the user to model the environment and initial fluid surface in a 3D modeling program and then read this model into the simulation software. To solve the Navier-Stokes equations that drive the simulation kinetically we will use the projection method with the stable fluids approach introduced by Stam [12]. This will achieve a stable simulation environment that allows for different time-steps to be used for the Navier-Stokes solver and the level sets. The simulation system is meant to achieve visually pleasing results and some compromises will be done with regard to the accuracy of the physics involved.

2 The theory

The theoretical background to this thesis is mainly based on work by Stam [11], [12] , Enright et. al. [1] and Enright [2]. The book on level set methods by Osher and Fedkiw [9] has also been very helpful as well as the work by Peng et. al. [10].

2.1 Notations

A few words should be said about the notations used in this thesis. Variables and functions will be written in italic letters and vectors will be written in bold, italic letters or with an arrow over the vector. Sets will be written in upper-case letters and matrices and operators in upper-case bold letters. Units will be written in brackets. Below is a short list of some of the most commonly used notations:

x	the variable x
$f()$	the function f
$\boldsymbol{x}$	the vector x
$\vec{x}$	another way of writing the vector x
S	the set S
M	the matrix M
P	the operator P
[m/s]	the unit ‘velocity’ (m/s)
$\forall$	‘for all’
:	‘such that’

2.2 Level sets

Given a function

$$\phi : D \to \eta, D \subset \Re^n, \eta \in \Re$$

we can define the level set

$$L_f(\eta) \equiv \{\xi \in D : \phi(\xi) = \eta\} \tag{2.1}$$

in other words the set of points that solve the equation

$$\phi(\xi) = \eta \tag{2.2}$$

Let us look at the level set $\phi(x,y) = x^2 + y^2$. This describes all possible circles that have their center-point at the origin. To find the actual circle with radius r one must solve equation (2.2) with $\eta = r^2$. η is often referred to as the iso-value for the implicit surface it specifies. This n-1 dimensional subspace to the *n* dimensional level set is called the iso-surface associated with the specified iso-value. If we for example choose $r = 1 \Rightarrow \eta = 1$ we get the equation $x^2 + y^2 = 1$ which is the implicit description of the circle with radius 1. One can easily imagine that there exists a set of values for *x* and *y* that satisfies this equation but it is not immediately clear which values since they are implicitly given. For every value of *x* and *y* we can however identify three important cases which are shown in figure 2.1

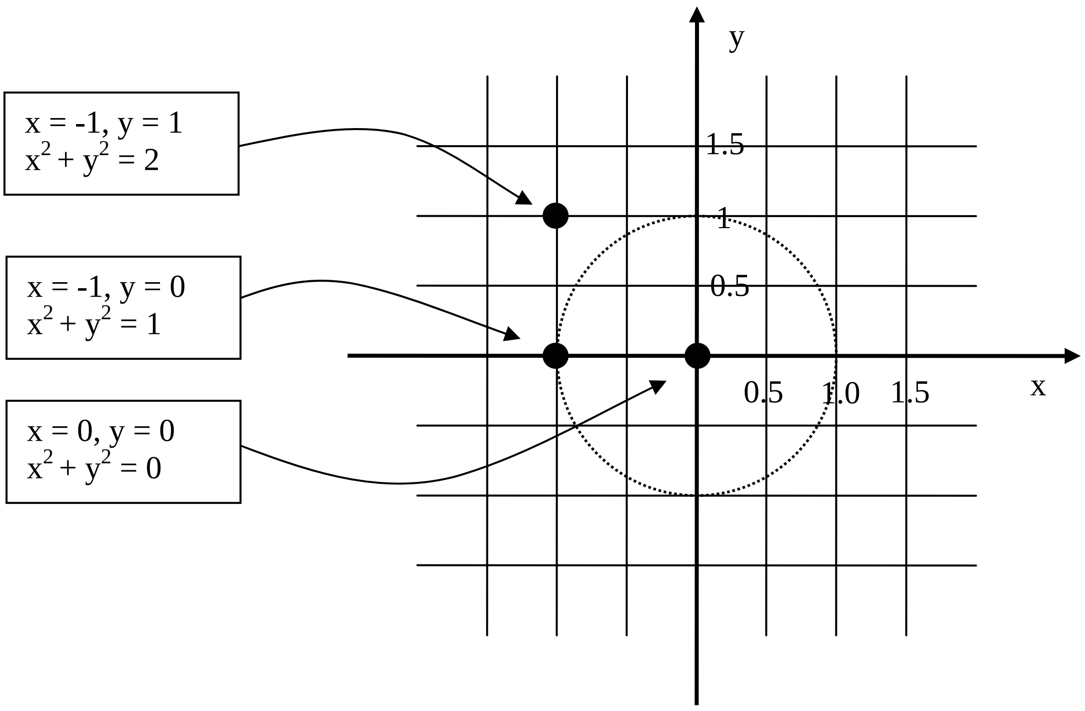


Figure 2.1 Implicit circle with $r = 1$ showing the three principal cases: outside, inside and on the iso-surface.

From this simple example we can see that we can determine if a point in space is inside, outside or on the implicit circle by comparing the value of the implicit function to the iso-value corresponding to that radius. The iso-surface can be seen as the interface that separates two regions of space and we will hereby refer to it simply as the *interface* S. Using vector notation and where η is the iso-value the interface can be defined by

$$S \equiv \{\boldsymbol{x} : \phi(\boldsymbol{x}) = \eta\} \tag{2.3}$$

For the interior region Ω bounded by S we get

$$\phi(\boldsymbol{x}) < \eta \Longrightarrow \boldsymbol{x} \in \Omega$$
$$\phi(\boldsymbol{x}) > \eta \Longrightarrow \boldsymbol{x} \notin \Omega$$
$$\phi(\boldsymbol{x}) = \eta \Longrightarrow \boldsymbol{x} \in \partial\Omega \equiv S$$

The gradient of an n-dimensional level set is calculated as

$$\nabla\phi \equiv \left(\frac{\partial\phi}{\partial x_1}, \frac{\partial\phi}{\partial x_2} \cdots \frac{\partial\phi}{\partial x_n}\right) \qquad (2.4)$$

and the unit normal will be

$$\vec{n} \equiv \frac{\nabla\phi}{|\nabla\phi|} \qquad (2.5)$$

The normal ***n*** is always perpendicular to the iso-surfaces and point in the direction of increasing ϕ.

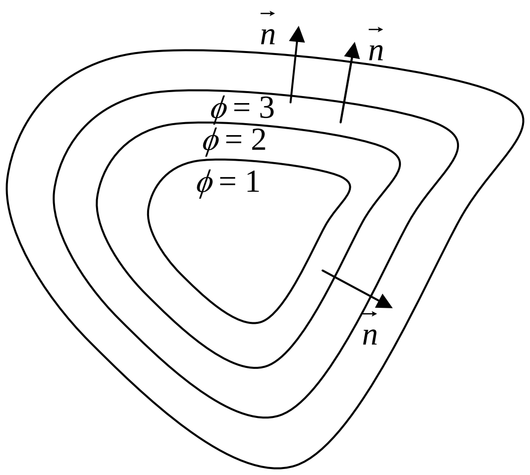


Figure 2.2 Gradients and normals on implicit functions

2.3 Euclidian distance fields

A special kind of implicit function that will be very useful for us is the Euclidian distance function which is defined by

$$\phi(\boldsymbol{x}) = \min(|\boldsymbol{x} - \boldsymbol{x}_s|)\ \forall\ \boldsymbol{x}_s \in \partial\Omega \qquad (2.6)$$
$$|\nabla\phi| = 1 \qquad (2.7)$$
$$\boldsymbol{x} \in \Re$$

This means that $\phi(\boldsymbol{x})$ is a scalar field that gives the closest Euclidian distance to the interface for every point in space. The geometrical information provided by this is very useful and from here on the level sets used will be (signed) Euclidian distance fields. The sign convention used in this thesis is negative distances on the inside of the interface and positive distance outside.

Though a powerful tool the distance field has one problem that one should be aware of: For points in space that are equidistant to more than one point on the interface the gradient of the signed distance field becomes ill defined. An example of this for a rectangle can be seen in figure 2.3.

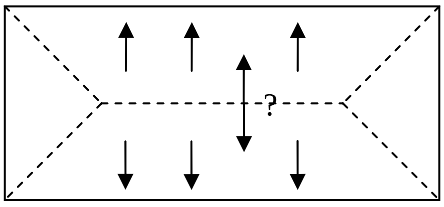


Figure 2.3 Undefined normals in the distance field of a rectangle

This will cause problems when solving equations that depend on the gradient of the distance field and one has to be extra careful when designing a numerical solver for these equations.

2.4 Level set dynamics

Since our goal is to simulate the dynamic surface of moving liquid there is one final and critical component that our level set need to have: Dynamics.
To achieve this we will use the dynamic level set formulation:

$$\mathrm{S}(t) = \{\boldsymbol{x}(t) : \phi(\boldsymbol{x}(t), t) = \eta\}. \tag{2.8}$$

By introducing time dependence to the distance function ϕ we can move the interface defined by the iso-value η. Since we may define the initial level set in such a way that the interface we are interested in corresponds to any iso-value we will hereby simply use $\eta = 0$ for convenience. Now let's see what happens if we differentiate equation (2.8) with respect to time.

$$\frac{d}{dt}\phi(x(t),t) = \frac{d}{dt}\eta \Rightarrow$$

$$\frac{\partial\phi}{\partial t} + \frac{\partial\phi}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial\phi}{\partial x_2}\frac{\partial x_2}{\partial t} + \ldots + \frac{\partial\phi}{\partial x_n}\frac{\partial x_n}{\partial t} = 0 \Rightarrow$$

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \bullet \frac{d\vec{x}}{dt} \Rightarrow$$

$$\frac{\partial\phi}{\partial t} = -|\nabla\phi| \cdot \frac{\nabla\phi}{|\nabla\phi|} \bullet \frac{d\vec{x}}{dt} \Rightarrow$$

$$\frac{\partial\phi}{\partial t} = -|\nabla\phi| \cdot \vec{n} \bullet \vec{v} = \tag{2.9}$$

$$= -|\nabla\phi| \cdot F(\vec{x},\vec{n},\ldots) \tag{2.10}$$

We now see that if we define a velocity $\boldsymbol{v}$ in every point in space, thus creating a velocity field we can use that in equation (2.9) to propagate the level set along the velocity field. In general one could specify some *speed function F* that can depend on any number of variables that can then be used to *advect* (move) the interface.

2.5 Navier-Stokes equations

In differential vector form without the viscous stress tensor the Navier-Stokes equations for viscous incompressible flow can be written as

$$\frac{\partial \vec{V}}{\partial t} = \vec{F} + \nu \cdot \nabla^2 \vec{V} - \left(\vec{V} \bullet \nabla\right) \cdot \vec{V} - \frac{\nabla p}{\rho} \qquad (2.11)$$

$$\nabla \bullet \vec{V} = 0 \qquad (2.12)$$

$$\vec{V} = \frac{\partial \vec{x}}{\partial t}$$

Where ***V*** is the velocity field, ***F*** is the external force vector field, ν is the viscosity, *p* is the pressure field and *ρ* is the density. Solving these partial differential equations (PDE:s) is a quite non-trivial problem and has to be done using numerical methods. Fortunately there has been a lot of previous work done in this area. A popular means of solution is the projection method used in [2], [3], [9], [12], [16] among others. A problem when numerically solving the Navier-Stokes equations is however that the solution tends to easily become unstable if the time-step is too large. In [12] Stam suggests an approach to solving the Navier-Stokes equations that is stable regardless of time-step size, something that is very nice for practical fluid simulations. Our solver will be based on this work.

The first step of the projection method is to merge equation (2.11) and (2.12) to create a single equation for the velocity ***V***. This can be done using a mathematical property of vector fields known as *Helmholtz-Hodge Decomposition*. The statement is that any vector field ***u*** can be decomposed into the form

$$\boldsymbol{u} = \boldsymbol{v} + \nabla w \qquad (2.13)$$

$$\nabla \bullet \boldsymbol{v} = 0 \qquad (2.14)$$

where *w* is a scalar field.

A projection operator **P** which projects the vector field ***u*** onto its divergence free part ***v*** = **P*u*** is obtained by first calculating the divergence part *w* of the vector field via the equation

$$\nabla \bullet \vec{u} = \nabla \bullet \vec{v} + \nabla^2 w \Rightarrow$$

$$\nabla \bullet \vec{u} = \nabla^2 w \qquad (2.15)$$

We then obtain the projection operator by using the relation (2.13) as follows:

$$\boldsymbol{v} = \mathbf{P}\boldsymbol{u} = \boldsymbol{u} - \nabla w$$

Using this operator guarantees that equation (2.12) is satisfied and also allows us to rewrite equation (2.11) to

$$\frac{\partial \vec{V}}{\partial t} = \mathbf{P}\left(\vec{F} + \nu \cdot \nabla^2 \vec{V} - \left(\vec{V} \bullet \nabla\right) \cdot \vec{V}\right) \tag{2.16}$$

using **P*V*** = ***V*** since ***V*** is divergence free and where **P**(∇*p/ρ*) = 0 since *p* is a scalar field and thus is only divergence free if the pressure field is constant in which case ∇*p* = 0 trivially.

The idea is now to break up the PDE into components and solve it in parts. The three components will be the external force component ***F***, the self-advection component -(***V***•∇)·***V*** and the diffusion component $\nu\cdot\nabla^2$***V***. We will then obtain the velocity field $\mathbf{w}_{t+dt}$ for some future time *t*+d*t* by updating the old velocity field $\mathbf{w}_t$ according to the algorithm

$$\vec{w}_t \xrightarrow{\text{add force}} \vec{w}_1 \xrightarrow{\text{advect}} \vec{w}_2 \xrightarrow{\text{diffuse}} \vec{w}_3 \xrightarrow{\text{project}} \vec{w}_{t+dt}$$

2.5.1 Forces

Solving the external force term is quite straight-forward. We simply define the force-field ***F*** and create $\boldsymbol{w}_1$ from $\boldsymbol{w}_t$ by using the equation

$$\frac{\partial \vec{w}_1}{\partial t} = F \tag{2.17}$$

2.5.2 Self-advection

The self-advection term is the non-linear -(***V***•∇)·***V*** term of the Navier-Stokes equations. This term is quite important since it allows for non-linear phenomenon like vortices. Using straight-forward methods for representing the self-advection term will unfortunately result in solvers that easily become unstable and thus we will borrow the clever idea for stable fluids presented in [12]. In this paper Stam shows that one can avoid the stability issue by modeling the effect of this term instead of solving it explicitly. The self-advection term can be interpreted as the motion of the velocity field along itself and Stam shows that one can use this insight to model the term as follows:

Given the velocity field $\boldsymbol{w}_1$ we create $\boldsymbol{w}_2$ by using the velocity of $\boldsymbol{w}_1$ on a position that corresponds to the position a zero-mass particle would have had dt time-units ago as seen in figure 2.4

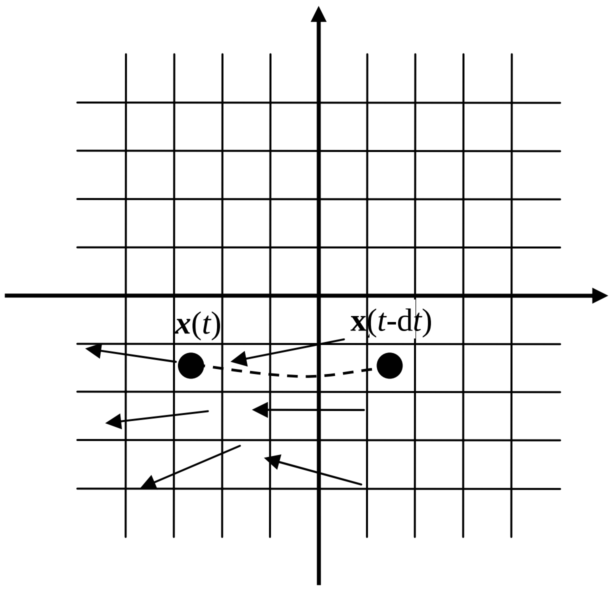


Figure 2.4 Self-advection using the stable fluids approach

If we define the particle trace operator **T** we can write this step as the equation

$$\overrightarrow{w_2}\left(\vec{x}\right) = \overrightarrow{w_1}\left(\mathbf{T}\left(\vec{x}, -\mathrm{d}t\right)\right) \tag{2.18}$$

2.5.3 Diffusion

The diffusion step is pretty straight-forward and the velocity field ***w***$_3$ can be obtained by solving the diffusion equation

$$\frac{\partial \overrightarrow{w_2}}{\partial t} = \nu \nabla^2 \overrightarrow{w_2} \tag{2.19}$$

2.5.4 Projection

The final step in this algorithm to solve equation (2.16) is to apply the projection operator to the vector field **w**$_3$. In order to do this we need to find the scalar field in equation (2.13) that represents the divergence component of **w**$_3$ by solving equation (2.15) and then use the relation

$$\overrightarrow{w_{t+dt}} = \overrightarrow{w_3} - \nabla q \tag{2.20}$$

where *q* is scalar field representing the divergence of **w**$_3$. This allows us to create the final, divergence free vector field **w**$_{t+dt}$.

3 The implementation

The numerical implementation of the algorithms and equations described above is based primarily on previous work described in [2], [9] and [16]. A few novel features are also included, most importantly a modified particle system and a method to force volume conservation regardless of the fluid lost due to numerical inaccuracy or other errors in the simulation. The implementation will be written in C++ to achieve object orientation and fast execution code. The implementation will to a large extent be described at an algorithm level, leaving out the details of how to implement them in a specific programming language.

3.1 Discretization

3.1.1 The computational grid

The numerical foundation of the simulation software will be Eulerian methods and thus we need to spatially discretize our simulation domain into a computational grid. In order to simplify the numerical schemes and thus reduce the overall computational cost we will define the grid cell to be a cube with the side $\Delta\delta = 1$, i.e. $\Delta x = \Delta y = \Delta z = 1$. This will greatly reduce the computational complexity of the overall simulation system. We will allow for different number of grid cells in each dimension but the sampling rate will be the same along each axis. Numerical values such as closest distance and velocities will be stored in the center of the grid cell.

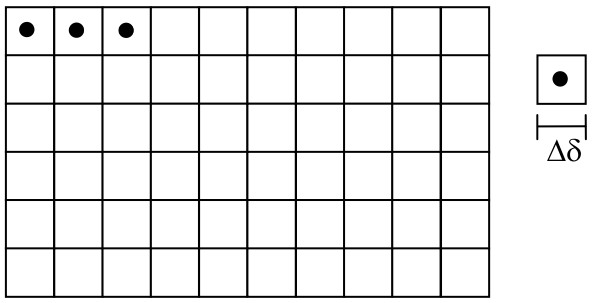


Figure 3.1 A 2D grid with values stored at the cell centers

Though we use the regular grid described above for our simulation system there exists a more advanced type of grid that has shown to work well for fluid simulation. This grid is known as a MAC grid and is described briefly in [9]. For this grid velocities is stored at the middle of each cell face instead of at the center. Distance information is however still stored as for the regular grid.

3.1.2 Spatial differentiation

The spatial differentiation operator for some function *f* in the direction *x* may be discretized as

$$D^{+} = \frac{\partial f}{\partial x} \approx \frac{f_{i+1} - f_i}{\Delta x} \tag{3.1}$$

$$D^{-} = \frac{\partial f}{\partial x} \approx \frac{f_i - f_{i-1}}{\Delta x} \tag{3.2}$$

$$D^{0} = \frac{\partial f}{\partial x} \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} \tag{3.3}$$

where D^+ is the forward difference, D^- is the backward difference and D^0 is the central difference. D^+ and D^- are first order accurate and D^0 is accurate to the second order. These operators will be used when execution speed is the most important factor.

When numerical accuracy is important we will instead use the Hamilton-Jacobi Weighted Essentially Non Oscillating scheme hereby referred to as HJ WENO. This is a weight optimized version of the third order accurate HJ ENO scheme and under ideal conditions it will be fifth order accurate. According to the HJ WENO scheme as described in [2] we may discretize the spatial differentiation operator as

$$D^{\pm} = w_1\left(\frac{v_1}{3} - \frac{7v_2}{6} + \frac{11v_3}{6}\right) + w_2\left(-\frac{v_2}{6} + \frac{5v_3}{6} + \frac{v_4}{3}\right) + w_3\left(\frac{v_3}{3} + \frac{5v_4}{6} - \frac{v_5}{6}\right) \tag{3.4}$$

Where, for D^+

$$v_1 = \frac{f_{i+3} - f_{i+2}}{\Delta x} \quad v_2 = \frac{f_{i+2} - f_{i+1}}{\Delta x}$$
$$v_3 = \frac{f_{i+1} - f_i}{\Delta x} \quad v_4 = \frac{f_i - f_{i-1}}{\Delta x}$$
$$v_5 = \frac{f_{i-1} - f_{i-2}}{\Delta x}$$

and for D^-

$$v_1 = \frac{f_{i-2} - f_{i-3}}{\Delta x} \quad v_2 = \frac{f_{i-1} - f_{i-2}}{\Delta x}$$
$$v_3 = \frac{f_i - f_{i-1}}{\Delta x} \quad v_4 = \frac{f_{i+1} - f_i}{\Delta x}$$
$$v_5 = \frac{f_{i+2} - f_{i-1}}{\Delta x}$$

The weights are calculated as

$$w_1 = \frac{a_1}{a_1 + a_2 + a_3} \qquad a_1 = \frac{1}{10}\frac{1}{(\varepsilon + S_1)^2}$$
$$w_2 = \frac{a_2}{a_1 + a_2 + a_3} \qquad a_2 = \frac{6}{10}\frac{1}{(\varepsilon + S_2)^2}$$
$$w_3 = \frac{a_3}{a_1 + a_2 + a_3} \qquad a_3 = \frac{3}{10}\frac{1}{(\varepsilon + S_3)^2}$$

where $\varepsilon = 1e^{-6}$ and the smoothness S is given by

$$S_1 = \frac{13}{12}(v_1 - 2v_2 + v_3)^2 + \frac{1}{4}(v_1 - 4v_2 + 3v_3)^2$$
$$S_2 = \frac{13}{12}(v_2 - 2v_3 + v_4)^2 + \frac{1}{4}(v_2 - v_4)^2$$
$$S_3 = \frac{13}{12}(v_3 - 2v_4 + v_5)^2 + \frac{1}{4}(3v_3 - 4v_4 + v_5)^2$$

3.1.3 Time integration

When solving time dependant PDE:s on the form

$$\frac{\partial f}{\partial t} = q(f)$$

we will need an operator that performs time integration. Time integration will be done explicitly using a third order accurate Runge-Kutta scheme. Advancing the function f Δt time-units forward in time will be done through the relation

$$f_{t+\Delta t} = D_t(f_t, q) = \frac{1}{3} f_t + \frac{2}{3}E\left(\frac{3}{4} f_t + \frac{1}{4}E(E(f_t))\right) \tag{3.5}$$

where

$$E(f_t) = f_t + \Delta t \cdot q(f) \tag{3.6}$$

3.1.4 The divergence operator

The divergence of a vector field $\boldsymbol{V}$ is calculated as $\nabla\cdot\boldsymbol{V}$. Using central difference the discreet divergence operator in three dimensions becomes

$$\nabla \bullet \vec{V}_{i,j,k} = \frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\Delta y} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{2\Delta z} \tag{3.7}$$

where *u*, *v*, *w* are the vector components.

3.1.5 The Laplace operator

The Laplace operator ∇^2 acting on the scalar field *u* can be discretized as

$$\nabla^2 u_{i,j,k} = \frac{u_{i+1,j,k} + u_{i-1,j,k} - 2u_{i,j,k}}{(\Delta x)^2} + \frac{u_{i,j+1,k} + u_{i,j-1,k} - 2u_{i,j,k}}{(\Delta y)^2} + \frac{u_{i,j,k+1} + u_{i,j,k-1} - 2u_{i,j,k}}{(\Delta z)^2} \quad (3.8)$$

3.1.6 Poisson equations

When solving the Navier-Stokes equations one will encounter some Poisson equations. In general Poisson equations can be written on the form

$$\nabla^2 u = v \quad (3.9)$$

where *u* is a scalar potential and *v* is a source function. Discretizing this equation using the discrete Laplace operator as described above will result in a linear equation system with one variable for each element in the scalar field *u*. For a 3D simulation grid with 100x100x100 voxels this means that we will have $1 \cdot 10^6$ scalar elements in *u* – one for each voxel. This will result in a gigantic 10^6 x 10^6 matrix to which we need to find the inverse in order to solve the equation system. Using regular methods for doing this is thus out of the question but since the matrix will be very sparse we may use a solver that can take advantage of this property. In our software a preconditioned conjugate gradient solver is used and the algorithm is only iterated until the largest residual element is less than 0.5. This has experimentally proven to be a good compromise between precision and speed. All matrices are stored efficiently in dynamic arrays of size $O(n^3)$ instead of $O(n^6)$ for the full matrix.

3.1.7 Boundary conditions for Poisson equations

When solving the Poisson equations encountered for free surface fluid simulation one will have to take two boundary conditions into consideration – the Dirichelt boundary condition and the Neumann boundary condition.

3.1.7.1 The Dirichlet boundary condition

The Dirichlet boundary condition is represented by the equation

$$\vec{v} \bullet \vec{n} = 0 \quad (3.10)$$

and simply states that there can be no flow into or out of the boundary surface to which ***n*** is the normal. In our case this will be solid objects. In the program this condition will be explicitly enforced by projecting all velocities neighboring a boundary surface onto the plane to which ***n*** is normal.

3.1.7.2 The Neumann boundary condition

The Neumann boundary condition represented by the equation

$$\frac{\partial \vec{v}}{\partial \vec{n}} = 0 \qquad (3.11)$$

states that there shall be no change of flow along the normal ***n*** of a boundary surface. This condition is an essential complement to equation (3.10).

Equation (3.11) can be enforced by eliminating the connection between solid boundary cells and fluid cells when building the linear equation system for the discreet Poisson equation. If we assume that the grid cell at position i,j,k contains fluid while the grid cell i-1,j,k belongs to a solid object we enforce the Neumann boundary condition by assuming

$$u_{i-1,j,k} - u_{i,j,k} = 0 \qquad (3.12)$$

when discretizing the Laplace operator for grid cell i,j,k.

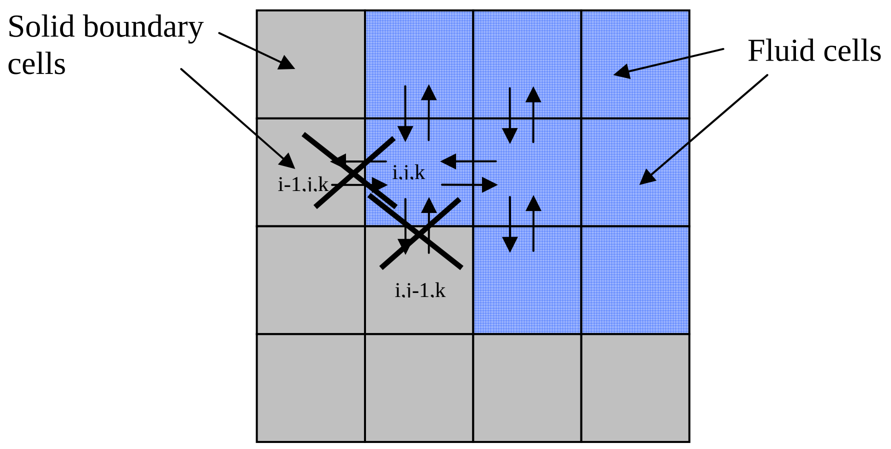


Figure 3.2 The Neumann boundary condition at a solid boundary

3.2 *Level set implementation*

The level set implementation is based on the work by Enright [2]. The level set is represented with a scalar field where every cell in the computational grid contains a scalar representing the closest distance from that point to the interface.

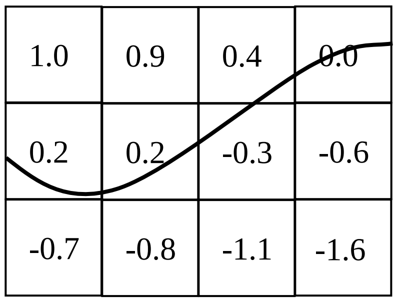


Figure 3.3 Interface representation using level sets (values are approximate)

Since the interface is implicitly represented by the distance information stored in our computational grid it is necessary to define the discreet inside set $\mathbf{\Omega}$. This will be done using the convention

$$\phi(\mathbf{x}) < 0.5 \cdot \Delta\delta \Longrightarrow \mathbf{x} \in \mathbf{\Omega}$$
$$\phi(\mathbf{x}) > 0.5 \cdot \Delta\delta \Longrightarrow \mathbf{x} \notin \mathbf{\Omega}$$
$$|\phi(\mathbf{x})| <= 0.5 \cdot \Delta\delta \Longrightarrow \mathbf{x} \in \partial\mathbf{\Omega}$$

$$\Delta\delta = 1$$

Thus, in essence, we regard a grid cell as being part of the interface-cell set if any part of the interface passes through the circle, or sphere in 3D, with radius 0.5 that is inscribed in the cell.

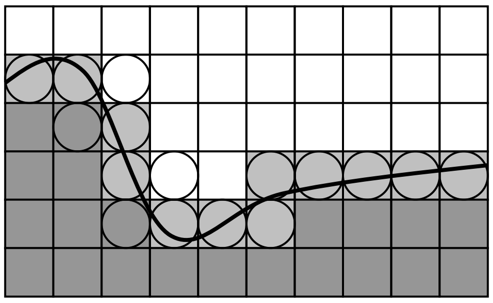


Figure 3.4 Cell classification for the discreet level set

In order to advect the level set we also need a velocity field. This is defined as a velocity vector at each grid point.

3.2.1 Local level set method

Solving the level set PDE:s in the full volume of the grid is extremely time-consuming but as shown by Peng et. al. [10] it is possible to use a localized representation of the level set where the equations are only solved in a narrow band around the interface. This is possible since only the interface cells and the cells in the direct proximity to the interface will be of interest when doing level set advection and we can thus focus on this region instead of the entire grid.

The exact width of this band will depend on the width of the differentiation kernel used by the numerical schemes. For first order we only need one grid cell on each side of the interface cell and for fifth order HJ WENO we will need three cells on each side. Instead of the function ϕ being a signed distance function for the entire simulation domain we now only require this inside a band with thickness γ:

$$\phi(\vec{x}) = \begin{cases} \gamma & \text{if } d(\vec{x}) > \gamma \\ d(\vec{x}) & \text{if } \left|d(\vec{x})\right| \leq \gamma \\ -\gamma & \text{if } d(\vec{x}) < -\gamma \end{cases} \tag{3.13}$$

where *d*(**x**) is the signed closest distance from **x** to the interface.

The values of γ used in the program depend on the accuracy of the scheme used. Both first and fifth order is available. For first order spatial difference we will need one cell on each side of the interface and for HJ WENO we will need three. Taking the cell containing the interface into account the band 'radius' γ will be

$$\gamma = \lceil 1.5 \rceil = 2 \quad \text{for first order spatial difference}$$
$$\gamma = \lceil 3.5 \rceil = 4 \quad \text{for fifth order HJ WENO}$$

The band structure itself is stored as a dynamic array containing the coordinates of all grid points that are within the band, from here on referred to as the *narrow band*.

As the interface moves it becomes necessary to update the narrow band. In order to do this an additional outer band will be needed to keep track of grid-cells that are to be added to, or removed from, the narrow band. For practical reasons this outer band need to be at least two grid cells thick for both first and fifth order methods resulting in a final band 'radius' of 4 and 6 grid cells respectively.

3.2.2 The particle level set

Though the implicit representation of the interface allows us to represent its position with arbitrary precision there is still a price to pay when discretizing the simulation space. The coarser the grid the more difficult it becomes for the level set to preserve regions of high curvature resulting in the erosion of sharp corners and fine features. It is also impossible to preserve surfaces that are thinner than the width of one grid cell. The two cases are shown in figure 3.5.

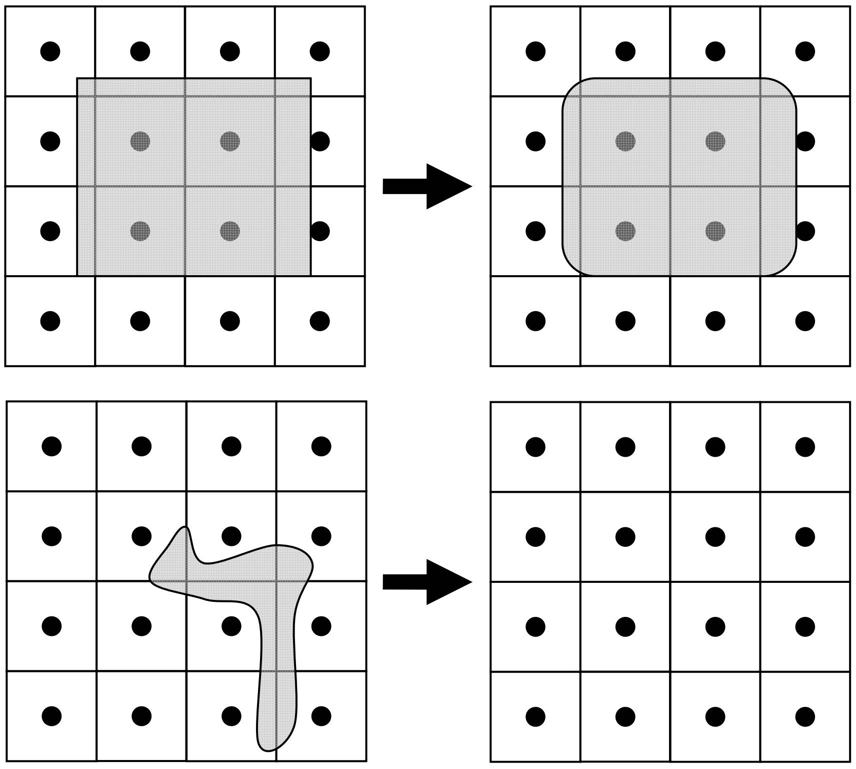


Figure 3.5 Errors when advecting the discreet level set

In [2] Enright propose the use of a hybrid level set method to lessen the impact of these problems. His idea is to use marker particles with no mass distributed around the interface to help preserve fine features. The idea is that each particle will have a signed radius that gives the closest distance to the interface from the position of that particle. This radius information will be kept as the particle is advected. The information about the position of the interface carried by the particles can then be compared to the position of the interface after it has been advected. Any errors can thus be identified and corrected since the particles will not have been affected by any smoothing effects native to the discreet level set representation.
The basic idea can be seen in figure 3.6

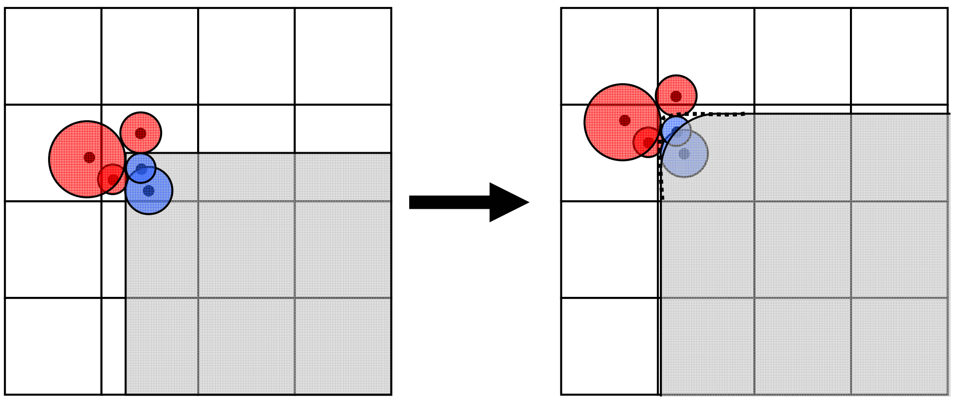


Figure 3.6 Particle correction of the level set interface

The red particles are positive (outside) particles and the blue are negative (inside) particles.

Correcting the level set using particle information is done along the lines suggested by Enright in [2]: A positive and negative corrected distance field ϕ^+ and ϕ^- is initiated with the damaged field ϕ and then a Boolean union operation is performedwith all the particles:

$$\phi^+ = \max(\phi^+, r_p) \ \forall \ r_p \geq 0$$
$$\phi^- = \min(\phi^+, r_p) \ \forall \ r_p < 0$$

where r_p is the signed radius of the particle. Enright then suggests that the two representations should be merged using the value of ϕ^+ or ϕ^- depending on which has the least magnitude. This did not work very well in our implementation, probably due to the different way particles are seeded and reseeded as described in section 3.2.3. Instead the mean value

$$\phi = \frac{\phi^+ + \phi^-}{2} \tag{3.14}$$

is used.

The particles are implemented in a dynamic array and lives in free space. The connection between particles and the grid is made by transforming the particle coordinates from the global coordinate system to grid coordinates.

3.2.3 Seeding and re-seeding particles

The main practical problem with the particle system is that we need to create and maintain a uniform distribution of particles around the interface. Since regions of the level set may stretch and contract we must continuously monitor and maintain the target particle density in each grid cell. Too many particles in any cell is a waste of memory and processing power and too little means we loose the effectiveness of the particle correction.

3.2.3.1 Seeding new particles

We create the particle band around the interface by assigning a particle density *k* to each cell in the narrow band according to the distribution

$$\rho_p\left(\vec{x}\right) = \begin{cases} k & \forall \vec{x} : \left|\phi\left(\vec{x}\right)\right| \leq \beta \\ 0 & \forall \vec{x} : \left|\phi\left(\vec{x}\right)\right| > \beta \end{cases} \tag{3.15}$$

We then loop through the particles and count how many of them there are in each cell in the narrow band. For all cells that have too few particles we randomly seed new ones until (3.15) is satisfied for all cells in the narrow band. Each seeded particle is assigned a signed radius that is equal to the signed distance from the position of that particle to the interface. This allows the particle to explicitly represent information about the position of the interface and allows reconstruction of damaged parts of the level set during the particle correction step. The radii used are the same as suggested in [2]:

$$r_p = \begin{cases} sign\left(\phi\left(\vec{x}_p\right)\right) \cdot r_{\max} & \text{if } \left|\phi\left(\vec{x}_p\right)\right| > r_{\max} \\ \phi\left(\vec{x}_p\right) & \text{if } r_{\min} \leq \left|\phi\left(\vec{x}_p\right)\right| \leq r_{\max} \\ sign\left(\phi\left(\vec{x}_p\right)\right) \cdot r_{\min} & \text{if } \left|\phi\left(\vec{x}_p\right)\right| < r_{\min} \end{cases} \tag{3.16}$$

$$r_{\max} = 0.5$$
$$r_{\min} = 0.1$$

where *sign*($\mathbf{x}_p$) is the sign function

$$sign(x) = \begin{cases} 1 & \forall x > 0 \\ 0 & x = 0 \\ -1 & \forall x < 0 \end{cases} \tag{3.17}$$

If a seeded particle happens to be placed closer to the interface than the distance r_{min} the particle is re-seeded until it can be placed at a distance greater than r_{min}. If no acceptable coordinate has been found for the particle after five attempts it is discarded and we allow the cell have less particles than the target density.

3.2.3.2 Treatment of existing particles

In spite of the particles continuously feeding distance information back to the level set there will be a difference in the positive and negative corrections ϕ^+ and ϕ^-. This will result in the particle radii not being accurate for all particles after the correction is done. This effect is unwanted and is removed by feeding distance information back from the corrected level set to the particles by changing the particle radii so that the particle representation once again is true to the current implicit interface representation of the level set. Since the minimum radius of the particles have been restricted but not the minimum distance of each particle to the interface with the exception of the newly seeded particles we may have particles that are closer to the interface than their radius. The errors introduced by this is not a big issue since the smoothing operation of (3.14) will result in an interface that still is located approximately at the right position.

As stated above the particles will help preserve fine features and thin interfaces for any level set. But for a level set representing a region of liquid it will prove very useful if the interface never becomes thinner than what can be represented by the level set, i.e. one grid cell thick. Though this will impair the ability of the liquid to split apart it will greatly help preserve thin interfaces and thus improve overall visual appearance. One should be aware however that this is at the expense of simulation accuracy with regard to physics. Since we are primarily interested in simulations that are visually pleasing and not necessarily true to nature this is a price we are willing to pay. This goal will however introduce one problem since it means that two of our requirements on the liquid will be conflicting in regard to how the particles are to behave: We want two parts of the liquid to merge if they collide and we also want thin sheets of liquid to be preserved as well as possible with the aid of the particles. The two cases are exemplified in figure 3.7.

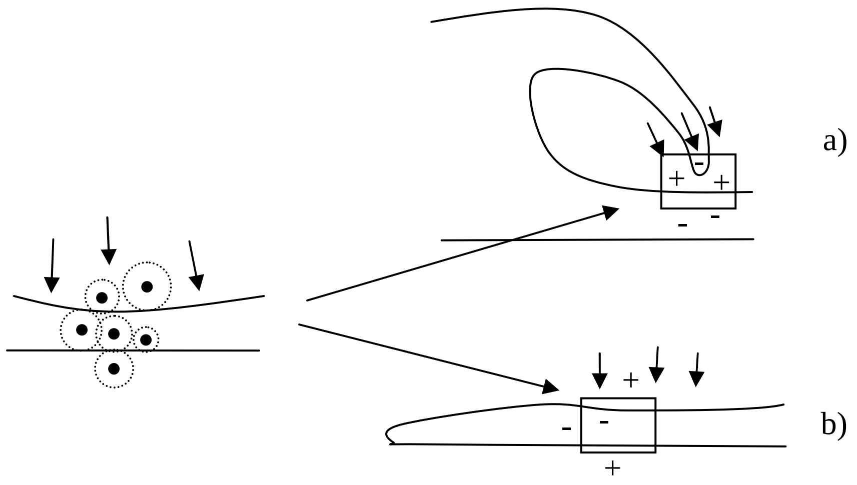


Figure 3.7 Special cases when doing particle correction

The principal case to the left shows two interfaces that are close together and moving towards each other with particles on both sides of the interface. As shown to the right there are two principal situations where this arises – when two parts of the level set is about to merge as shown in a) and when a thin sheet of liquid is being made too thin to represent by the grid as shown in b). Situation b) is the one where we want the particles to force the interfaces to stay apart thus keeping the liquid interface from being too thin. In situation a) however we want the two parts of the level set to merge which means that we want to allow the two approaching interfaces to collide. The solution to this problem is based on the sign of the particles. As can be seen in a) there will be positive particles, i.e. particles outside the liquid, in between the two interfaces and in situation b) the squeezed particles will be negative.

When we update the radius of the particles we can get the effect we want by treating positive and negative particles differently: If a positive particle appears on the negative side of the interface of the corrected level set by more than r_{min} we will change the sign and radius of the particle to represent its new position. This situation will arise if two parts of the level set has collided as in a). If on the other hand a negative particle appears on the positive side of the interface we will regard this as situation b) and not allow the particle to change its sign. We will however still change the radius making the particle larger the further on the "wrong" side it is, thus increasing the correctional effect. The mean value representation calculated by (3.14) will then guarantee that we get a tendency to push the interfaces apart as exemplified in figure 3.8.

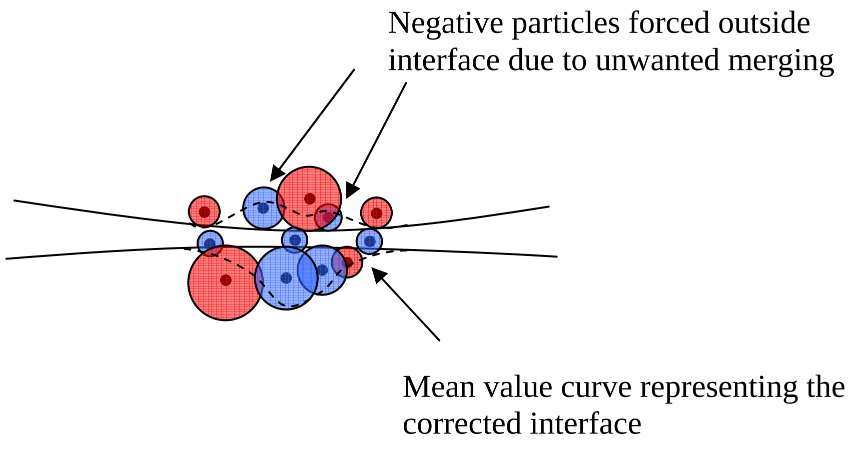

Negative particles forced outside interface due to unwanted merging
Mean value curve representing the corrected interface

Figure 3.8 Thin interface preservation prioritizing negative particles.

If, in spite of the effect described above, a negative particle for some reason moves a distance greater than r_{max} into the positive region it is regarded as an indication that the level set wants that region to disappear and we will allow the particle to change sides. This may for example be the case if the driving velocity field wants to split up a region of the level set. It will also help preserve the integrity of the particle representation since we will not allow negative particles drifting too far into the positive region of the level set.

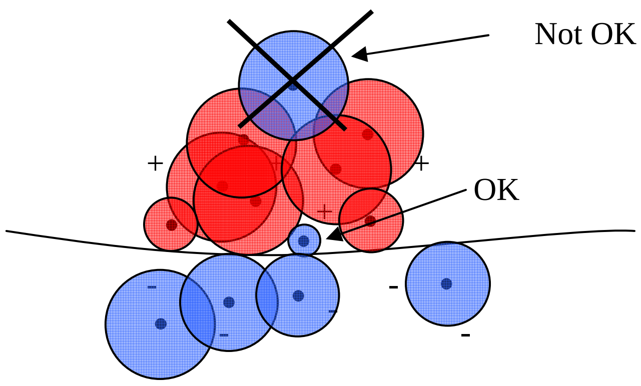


Figure 3.9 Condition for allowing negative particles to change side of interface

3.2.3.3 Deleting particles

If a region develops where there is a higher particle density than the target density *k* we will have to remove some particles. The deletion of particles should however be done with great care since they carry information about the correct position of the interface, information that is lost when the particle is deleted. In [2] Enright solves this problem by deleting the particles in radius order, starting with the ones that are furthest away from the interface and thus has the largest radius. He uses a max-heap to accomplish this. We will however use our own method for achieving this selective deletion of particles: For each cell in the narrow band we calculate a condition number ξ according to the formulae

$$\xi(n_p) = \begin{cases} r_{\min} & \text{if } n_p > 2 \cdot \rho_p \\ r_{\min} + \dfrac{r_{\min} + r_{\max}}{2} \cdot \dfrac{2\rho_p - n_p}{\rho_p} & \text{if } n_p \in \left[\rho_p, 2\rho_p\right] \\ \infty & \text{if } n_p < \rho_p \end{cases} \qquad (3.18)$$

where n_p is the number of particles in the cell. As we update r_p for the particles we simply compare its radius r_p to ξ and if $r_p \geq \xi$ we delete the particle. This is done until $n_p = \rho_p$ or all particles have been visited once. This means that we, in extreme cases, may allow up to $2\rho_p$ particles in cells where all the particles are very close to the interface and thus regarded as very important. In reality this situation is rather unlikely and the particle density will be close to ρ_p but without the need for heap sorting. Our method has the advantage of being faster than heap sorting as long as the simulation is well behaved so that the excessive amount of particles in each cell is small. It also provides a more dynamic particle density that depends on how close the particles are to the interface.

3.2.4 Advection

Advecting the level set is done by solving the "level set equation" (2.9). In order to do this we need to calculate the normal ***n*** according to equation (2.5). This is done by using an upwind differential scheme for the spatial differentials to take advantage of the direction of information flow. Up-winding will increase stability and improve the behavior of the solution since it prioritizes the differential direction from which information that affects the current simulation cell originates. This simply translates to choosing the directional differential D^+ or D^- depending on the direction of the velocity vector $\boldsymbol{v} = (v_x, v_y, v_z)$ at each grid point. If a component of the vector is positive we use backward-difference for that component, otherwise we use forward difference. In pseudo-code this step can be written as:

```
for (n = x,y,z)
  if $v_n$ >= 0
    use $D^-_n$
  else
    use $D^+_n$
```

Here the spatial differentials can be either first order or fifth order depending on user demand. For time integration we use the third order Runge-Kutta method (3.5). When the normal is known propagation of the interface from time *t* to *t* + *Δt* is done applying the discretized level set equation

$$\phi(\vec{x})_{t+\Delta t} = D_t(-\vec{n} \bullet \vec{v}(\vec{x})) \tag{3.19}$$

to each point in the narrow band.

This solution algorithm is unfortunately not unconditionally stable with regard to the time-step Δ*t*. It can however be shown that stability can be ensured using the Courant-Friedreichs-Lewy (CFL) condition:

$$\Delta t = \frac{\alpha}{\max\left(\frac{|v_x|}{\Delta x}, \frac{|v_y|}{\Delta y}, \frac{|v_z|}{\Delta z}\right)} \tag{3.20}$$
$$\alpha \in]1,0[$$

In essence the CFL condition states that the interface must move less than one grid cells in any direction for each update cycle of the level set (for α = 1). In the program we use α = 0.5 which in our case proved to be a good compromise between stability and performance. Using this and the earlier convention of Δx = Δy = Δz = 1 equation (3.20) can be rewritten as

$$\Delta t = \frac{0.5}{\max(|v_x|,|v_y|,|v_z|)} \tag{3.21}$$

It should be noted that the CFL condition may be avoided if a semi-Lagrangian method such as the one used when solving the self-advection term of the Navier-Stokes equations is used to achieve the advection of the level set.

When advecting the level set we also advect the particles used for correcting the interface position. This is done using Newtonian mechanics for a zero mass particle by solving the equation

$$\frac{\partial \vec{x}}{\partial t} = \vec{v}(\vec{x}) \tag{3.22}$$

in every cell containing particles. Discretized this becomes

$$\vec{x}_{t+\Delta t,n} = D_t(\overrightarrow{x_{t,n}}, \vec{v}_n(\vec{x})) \tag{3.23}$$
$$n = x, y, z$$

thus producing one equation for each component of ***x*.** Since particle positions are in machine precision while velocity vectors are only defined in each simulation cell we will need to do interpolation to find the velocity ***v*** at the particle position ***x***. In the program we will use tri-linear interpolation for this.

3.2.5 Re-initialization

As the level set is propagated forward in time it will loose the property of an Euclidian distance field. This is mainly caused by the velocity field of the fluid advecting different parts of the level set by different amounts. This is unacceptable since many equations and relations are dependant on $|\Delta\phi| = 1$ to be accurate. To correct this we do a re-initialization step after each advection of the level set. The purpose of this is to restore the level set to an Euclidian distance field once again.

This non-trivial problem can be solved using the re-initialization equation

$$\frac{\partial \phi}{\partial \tau} + S(\phi_0) \cdot \left(|\nabla \phi| - 1 \right) = 0 \tag{3.24}$$

$$S(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + |\nabla \phi|^2}} \tag{3.25}$$

assuming $\Delta x = \Delta y = \Delta z = 1$. As suggested in [9] we use a smeared out sign function *S* instead of a discreet sign function to get better numerical results.

Equation (3.24) is closely related to the level set equation (2.9) but this time we propagate the level set using a function that forces it to regain the property of an Euclidian distance field. To solve equation (3.24) we use upwind differentiation in the direction of the interface when calculating $|\nabla\phi|$ in accordance to Godunov's scheme as described in [9]. This means we will prioritize information in the interface direction which in turn results in information propagating outwards with the interface as boundary condition. For each time step of the fictious time τ ϕ will be forced towards an Euclidian distance field which will mean that $|\nabla\phi|$ will approach 1. When $|\nabla\phi| = 1$ equation (3.24) is reduced to

$$\frac{\partial \phi}{\partial t} = 0$$

which means that we have reached a steady state where the level set does not change.

As when solving equation (2.9) there is a CFL condition involved. In our case the re-initialization time step

$$\Delta \tau = 0.5$$

has proven to work well. If the level set is far from an Euclidian distance field a more conservative time-step may be needed. Numerically this means that information will be propagated outwards from the interface at a rate of 0.5 grid cells per iteration of the solver.

Thus we need to iterate the solution algorithm to equation (3.24)

$4/\Delta\tau = 8$ times if first order spatial differentials are used
$8/\Delta\tau = 12$ times if HJ WENO differentials are used

in order to be sure that the entire level set is re-initialized.

The main disadvantage of this solution is that the interface may move slightly as equation (3.24) is iterated towards steady-state. To reduce this phenomenon we do a particle correction step before the first two re-initialization cycles.

Another thing that needs to be addressed in conjunction with re-initialization is updating the narrow band. Since the interface has moved during the advection step the band now becomes too thin on some places and too thick on others as shown in figure 3.10.

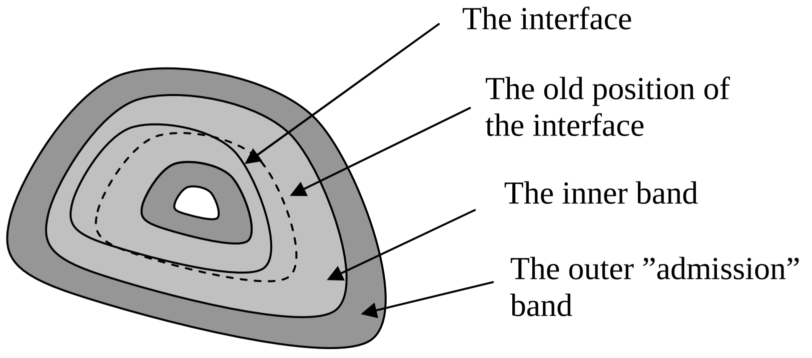
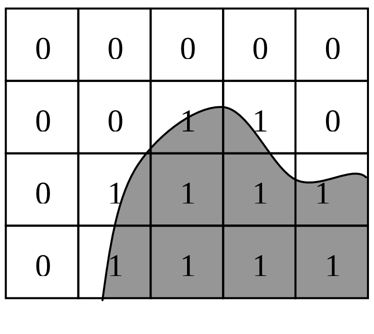


Figure 3.10 Errors to the band structure after interface advection

We know however that the interface has at most moved one grid cell due to the advection CFL condition. Using the distance information stored in the grid cells in the outer band we can now remove the band-cells that contain too large distance values and thus are regarded as outside the band structure. Admitting new cells to the band is a bit harder but can be done storing the band ID in each grid cell. ‘0’ means that the cell is not part of the band structure. We can now identify cells that shall be added by finding those cells in the outer band that has a distance smaller than the bandwidth -1 and that has a neighboring cell that is not part of the band.



0	0	0	0	0
0	0	1	1	0
0	1	1	1	1
0	1	1	1	1

Figure 3.11 Classification data structure to keep track of cells included in the band structure

The algorithm for this will be:

```
For (all cells in the outer band)
  If (distance > outer band width)
    Remove cell from band
  Else if (distance < outer band width – 1 and
            neighboring cell is not part of band structure)
    Add neighboring cell to band structure
```

This allows us to keep the width of the band structure constant. Next we also need to update which cells belong to the inner respective the outer band. To do this we now re-initialize the entire band structure, including the newly added cells. We can then look at the distance each cell has to the interface to rebuild the inner and outer band according to.

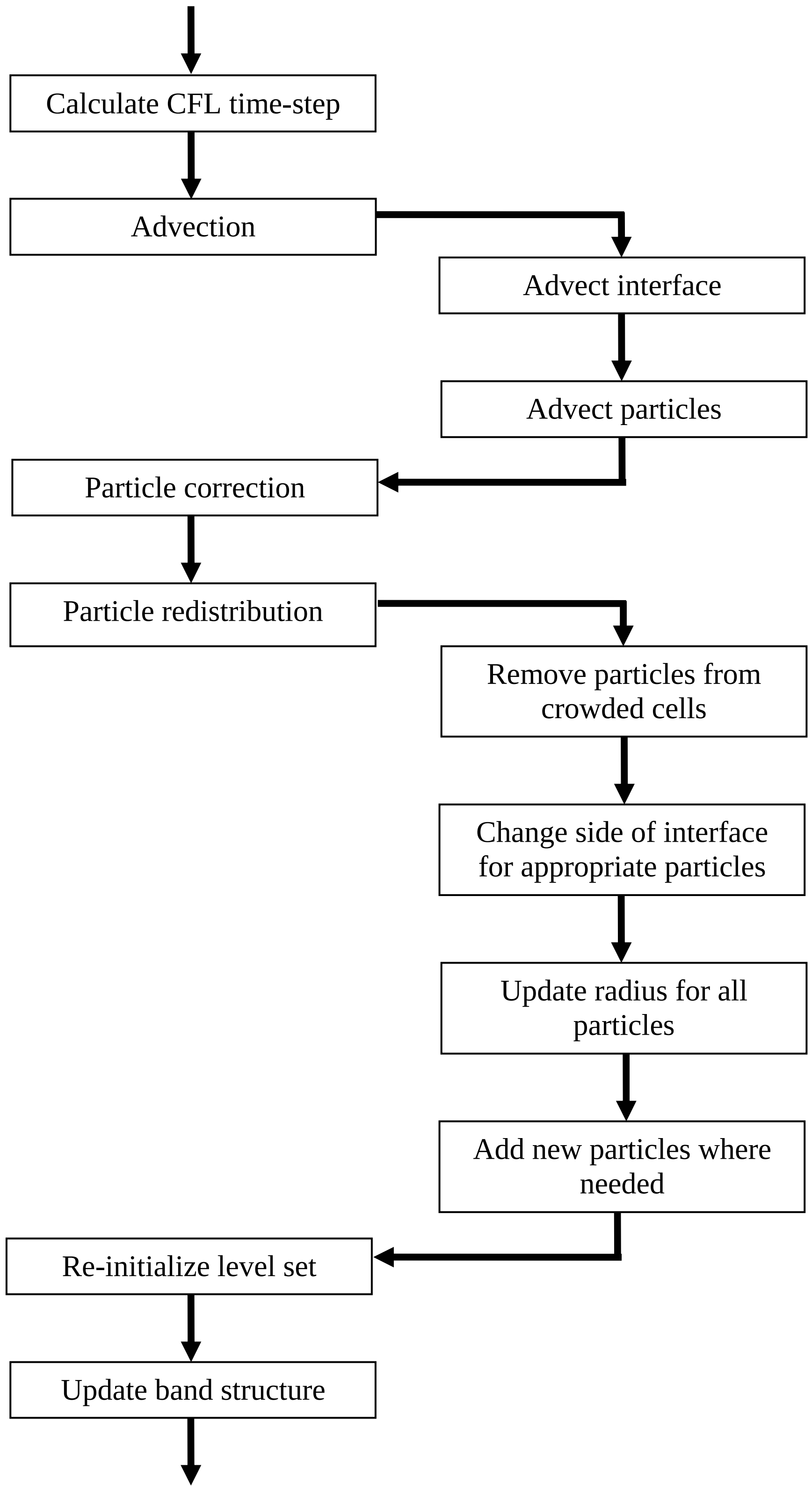
```
If (distance < inner band width)
  cell belongs to inner band
else
  cell belongs to outer band
```

3.2.6 The inside set

Since our level set is to be used as surface representation in a fluid simulation we need to add an additional set of cells representing all grid cells that contain fluid. This is the inside set. The set is initiated by doing a flood-fill of the region inside the interface and is then maintained as follows:
If, when advecting the level set, a cell gets its distance value changed from $\phi(\boldsymbol{x}) > 0.5$ to $\phi(\boldsymbol{x}) <= 0.5$, corresponding to the interface entering the cell, this cell is regarded as containing fluid and is added to the inside set. We then loop through the inside set and remove all cells that no longer contain fluid, i.e. all cells for which $\phi(\boldsymbol{x}) > 0.5$. As with the band structure the inside set is stored in a dynamic array.

3.2.7 The level set update cycle

The overall structure of the level set implementation can be seen in the following flow diagram describing one update cycle of the level set:


Calculate CFL time-step
Advection
Advect interface
Advect particles
Particle correction
Particle redistribution
Remove particles from crowded cells
Change side of interface for appropriate particles
Update radius for all particles
Add new particles where needed
Re-initialize level set
Update band structure

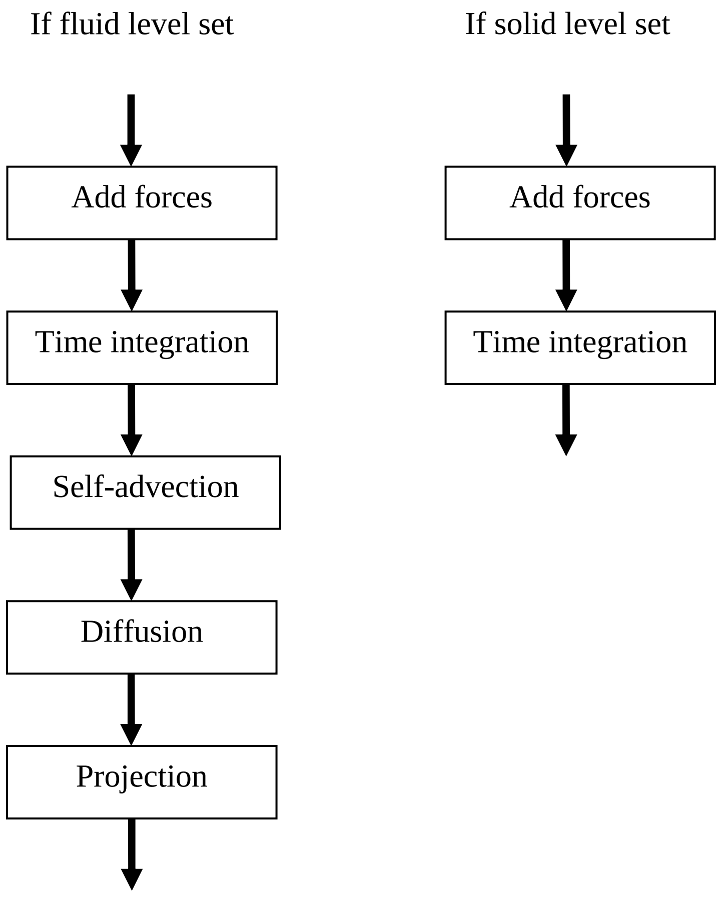
3.3 Multiple level sets

There will be at least two different types of level sets used by our program. One fluid level set keeping track of the fluid interface and one solid boundary level set that represents walls and obstacles. One could also imagine having a number of moving solid objects in which case each of those will also need to be represented by a level set.

To handle this we will add object properties to each level set containing information describing the level set. The most important of these properties will be ‘solid’, ‘fluid’ and ‘stationary’. A level set may be solid or fluid and a solid level set may or may not be stationary. Stationary objects will not be affected by any forces and will act as solid boundaries.

3.4 The Navier-Stokes solver

Using the solution method outlined in section 2.5 with the added ability to handle simple solid object physics the Navier-Stokes update cycle used to advance the velocity field forward one time step will look as follows:



The solver will take an array of level sets and update the velocity fields associated with each level set. Since the Navier-Stokes solver is based on the stable fluids approach of [12] we need not worry about the size of the time-step. For simplicity the same time-step as that of the level-set will be used, though it is pretty straight-forward to allow two or three level set updates for each Navier-Stokes cycle. The practical limitation is set by the width of the narrow band since we will need correct velocities to address during the particle trace involved in solving the self-advection term.

3.4.1 Simplifications and approximations

The goal of the simulation system is to produce realistic and visually pleasing but not necessarily physically accurate fluid simulations. This compromise makes it possible for us to do a number of simplifications to the simulation engine, the most important of which are discussed in more detail below.

3.4.1.1 No air simulation

Only the region containing fluid is simulated. Though the region outside the fluid is supposed to represent air no simulation of this medium is done and we will refer to this medium as *void*. This will mean that there are no forces that limit the motion of the fluid into this empty domain. In general the visual effect of this approximation is very small but there is one case where the effect is apparent: The formation of bubbles.

If a region of the void, gets enclosed inside the fluid one would expect a bubble to form as the captured air has nowhere to go. But since the air is not simulated it will exact no pressure on the surrounding fluid, thus exact no forces that prevent the bubble from getting consumed in a way that clearly defies expected physics. Fortunately this effect is very seldom apparent and may go unnoticed if one does not look for it.

3.4.1.2 No surface tension

The Navier-Stokes equations used in our simulation software do not include the viscous stress tensor. The most apparent effect of this approximation is the loss of surface tension effects. If this will be of consequence or not depends on the scale of the simulation. The most prominent visual effect of surface tension is the formation of droplets. The lack of this effect will only be apparent for small scale simulations since the droplets will be too small to see in large scale simulations and can then be modeled by particle systems or visual effects added to the general motion of the fluid.

3.4.1.3 Incompressible fluid

Another simplification to the Navier-Stokes equation is incompressibility. This means that the pressure inside the fluid is assumed to be constant. This will mean that pressure waves inside the liquid cannot form. The visual impact of this is however insignificant for most everyday fluids. The bulk modulus for water for example is ≈ $2.2 \cdot 10^9$ N/m^2 which comes close to that of steel (≈ $160 \cdot 10^9$ N/m^2).

Since the density will be constant across the fluid the mass of a fluid filled grid cell will be constant and we will hence forth treat all grid cells containing fluid as having mass 1. This is another approximation since it will mean that cells only partially containing fluid will still be treated as having mass 1 and thus the momentum carried by the fluid of that cell will be represented as higher than it should be.

3.4.2 Velocity extension

In order for us to be able to advect any level set along a velocity field we must guarantee that the velocity field is defined at least in the narrow band. Unfortunately this is not the case with the field generated by the Navier-Stokes solver. It is only able to calculate velocities for simulation cells that contain fluid which translates to all cells on or inside the interface. This means we must find some way to extend the velocity field to the outside of the interface so that the level set will be moved properly. A way to do this is using velocity extension as described in [9]. The idea is to propagate velocity information out in the normal direction from the interface the same way as distance information is being propagated during the level set re-initialization. To do this we use the general equation for extrapolating some characteristic *S* in the direction of the normal ***n***:

$$\frac{\partial S}{\partial \tau} + \vec{n} \bullet \nabla S = 0 \tag{3.26}$$

In our case *S* will be a velocity vector leading to the corresponding equation for extrapolating the velocity field ***V*** in the direction of a normal field ***N***:

$$\frac{\partial \vec{V}(\vec{x})}{\partial \tau} = -\vec{N}(\vec{x}) \bullet \nabla \vec{V}(\vec{x}) \tag{3.27}$$

It should be noted that this equation only extrapolates velocities into the outside region of the level set, but this is what we want since we already have correct velocities obtained from the Navier-Stokes equations on the inside. If one wants to extrapolate velocities in both directions this is easily accomplished by adding the sign of the distance field in front of the right hand side (RHS) of equation (3.27).

Once again we will use the information on the interface as boundary condition, thus extrapolating the velocity field on the interface outwards along the normal field provided by the level set. In order to successfully do this we need to use upwind differentiation. In the program upwind differentials are used both when calculating the normal according to equation (2.5) and when calculating the gradient of the velocity field in equation (3.27) according to the following scheme:

We loop through all cells in the band structure that has $\phi > 0.5$, i.e. all cells in the band that does not contain fluid. We choose to differentiate in the direction of smallest distance, i.e. the direction of the closest interface:

```
If ($\phi(x+1) < \phi(x)$ and $\phi(x-1) < \phi(x)$)
  If ($\phi(x+1) > \phi(x-1)$)
    Use $D_x^+$
  Else
    Use $D_x^-$
Else if ($\phi(x+1) <= \phi(x)$)
  Use $D_x^+$
Else if ($\phi(x-1) < \phi(x)$)
  Use $D_x^-$
Else
  Direction undeterminable, no information should flow into this point. Set all differentials to
  zero.
```

This example is along the x-axis but the algorithm looks the same for the y- and z-axis.

After calculating the normal vector and the velocity gradient matrix at each cell we can calculate the RHS of equation (3.27) and use the third order Runge-Kutta scheme to update the field one time-step. As for the re-initialization equation we iterate these steps until the RHS of equation (3.27) approaches zero and we have reached a steady state. In the program a cutoff value of 0.1 is used.

When solving equation (3.27) one must also take a CFL condition into account. As for re-initialization 0.5 works fine, but in the program the more aggressive time-step of 0.7 is used to get faster convergence.

3.4.3 Forces and scaling

Forces are represented by a force field. Though an arbitrary function can be used to generate this field we only use the gravitational force in our simulations. The force field will be used to update the initial velocity field $\boldsymbol{w}_t$ to $\boldsymbol{w}_1$ using equation (2.17) where we once again use the third order accurate Runge-Kutta scheme when doing time integration:

$$w_1(\vec{x}) = D_t\left(w_t(\vec{x}), F(\vec{x})\right) \tag{3.28}$$

The effect of scale will also need to be represented by changes to the force term. This may not seem intuitive at first but is an effect of our choice to define each grid cell as being unit size, i.e. $\Delta x = \Delta y = \Delta z = 1$. A simulation domain consisting of 100 grid cells in each direction will thus represent a cube with the side 100 meters if we define the unit distance to be 1 meter. A unit distance of 3.15 cm will create a cubical simulation domain with the side 3.15 meters etc. This means that the scaling will be naturally represented in all grid-bound equations due to the nature of the grid. The only thing needed to get the physics right is a modification of all units that depend on distance. The unit of force is Newton [N] = [kg·m/s^2] which is distance dependant. Thus if we redefine the unit distance we must compensate for this in the force unit. We introduce the scale-factor α such that the side of each cell in our simulation domain is α meters. This means that we define our local distance unit $[m_{sim}] \equiv [\alpha \cdot m]$ and as a consequence the force unit will change as follows:

$$
\begin{aligned}
[N] &= \frac{kg \cdot m}{s^2} = \frac{1}{\alpha} \cdot \frac{kg \cdot \alpha \cdot m}{s^2} = \frac{1}{\alpha} \frac{kg \cdot m_{sim}}{s^2} = \frac{1}{\alpha}[N_{sim}] \\
[N] &= \frac{[N_{sim}]}{\alpha}
\end{aligned}
\qquad (3.29)
$$

Take the gravitational force exacted on an object with mass *m* for example:

$$F_G = m \cdot G[N] = \frac{m \cdot G}{\alpha}[N_{sim}]$$

As can be seen we need to scale all forces by dividing with α, i.e. the unit size of the simulation cell. Scaling will also affect a few other units used in the simulation engine and we will point them out as they are encountered.

3.4.4 Self-advection

The self-advection term will update the intermediate velocity field $\boldsymbol{w}_1$ to $\boldsymbol{w}_2$. This step is done along the lines of the stable fluid approach by Stam as described in [12] and section 2.5.2. Given the old velocity field $\boldsymbol{w}_1$ we do a particle trace backwards through the field from the center of each grid cell containing fluid. We do this using the third order Runge-Kutta integrator to advect a zero-mass particle backwards in time from the center of the current cell to the position it would have had Δt time units ago. We then use tri-linear interpolation to estimate the velocity vector at that point in space and assign that velocity to the cell in $\boldsymbol{w}_2$ that is on the original coordinates of the zero-mass particle. In pseudo-code the algorithm will look like this:

```
For (each cell containing fluid)
  $\boldsymbol{x}_p$ = coordinate of current cell center
  $\boldsymbol{v}_p = \boldsymbol{w}_1(\boldsymbol{x}_p)$

  $\boldsymbol{x}_{new}$ = RungeKuttaIntegration($\boldsymbol{x}_p, -\Delta t, \boldsymbol{v}_p$)
  $\boldsymbol{v}_{new}$ = FindVelocityAtCoordinate($\boldsymbol{x}_{new}$)
  $\boldsymbol{w}_2(\boldsymbol{x}_p) = \boldsymbol{v}_{new}$
```

3.4.5 Diffusion

The diffusion step consists of solving the Poisson equation (2.19). Using first order time discritization this equation translates to

$$\frac{\vec{w}_3 - \vec{w}_2}{\Delta t} = \nu\nabla^2 \vec{w}_3 \Rightarrow$$
$$\left(\boldsymbol{I} - \nu\Delta t\nabla^2\right)\vec{w}_3 = \vec{w}_2 \qquad (3.30)$$

where ν is the viscosity and $\boldsymbol{I}$ is the identity matrix. By discretizing the Laplace operator as described in section 3.1.5 three sparse linear equation systems are obtained, one for each component of the velocity vector. The equation systems are solved using a preconditioned conjugate gradient solver.

3.4.6 Projection

As discussed in section 2.5 our solution method to the Navier-Stokes equations depends on a projection operation that will project the velocity field $\boldsymbol{w}_3$ onto its divergence free part $\boldsymbol{w}_{t+\Delta t}$, completing the Navier-Stokes update-cycle. This operator is realized by first applying equation (2.15) to the intermediate field $\boldsymbol{w}_3$

$$\nabla^2 q = \nabla \bullet \vec{w}_3 \qquad (3.31)$$

Where the divergence $\nabla \bullet \boldsymbol{w}_3$ is calculated using the discreet divergence operator (3.7). These divergence values will become the RHS of the Poisson equation and will thus act as the source function. The field q that satisfies equation (3.31) will be the divergence component of $\boldsymbol{w}_3$. This field is initiated to zero for the entire simulation domain. This actually translates to setting the atmospheric pressure to zero but this is not an issue since the only thing of importance when simulating incompressible fluids is that the pressure is constant. Thus the actual numerical value of the pressure can be chosen arbitrarily. Equation (3.31) is solved by using the discreet Laplace operator and then using a preconditioned conjugate gradient solver on the resulting equation system. After solving the Poisson equation the calculation of the final velocity field $\boldsymbol{w}_{t+\Delta t}$ becomes straight-forward using the relation

$$\overrightarrow{w_{t+\Delta t}} = \overrightarrow{w_3} - \nabla q \qquad (3.33)$$

that is derived from equation (2.13).

3.4.7 Moving solid objects

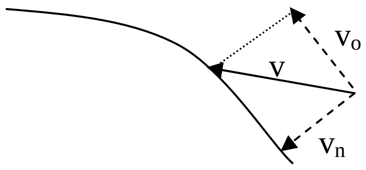
Moving solids are handled using the ghost cell method as described in [9]. This is done by taking into account the velocity of neighboring boundary cells when calculating the divergence $\nabla \bullet \boldsymbol{w}_3$ in equation (3.32).

3.4.8 Surface friction

Surface friction is modeled by modifying the Dirichlet boundary condition. By introducing a friction coefficient $\mu \in [0,1]$ we may choose how much of the linear momentum bound in the velocity component parallel to the surface normal that is to be preserved. Linear momentum is directly dependant on velocity and we may model friction by only looking at the velocities. Assume that the original velocity of the fluid in the cell is ***v***. Then the relation

$$\left|\vec{v}\right| = \sqrt{\left|\vec{v}_p\right|^2 + \left|\vec{v}_n\right|^2}$$

describes the relation between the original velocity magnitude and the magnitude of the velocity components parallel and orthogonal to the surface normal.



By setting the magnitude of the orthogonal component $\boldsymbol{v}_{\mathrm{o}}$ to

$$\vec{v}_o^{\,new} = \frac{\vec{v}_o}{\left|\vec{v}_o\right|} \cdot \sqrt{\left|\vec{v}_p\right|^2 + (1-\mu)^2 \cdot \left|\vec{v}_n\right|^2} \qquad (3.33)$$

μ can be used to determine how much of the magnitude of $\boldsymbol{v}_{\mathrm{n}}$ that is to be transferred to the orthogonal vector $\boldsymbol{v}_{\mathrm{o}}$. If $\mu = 1$ all velocity magnitude and thus all momentum bound in $\boldsymbol{v}_{\mathrm{n}}$ is lost. If $\mu = 0$ the final length of $\boldsymbol{v}_{\mathrm{o}}$ will be equal to ***v*** and all momentum is preserved.

3.4.9 Sources, sinks and volume conservation

We have created a system for adding sources and sinks to the simulation. This is done in the projection part of the solver by adding the size of the source to the component of the RHS of equation (3.31) that corresponds to the cell in which the source resides. This will create a bias for that cell when doing projection, thus allowing for a specified amount of divergence to be present in that cell. The unit for the source is [m^3/s] and is thus distance dependant. This means that we need to take scale into account resulting in the scale accurate unit [m^3/s]/α^3.

A novel feature of our simulation system is the ability to use forced volume conservation. One of the largest visual errors associated with the method used for our simulations is the loss off volume for the simulated fluid. There are several reasons for this unwanted phenomenon. First of all volume is lost due to the smoothing effect of the level set as described in section 3.2.2. Though the particle system compensates for this it is unable to completely remove the effect and over time the small errors will start to add up and become visible as lost volume. The Navier-Stokes solver is also responsible for some volume loss. This is due to the approximate solution to the projection operator, resulting in a velocity field that is not completely divergence free. In general this will result in mass loss mainly due to gravity trying to force fluid into the floor. The explicit enforcement of the Dirichlet boundary condition will guarantee that no velocity vector can point into a solid object, but if the projection operator is unable to completely adapt the rest of the velocity field to this change small sinks will form at the floor boundary as shown in figure 3.12.

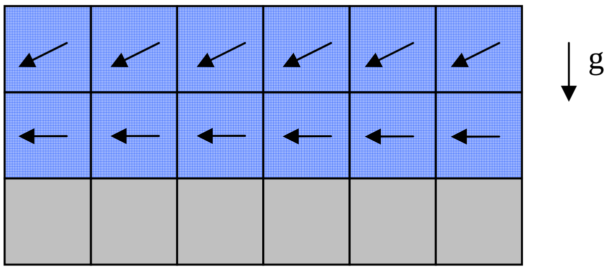


Figure 3.12 Sinks created in the velocity field if the projection operator is not resolved to infinite precision

To compensate for lost mass we will keep track of the theoretical volume V_0 and compare it to the actual volume. This theoretical value is calculated from the initial fluid volume. If sources or sinks are present the theoretical effect of these are added to the volume via the relation

$$V_0 = V_0 + S \cdot \Delta t \tag{3.34}$$

where S is the total effect of all sources and drains in the simulation domain. This is done after the projection step of the Navier-Stokes solver to prevent conflict with the compensation step described below.

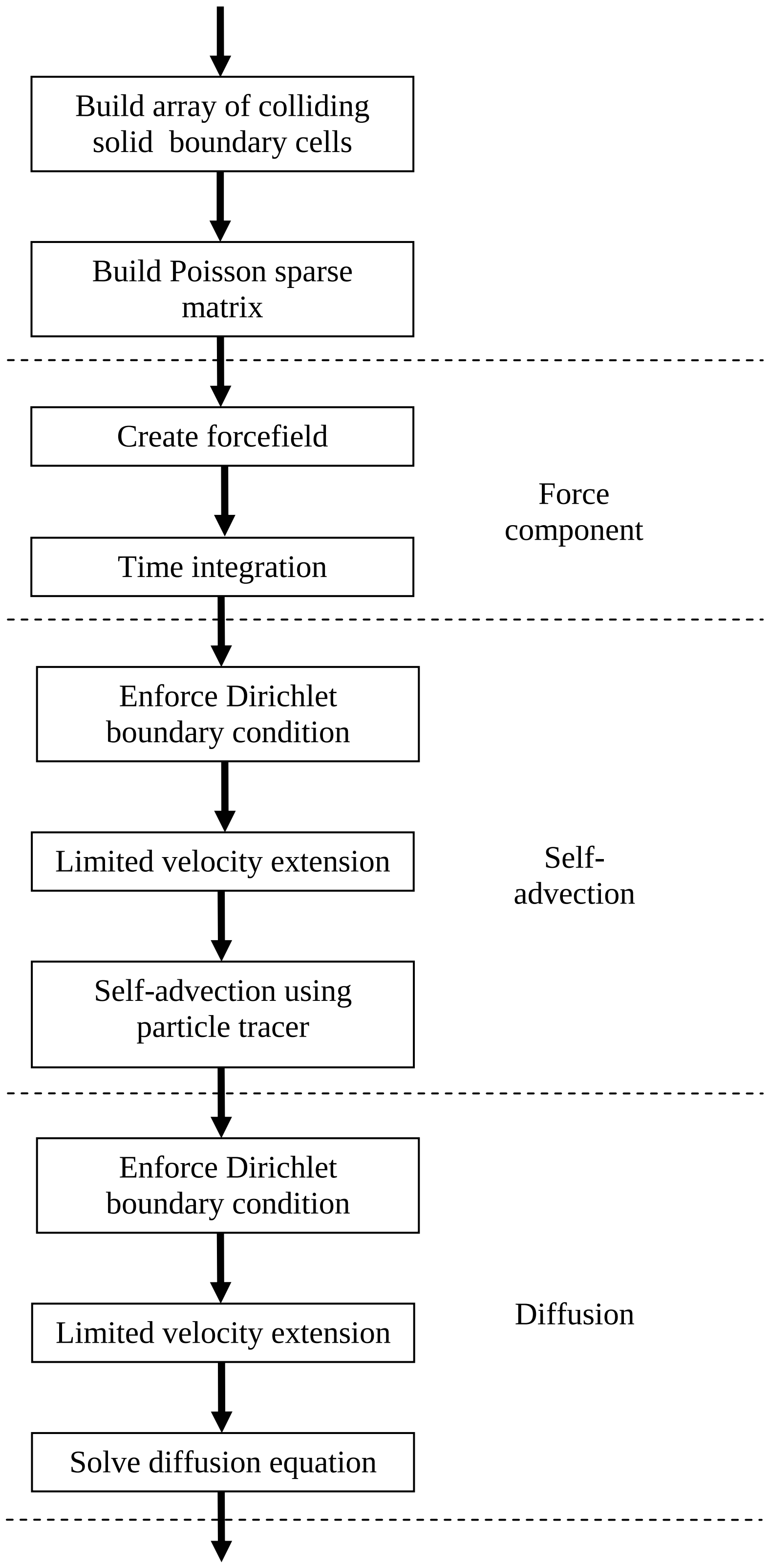
The first step of the forced volume conservation algorithm is to calculate the difference between the current fluid volume V_c and the theoretical value V_0. If the difference indicates that the current volume is equal to or greater then V_0 nothing is done, but if the difference indicates that volume has been lost we proceed to the next step. Assume that we have lost a volume of ΔV volume units (VU). To compensate for the lost volume we assign a small source in each grid cell containing fluid so that that the combined effect of all the sources will compensate for the lost volume in one time-step. The formulae will look like this:

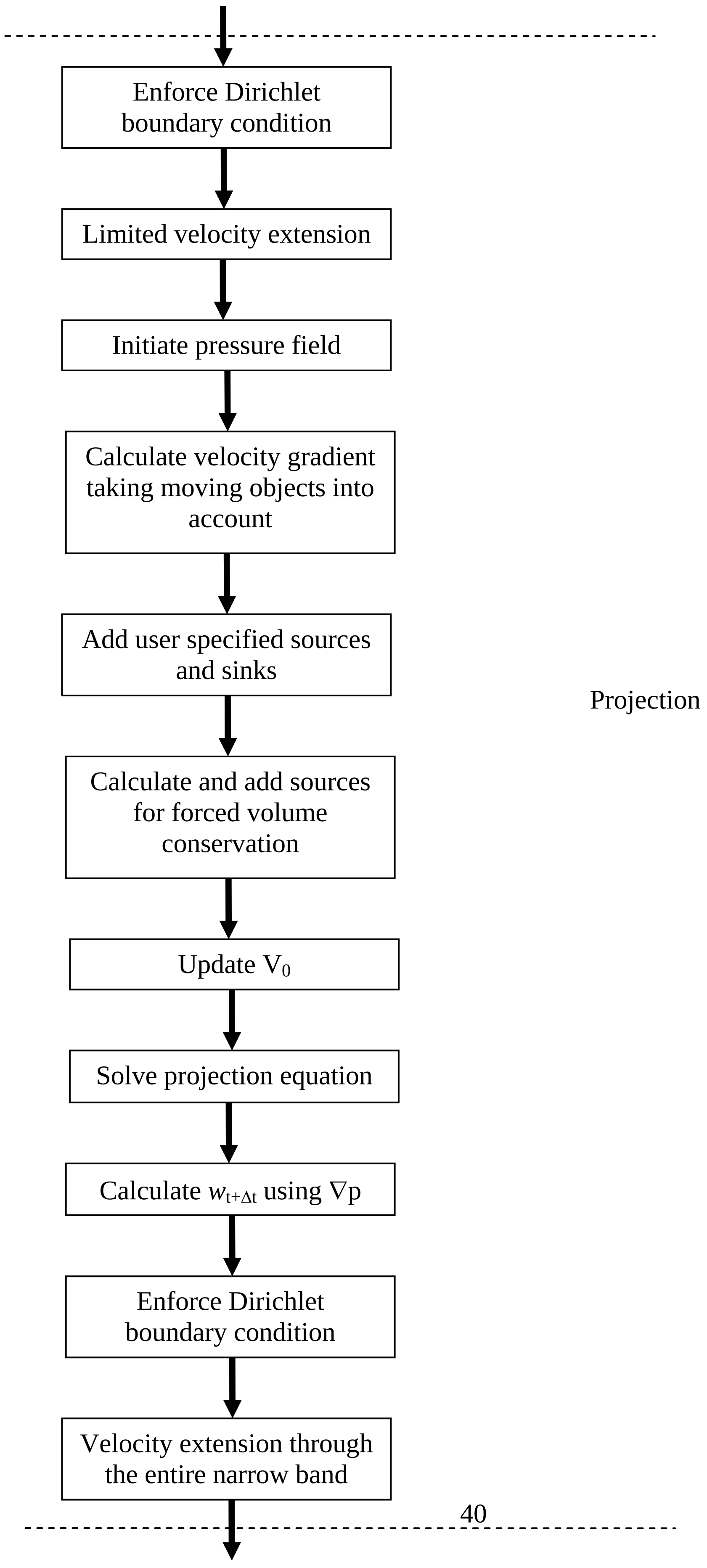
$$S_{cell} = \frac{V_0 - V_c}{n \cdot \Delta t} \tag{3.35}$$

Here n is the total number of grid cells containing fluid. Compensating for all lost volume in one time-step may sound aggressive but this will generally not be a problem since the volume loss for each time-step is very small. The idea is that as long as the volume lost during each time-step is small it will be impossible to see the effect of the tiny sources distributed throughout the fluid volume. The uniform distribution of sources is chosen since it means that the bulk of the added volume will be to regions that already contain large volume. These regions generally have a small surface area compared to the volume and thus the tiny disturbance of the motion of the surface caused by the added sources will be too small to see.

3.4.10 The Navier-Stokes update cycle

Let’s take a look at the flow diagram describing the overall update cycle for the Navier-Stokes solver:


Build array of colliding solid boundary cells
Build Poisson sparse matrix
Create forcefield
Time integration
Force component
Enforce Dirichlet boundary condition
Limited velocity extension
Self-advection using particle tracer
Self-advection
Enforce Dirichlet boundary condition
Limited velocity extension
Solve diffusion equation
Diffusion


Enforce Dirichlet boundary condition
Limited velocity extension
Initiate pressure field
Calculate velocity gradient taking moving objects into account
Add user specified sources and sinks
Calculate and add sources for forced volume conservation
Update V_0
Solve projection equation
Calculate $w_{t+\Delta t}$ using ∇p
Enforce Dirichlet boundary condition
Velocity extension through the entire narrow band
Projection

3.5 Input/Output

3.5.1 Standard output

The standard output format of the program is frame-files. These files store binary data containing the band structure of each level set in the simulation domain. Each frame is stored in a separate file. At default parameters the program outputs frames at 1/25 second intervals.

3.5.2 Vispack volumes

The simulation system is able to read Vispack volumes containing initial geometry for the solid boundaries and the fluid. More information about the Vispack file format can be found at the Vispack library webpage **http://www.cs.utah.edu/~whitaker/vispack/**

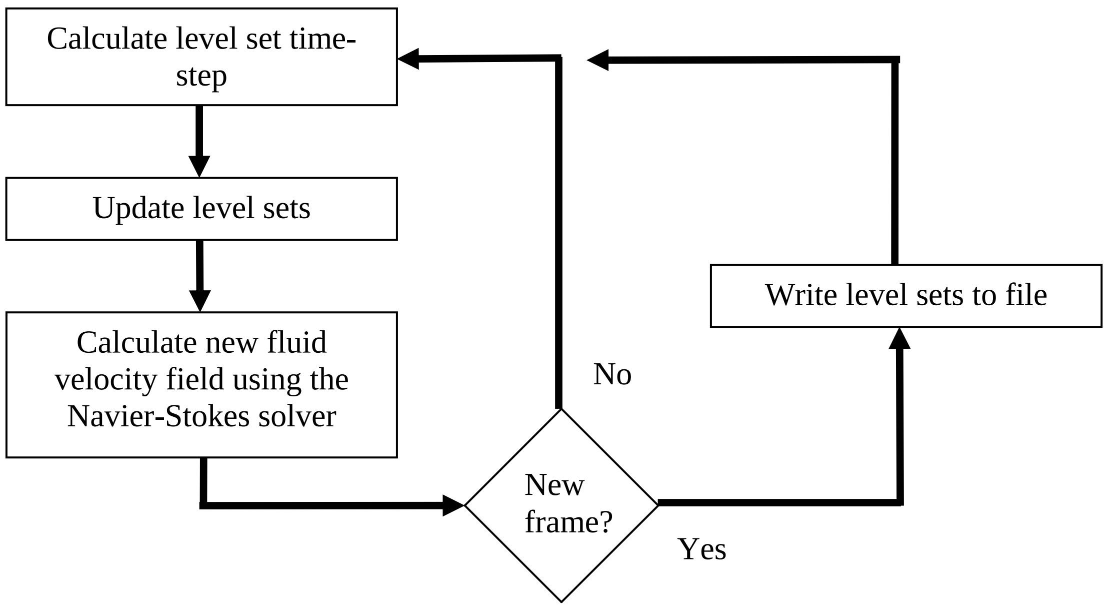
3.5.3 Scan conversion system

In order to allow for complex initial geometry a scan conversion system has been implemented using the closest point transform library by Sean Mauch found at **http://www.its.caltech.edu/~sean/**

Via a script it is possible for the user to take a Wavefront object file (.obj) and convert the polygonal model into a Vispack volume that can then be used as input to the simulation engine.

3.6 Simulation cycle overview

The overall simulation cycle of the program is presented below:


Calculate level set time-step
Update level sets
Calculate new fluid velocity field using the Navier-Stokes solver
New frame?
No
Yes
Write level sets to file

Since the Navier-Stokes solver is based on the stable fluids approach it is quite robust with regard to time-step size and thus we use the same time-step as for the level set when updating the velocity field through the Navier-Stokes solver.

3.7 Visualizing the results

In order to visualize the resulting simulation we use a fast implementation of Marching Cubes that takes advantage of the fact that we know the distance to the interface at each point in space and that we only need to march through the narrow band. This transforms the implicit surface representation of the level set into an explicit, in this case polygonal, surface. How well different features on the implicit surface will be represented depends on the nature of the algorithm used for this transform. One may use several kinds of interpolation schemes to achieve results pleasing to the eye. It is also possible to use ray-tracing to visualize the level set surface. Since this method allows for refraction and reflection effects to be taken into account it is much preferred when visualizing transparent fluids like water. Since a full fledged ray-tracer is quite a project in its own right we have chosen to just use the Marching Cubes algorithm for our program.

Using a playback function the it is possible to re-run a previous simulation frame-by-frame from the standard output files to see the finished simulation. It is also possible to output each frame to an image that can be used to make an animation of the completed simulation.

4 Results

4.1 Example 1

The example in figure 4.1 and 4.2 shows the improvement gained by the forced volume conservation algorithm. A sphere of water is being dropped from a height of approximately

1.4 meters into a rectangular bounding box. The dimensions of the simulation domain are 1.27x2.0x1.27 meters and consist of 70x110x70 voxels. The two simulations shown are run with identical initial conditions, but the simulation shown in figure 5.2 is using the forced volume conservation system while the simulation in figure 5.1 is not.

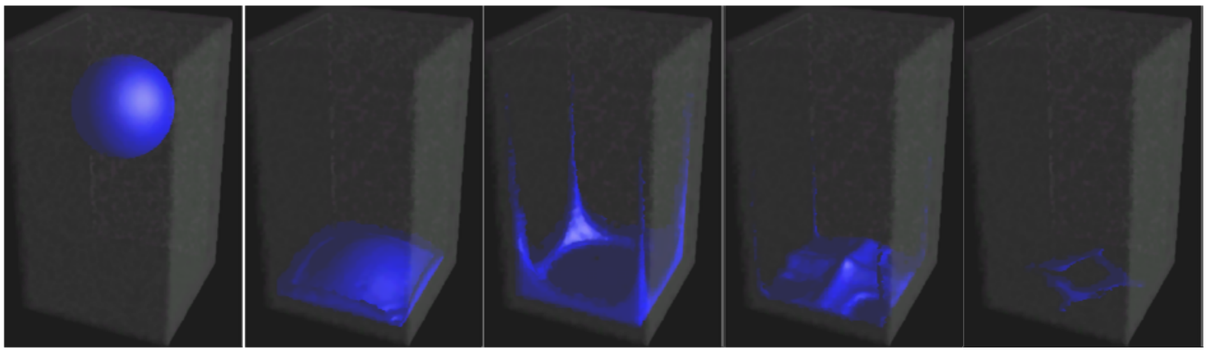


Figure 4.1 Falling water sphere simulation not using forced volume conservation. The pictures represent frame 1, 30, 50, 70 and 150 respectively

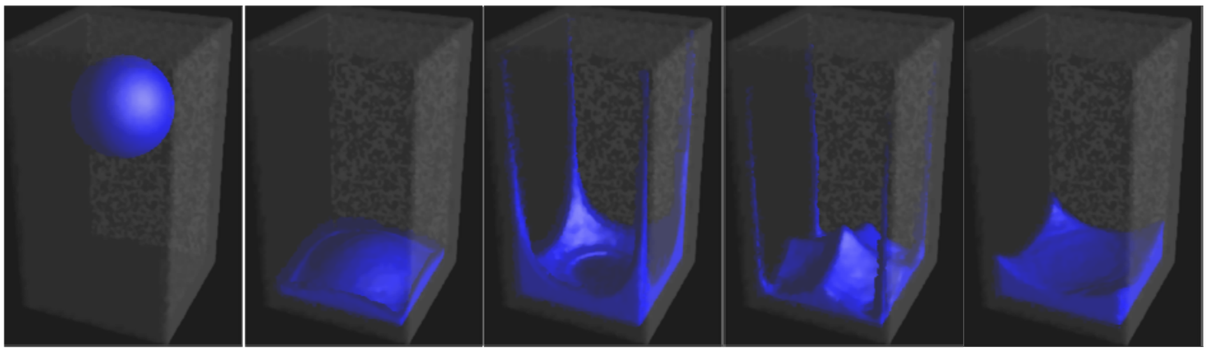


Figure 4.2 Falling water sphere using force volume conservation. The pictures represent frame 1, 30, 50, 70 and 150 respectively

4.2 Example 2

Example 2 shows a high resolution simulation that is designed to stress the components of the simulation engine and also show the ability of the particles to represent and maintain thin sheets of water on solid boundaries. A source injects water at high speed into the simulation domain and as the water hits the floor a large area is covered by a thin water surface. Turbulent flows develop creating a complex water surface. The particle system is able to well preserve the thin surface and the forced volume conservation algorithm compensates for the water that is still lost. The simulation domain is a 2x2x2 meter cube and the resolution is 130x130x130 voxels. The average simulation time per frame for this animation was 42 minutes.

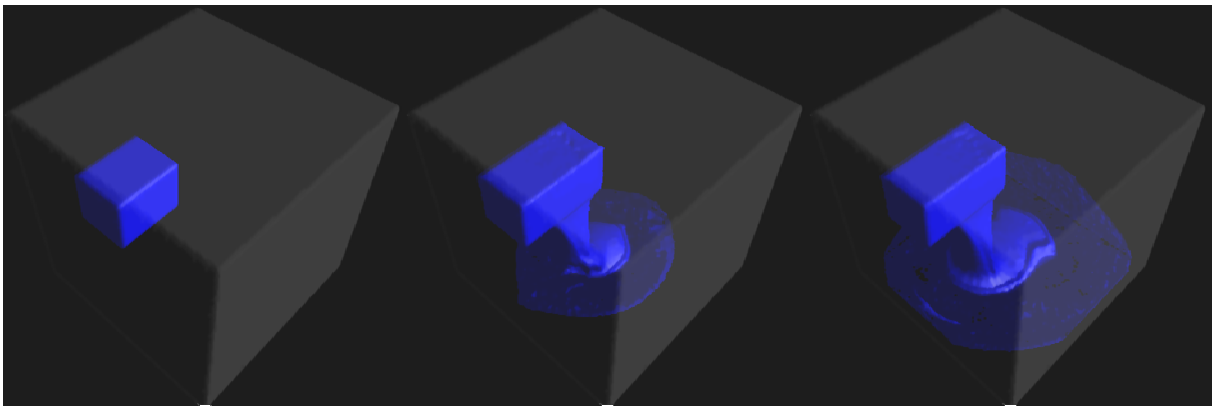


Figure 4.3 Water source animation frame 1, 12 and 14

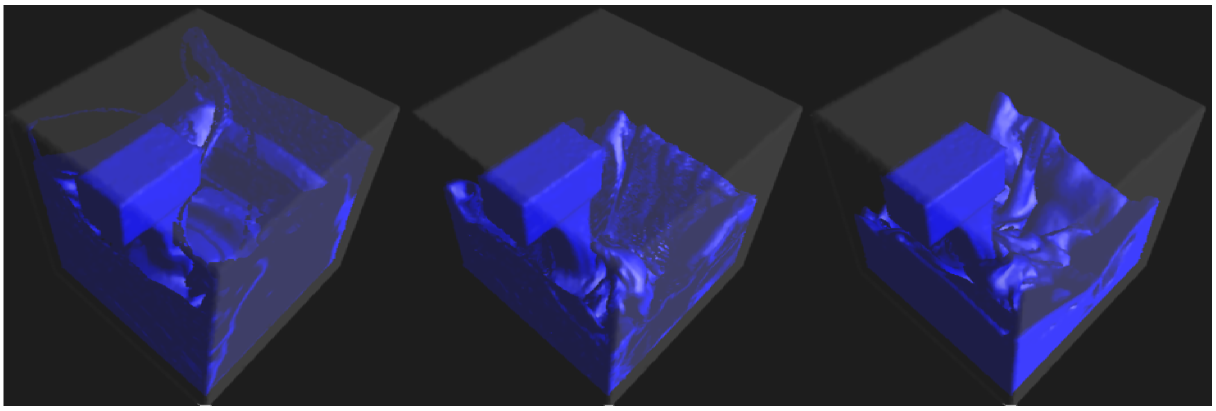


Figure 4.4 Water source animation frame 32, 46 and 65

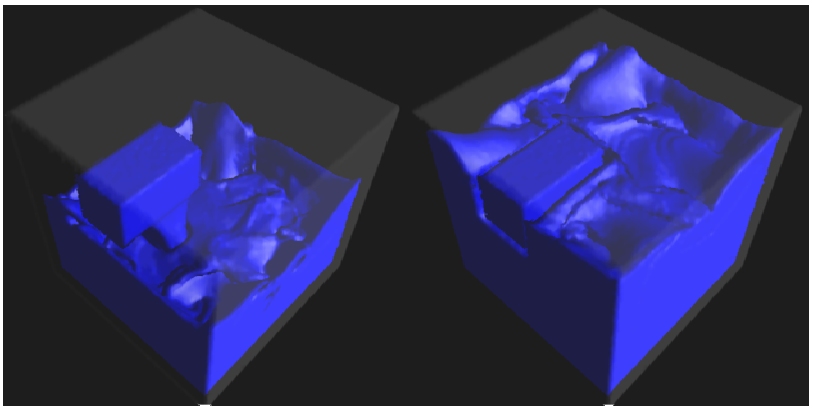


Figure 4.5 Water source animation frame 104 and 206

4.3 Example 3

This example shows the possibility of using polygonal models to create the initial conditions for the simulation engine. A model of an hourglass containing water is created and exported to two .obj files, one containing the hourglass and one containing the water. The files are run through the scan-conversion system and the resulting Vispack volumes are used as initial conditions to the simulation engine. The simulation domain is 0.98 x 2.0 x 0.98 meters and the resolution is 98 x 200 x 98 voxels.

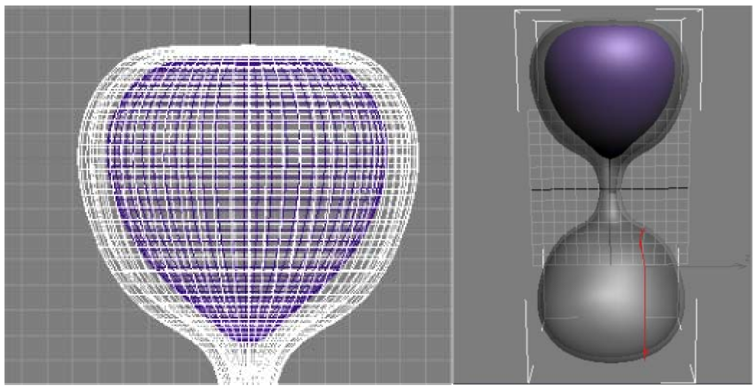


Figure 4.6 3D model of hourglass with initial water volume

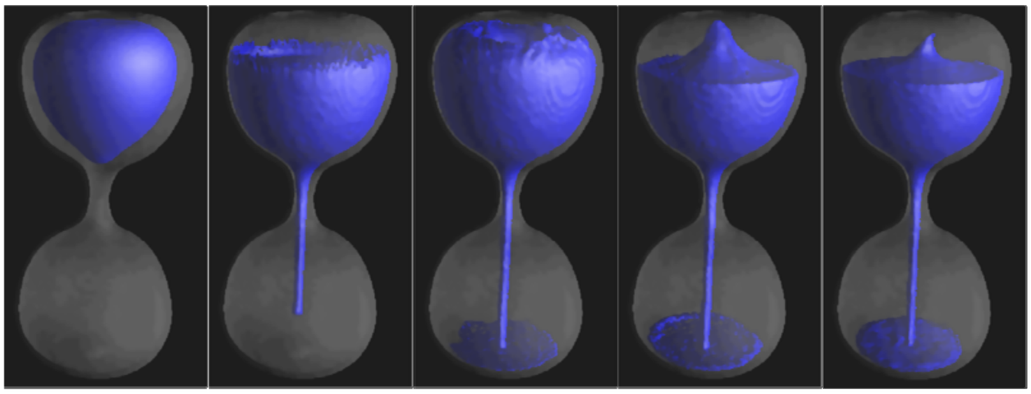


Figure 4.7 Hourglass simulation frame 1, 7, 13, 21 and 37

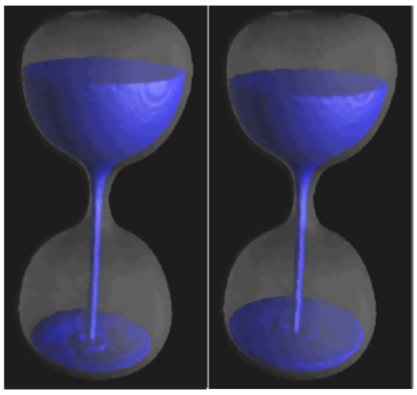


Figure 4.8 Hourglass simulation frame 60 and 180

5 Encountered problems and future work

Though the simulation system works well there were some problems encountered that were not solved. For the Navier-Stokes solver problems with keeping the fluid out of solid boundaries was encountered. As the simulation proceeded there was a tendency for the fluid to move further and further into the boundaries. This was not a problem for most simulations but did become a problem when violent flows or very narrow passages (only a few grid cells wide) were present. The cause for this was not discovered but is believed, at least partly, to be due to numerical errors when calculating the surface normal used in the Dirichlet boundary condition. A simple attempt at solving this problem was made by detecting fluid cells on their way into a boundary and adding a small velocity in the surface normal direction to these cells. Though this improved the situation it does not solve the problem.

There is room for improvements to the forced velocity conservation method. A way to correct for the velocity over time instead of correcting the entire error may be introduced as well as a non-uniform distribution for how fluid is added. The current volume conservation algorithm is also limited to adding fluid and not removing it if there happens to be too much in the simulation domain. This is to avoid removing fluid from thin interfaces thus increasing their rate of dissipation. The removal of fluid would be possible without degrading the visual appearance if the sinks were only added to regions where there already is a lot of fluid.

The visual appearance of the fluid, especially the lack of droplets may be modeled using particles. Some work has been done to achieve this but the combination of the Lagrangian methods that may be used on the particles and the geometry provided by the distance information of the level set provide a good platform for complex particle systems that live on and around the interface.

Finally the system for sources can be improved. Instead of adding actual sources to the projection operator one may model the sources by explicitly creating a difference in velocities that is added to the velocity field created by the Navier-Stokes solver.

6 Bibliography

[1] Enright, D., Marschner, S. and Fedkiw, R., *Animation and Rendering of Complex Water Surfaces*, SIGGRAPH 2002, ACM TOG 21, 736-744 (2002).

[2] Enright, D., *Use of The Particle Level Set Method For Enhanced Resolution of Free Surface Flows*, PhD Thesis, Stanford, (2002).

[3] Foster, N. and Fedkiw, R., *Practical Animation of Liquids*, SIGGRAPH 2001, 23-30 (2001).

[4] Foster, N. and Metaxas, D., *Realistic Animation of Liquids*, Graphical Models and Image Processing, 58, 471-483 (1996).

[5] Hong, J.-M. and Kim, C.-H., *Animation of Bubbles in Liquid*, EUROGRAPHS 2003, (2003).

[6] Mauch, S., *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*, PhD thesis, Caltech, (2003.)

[7] Museth, K., Breen, D., Whitaker, R., Mauch, S. and Johnsson, D., *Algorithms for Interactive Editing of Level Set Models*, submitted to ACM Journal of Eurographic: Graphics Foum, pp. 14

[8] Nordling, C., Österman, J., *Physics Handbook for Science and Engineering*, Studentlitteratur, (1999)

[9] Osher, S. and Fedkiw, R., *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag, (2003)

[10] Peng, D., Merriman, B., Zhao, H., Osher, S. and Kang, M., *A PDE Based Fast Local Level Set Method*, Journal of Computational Physics,Vol. 155, pp. 410-438, 1999.

[11] Stam, J., *Real-Time Fluid Dynamics for Games*, Technical report, (2001).

[12] Stam, J., *Stable Fluids*, SIGGRAPH 2001, (2001).

[13] Strain, J., *Semi-Lagrangian Methods for Level Set Equations*, Journal of Computational Physics (1998).

[14] Weisstein, E., *Mathworld*, http://mathworld.wolfram.com/

[15] Witacker, R., *VISPACK*, http://www.cs.utah.edu/~whitaker/vispack/

[16] Wrenninge, M., *Fluid Simulation for Visual Effects*, Master Thesis, Linkoping University, (2003).