Master's Thesis

Fluid Simulation for Computer Graphics

Student: Charis Kontaxis
Student Number: 3721728
Thesis Number: ICA-3721728

Supervisor: dr. R. T. Tan

February 2013

Department of Information and Computing Sciences
Utrecht University

Abstract

Fluid simulation for computer graphics is a special part of Computational Fluid Dynamics (CFD) which is used in graphics applications to generate realistic representations of different types of fluids such as water, smoke etc. The main difference between applications in computer graphics and more general CFD applications is that the results do not need to be physically correct as long as they seem convincing and visually appealing to the human observer. Computer graphics applications range from real-time simulations that are integrated in computer games and completely off-line ultra realistic simulations that are mostly used in special effects in the movie industry.

In this Master's Thesis we will focus on the Computer Graphics applications of fluid simulation used in the movie industry. We develop a three dimensional fluid simulator, which uses a staggered grid and follows the Eulerian viewpoint together with a semi-Lagrangian advection technique. MICCG(0) (modified incomplete Cholesky conjugate gradient, level zero) is used for the pressure projection. Different methods to track the fluid are implemented, including marker particles, level set method and particle level set method. The interaction between solids and fluid is possible under a fluid-solid coupling technique. The scenes are rendered using pbrt and Robert Bridson's grid-based implicit surface shape plugin for pbrt.

Each frame of a simulation that follows the Eulerian viewpoint usually takes several seconds to be computed. As result of this, we focused our efforts into researching what parts delay the simulation the most. Once these parts are identified we propose some methods to deal with these bottlenecks and speed up the simulation. For this purpose, we propose a new hybrid-grid structure, instead of the standard regular grid usually used in this type of simulations, in order to decrease the number of computations needed. Furthermore, we add multi-threading at some time consuming parts of the simulation to further optimize performance. Finally, we combine the two above methods to acquire the highest performance increase featured among our contributions.

Dedicated to you, to the Purpose and to the future.

Acknowledgements

I would like to thank my supervisor Robby for giving me the chance to work on a project like this, despite the fact that it did not exactly match his field of expertise. He was always available to reply to questions or provide some feedback and more importantly, during this project, he constantly tried to make me work harder and more efficiently.

I would also like to thank my family and my girlfriend for always being there for me. Their support on so many levels, helped me tremendously to focus on my studies and finally receive this Master's degree.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Fluid simulation has a long history in computer graphics and has attracted hundreds of researchers in the past three decades. The modeling of natural phenomena such as water remains a challenging problem in computer graphics. This is not surprising since the motion of fluids is highly complex. Visual fluid models have many obvious applications in the industry including special effects and interactive games. Ideally, a good computer graphics fluid model should both be easy to use and produce highly realistic results.

The field of computational fluid dynamics is devoted to the simulation of gases and other fluids such as water. Only recently have researchers in computer graphics started to explore the abundant CFD literature for algorithms that can be adopted and modified for computer graphics applications. Due to the computational expense of capturing the complex motion of fluids, fluid simulations are typically executed off-line.

There are two basic approaches to solving the fluid equations: the grid-based (Eulerian) and the particle-based (Lagrangian) approach. Both have been successfully used as off-line methods to create impressive effects in feature films and commercials. Faster methods can be made by reducing the grid resolution or the number of particles from the millions to the thousands. This reduction comes at a price of quality. Interesting features of a full 3D simulation such as splashes and overturning waves get lost because the height field representation cannot capture them. Producing faster simulations and improving the quality or maintaining it at similar levels has been the main focus of researchers in this field.

Chapter 2

Related work

In this chapter we present some of the most important work that has been done in the Fluid Simulation field. The reader can find both fundamental and more recent methods on this field and further investigate any of the subtopics that are presented here.

Early work in the field of Eulerian fluid simulation in computer graphics include [21] who used finite differences to solve the Navier-Stokes equations, [47] who introduced the semi-Lagrangian method for advection and [20] who combined Lagrangian particles with the level set method to track the free surface of liquids.

The simulation of a fluid can be split into four main parts: advection, surface tracking, pressure projection and fluid-solid interaction. Each of these parts has been the focus of many research papers.

2.1 Advection

[12] proposed a simple method of characteristics scheme for discretizing advection equations. These semi-Lagrangian type schemes are popular in many areas (for example in the atmospheric sciences community [48]), because they can be made unconditionally stable. The simplest semi-Lagrangian scheme is first order accurate in space and time. Generally, order of accuracy tells us how much improvement to expect if we are close to a solution and want to make it better. The order of accuracy of a semi-Lagrangian scheme can be improved by using higher order interpolation (e.g. [34]). However, this can significantly increase the complexity and computational cost of the method especially since high order polynomial interpolants require limiters to avoid oscillations and possible instability of the simulation overall. Another way to improve the fidelity of semi-Lagrangian schemes is via auxiliary information. [18] showed that vorticity confinement [48] , which is used to model the small scale rolling features characteristic of smoke, could be used to alleviate the high amount of dissipation allowing for visually intricate, albeit non-physical flows (see also [44] which used vortex particles). Similarly, [17] showed that the simple first order accurate semi-Lagrangian scheme can be used to obtain very accurate level set tracking as long as particles are tracked with higher order accuracy. [45] used a modified MacCormack scheme to lift the semi-Lagrangian step to second order accuracy.

2.2 Surface Tracking

In the level set method [38] the implicit material boundary/interface is given by the zero set of a scalar field. In order to reduce volume-loss, [15] added particles on both sides of the liquid surface to correct the level set. Apart from level sets, other representations of the liquid surface have been proposed such as the volume of fluid method and density based approaches. Explicit triangle meshes [3], [37], [7], [57] have also been researched.

2.3 Pressure Projection

[16] used the ghost fluid method to improve the accuracy of pressure projection near the free surface. In many cases, the pressure projection step is the slowest part in liquid simulations because it involves solving a large linear system at each time step. The preconditioned conjugate gradients (PCG) method is commonly used [20], [6] for solving this system efficiently. The regularity of Eulerian grids makes the multigrid approach an effective alternative to PCG[35]. [33] combined both approaches and used one multigrid V-Cycle as a conjugate gradients's pre-conditioner. To accelerate the multigrid approach [31] modified the restriction and prolongation stencils to only consider velocities on the faces of coarser cells.

2.4 Fluid-Solid Interaction

The complex and interesting motion of a fluid is typically caused by its interaction with the solid environment. Therefore, handling solid boundary conditions correctly has been a further active research area. [50] proposed to use a volume of fluid fraction method to handle two-way rigid body coupling accurately. [8] included fluid cells as well as cells occupied by rigid bodies in one pressure solve. Similarly, [29] combined fluid motion and rigid body momentum into a single linear system and solved it simultaneously. Later, the method was extended to include soft body-fluid coupling [9] and then re-formulated to conserve momentum yielding a symmetric system matrix in [42]. Two- way coupling of fluids with cloth and thin shells was studied by [24]. Using a variational formulation [4] were able to handle fluid-solid interactions with sub- grid accuracy.

2.5 Other Approaches

Other approaches to simulate liquids include particles-based methods such as [36], [40], [2], [46] and lattice-Boltzmann models [51], [53]. Real-time performance has been achieved by using the pipe model [56], the 2D wave equation [27] and the shallow water equations [52], [10] to name a few. Height field methods cannot capture the 3D phenomena faithfully though. So far, only a few researchers have shown 3D Eulerian liquid simulation at interactive rates. To achieve real-time performance [13] confined the liquid to a relatively small rectangular domain with- out general fluid-solid interaction, while [30] leveraged the discrete cosine transform to speed up their simulation.

Chapter 3

Fluid Simulation

In this chapter the mathematical foundations behind fluid simulation will be presented. The equations that govern fluid in real world and how these can be numerically approximated in a computer. Furthermore, the parts of the fluid simulation loop and the methods used to implement each one of these in my Master's Thesis will be analyzed.

3.1 Simulation Viewpoints

In physics, a fluid is a substance that continually deforms (flows) under an applied shear stress. Fluids are a subset of the phases of matter and include liquids, gases, plasmas and, to some extent, plastic solids. There are two approaches to track the motion of a moving fluid or deformable solid (continuum): The Lagrangian viewpoint and the Eulerian viewpoint.

1. Lagrangian viewpoint
   Conceptually, under this viewpoint, fluid consists of infinitely many points in space. Each of these points in the fluid or solid is labeled as a separate particle, with a position $\vec{x}$ and a velocity $\vec{u}$. Numerically, it corresponds to a particle system with or without a mesh connecting up the particles.

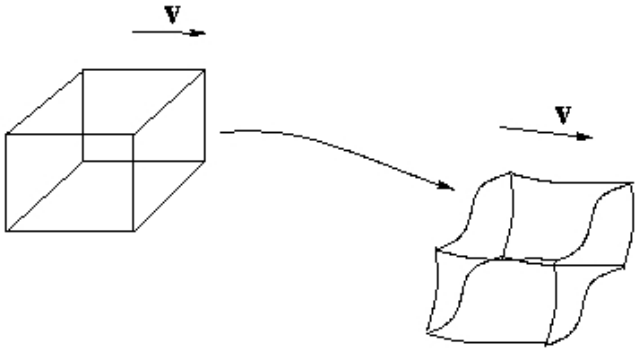

Figure 1: Lagrangian viewpoint: we observe each fluid element as it follows the fluid flow and changes its properties.

2. Eulerian viewpoint
   We maintain fixed points in space and see how measurements of fluid quantities, such as velocity, temperature, etc., at those points change in time. Numerically, corresponds to using a fixed grid that doesn’t change in space even as the fluid flows through it. In this viewpoint, it is easier to analytically work with spatial derivatives of some fluid's quantities and also numerically approximate these derivatives on a fixed Eulerian mesh.

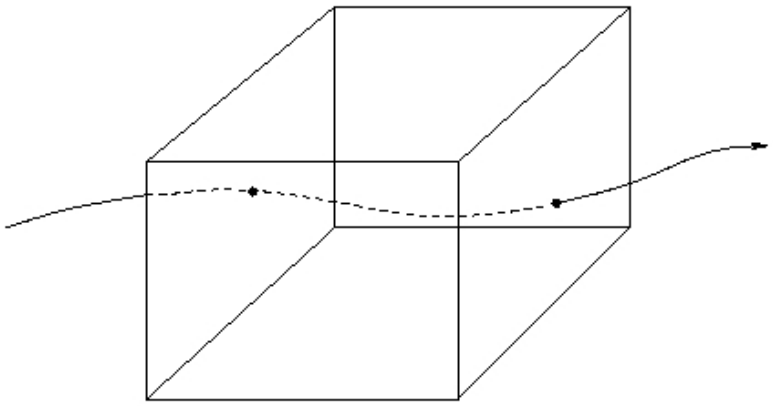

Figure 2: Eulerian viewpoint: stay at a fixed point in space and watch fluid move through your volume element: properties of fluid in volume continually changing

3.2 Fluid Equations

Most fluid flow is governed by the Incompressible Navier-Stokes equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + v \nabla \cdot \nabla \vec{u} \quad (1)$$

$$\nabla \cdot \vec{u} = 0 \quad (2)$$

3.2.1 Symbols

The letter $\vec{u}$ is traditionally used in fluid mechanics for the velocity of the fluid. The Greek letter ρ stands for the density of the fluid. The letter p stands for pressure, the force per unit area that the fluid exerts on anything. The letter $\vec{g}$ is the familiar acceleration due to gravity, $(0, -9.81, 0)\, m/s^2$. The Greek letter v is technically called the kinematic viscosity and measures how viscous the fluid is.

3.2.2 Momentum equation

Breaking Equation(1) down to three parts:

I. Material derivative:

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \vec{u} \cdot \nabla q \quad (3)$$

q here represents one of the simulation quantities that will be transported through the flow of the fluid. (e.g. temperature) and t represents time.

$\frac{\partial q}{\partial t}$ term shows how fast q is changing at that fixed point in space, an Eulerian measurement. The second term, $\vec{u}\cdot\nabla q$, is correcting for how much of that change is due just to differences in the fluid flowing past.
For example, imagine that in our simulation space the temperature is zero at the origin and gets warmer as we look further to the right: $T(x) = 20\mathrm{x}$ and there is a stream of water flowing at a constant velocity: $\vec{u} = c$
If the temperature of all molecules of the water does not change then the material derivate of temperature under the Lagrangian viewpoint is:

$$\begin{aligned} \frac{DT}{Dt} &= 0 \Rightarrow \\ \frac{\partial T}{\partial t} + \nabla T\cdot\vec{u} &= 0 \Rightarrow \\ \frac{\partial T}{\partial t} + 20\cdot c &= 0 \Rightarrow \\ \frac{\partial T}{\partial t} &= -20\mathrm{c} \end{aligned} \tag{4}$$

This means that at a fixed point in space, the temperature is changing at a constant rate of $-20\mathrm{c}$. If water is still ($c = 0$) then the temperature at a fixed point will not change. If water flows to the right at a constant velocity $c = 2$ then the temperature at a fixed point will drop at a rate of -40.

Subsequently, we see that while under the Lagrangian viewpoint the temperature of each individual molecule of water does not change and thus the Lagrangian derivative is zero, the Eulerian derivative will be anything depending on how fast and which direction the flow is moving.

When the quantity q we check is velocity then this is called self-advection as velocity appears in two different roles, namely: the velocity field in which the fluid is moving and as the fluid quantity that is getting advected. This leads us to the same equation that is one part of the momentum equation:

$$\frac{D\vec{u}}{Dt} = \frac{\partial\vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} \tag{5}$$

By advection in this context we mean a transport mechanism of a substance by a fluid, due to the fluid's motion in a particular direction. This term is used in the fluid simulation field to describe that the material derivative of a quantity q is equal to zero.

II. External forces that act on the fluid: $\vec{g}$ shows the effect that gravity has on the fluid.

III. Internal forces that act on the fluid are the two fluid forces which affect how fluid particles

interact with other particles nearby:

i. Term $\frac{1}{\rho}\nabla p$ shows the effect of pressure on the fluid. Pressure can be seen as being whatever it takes to keep the fluid at constant volume and the velocity divergence-free. High pressure fluid regions push on lower pressure regions. We can measure this imbalance in pressure at the position of the particle by simply taking the negative gradient of pressure: $-\nabla p$. Because of the fact that the gradient operator represents the direction of the steepest ascent, which in this case would be meaning that low pressure regions push towards higher pressure regions, we need to take the negative gradient of the pressure at each point.

ii. The second fluid force is due to viscosity. Viscous fluid tries to resist deforming. It is a force that tries to make our particle move at the average velocity of the nearby particles, meaning that it tries to minimize differences in velocity between nearby bits of fluid. The term $\nabla\cdot\nabla\vec{u}$ shows that as the Laplacian ($\nabla\cdot\nabla$) differential operator can be used to measure how far a quantity is from the average around it, and in this case it allows us to measure differences in velocity.

3.2.3 Incompressibility condition

Real fluids, even liquids like water, change their volume but usually fluids do not change their volume very much. The study of how fluids behave in these situations is generally called compressible flow. It is complicated and expensive to simulate tiny perturbations in the volume and as they have so small of an effect on how fluids move at a macroscopic level, they are practically irrelevant for animation.

We can measure how fast the volume of this chunk of fluid is changing by integrating the normal component of its velocity around the boundary. For an incompressible fluid, the volume should stay constant. So this rate of change should be zero:

$$\begin{aligned} &\frac{d}{dt}volume(\Omega) = \iint_{\partial\Omega}\vec{u}\cdot\hat{n} = 0 \Rightarrow^{*} \\ &\frac{d}{dt}volume(\Omega) = \iiint_{\Omega}\nabla\cdot\vec{u} = 0 \\ &^{*}\textit{by applying the divergence theorem} \end{aligned} \tag{6}$$

Thus we are lead to Equation(2): $\nabla\cdot\vec{u} = 0$, which is the condition that we have to satisfy in order to ensure that the fluid is incompressible.

3.3 Viscosity

In most cases viscosity plays a minor role in the animation and so we can drop it. Most numerical methods for simulating fluids unavoidably introduce errors that can be physically reinterpreted as viscosity. The Navier-Stokes equations without viscosity are called the Euler

equations and such an ideal fluid with no viscosity is called inviscid. Here are the final equations for inviscid fluid:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} \tag{7}$$

$$\nabla \cdot \vec{u} = 0 \tag{8}$$

There are two boundary conditions, solid walls and free surfaces.

i. A solid wall boundary is where the fluid is in contact with a solid. The fluid should not be flowing into the solid or out of it, thus the normal component of velocity has to be zero: $\vec{u} \cdot \hat{n} = 0$ or in case that solid is moving $\vec{u} \cdot \hat{n} = \vec{u}_{solid} \cdot \hat{n}$

ii. A free surface is where we stop modeling the fluid. Outside the area of our simulation probably exists another fluid (e.g. air). Since only differences in pressure matter (in incompressible flow), a free surface is one where $p = 0$.

3.4 Numerical Simulation

3.4.1 Fluid Algorithm

We will we split up the more complicated Euler equations[(7) and(8)] into their component parts and solve each one separately in turn on the incompressible fluid equations:

$$\frac{Dq}{Dt} = 0 \tag{9}$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{g} \quad \text{(body forces)} \tag{10}$$

$$\frac{\partial \vec{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0, \quad \text{such that } \nabla \cdot \vec{u} = 0 \quad \text{(pressure/incompressibility)} \tag{11}$$

Basic fluid algorithm:

1. Start with an initial divergence-free velocity field $\vec{u}^0$
2. For time step $n = 0,\ 1,\ 2,\ \ldots$
3. Determine a good time step Δt to go from time t_n to time t_{n+1}
4. Set . $\vec{u}^A = advect(\vec{u}^n, \Delta t, \vec{u}^n)$
5. Add $\vec{u}^B = \vec{u}^A + \vec{g} * \Delta t$
6. Set $\vec{u}^{n+1} = project(\Delta t, \vec{u}^B)$

3.4.2 MAC Grid

In order to discretize the space, a MAC grid is used (introduced in the marker-and-cell method [25]) which is a staggered grid meaning that different variables are stored at different locations. The pressure in grid cell (i, j) is sampled at the center of the cell, indicated by $p_{i,j}$. The velocity is split into its two Cartesian components. The horizontal u -component is sampled at the centers of the vertical cell faces, for example indicated by $u_{i+1/2,j}$ for the horizontal velocity between cells (i, j) and $(i+1, j)$. The vertical v -component is sampled at the centers of the horizontal cell faces, for example indicated by $v_{i,j+1/2}$ for the vertical velocity between cells (i, j) and $(i, j+1)$. In three dimensions the MAC grid is set up the same way.

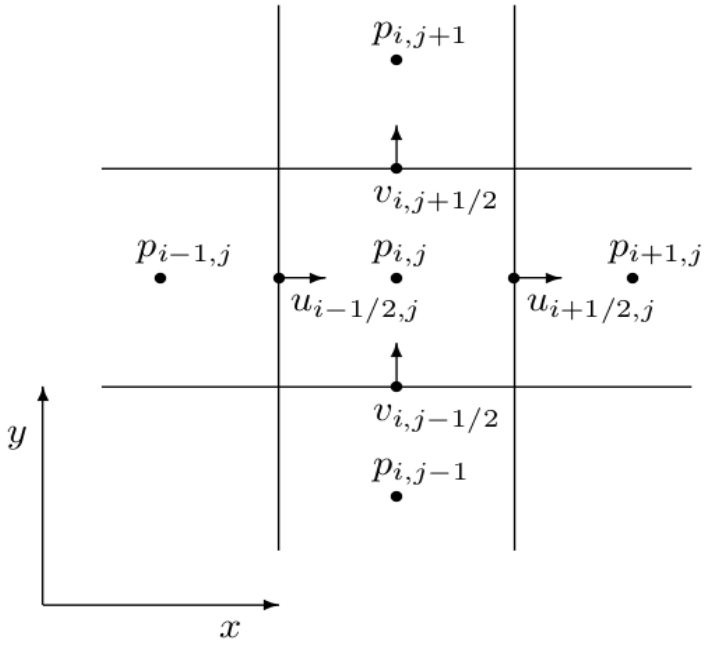

Figure 3: One cell from the two-dimensional MAC grid

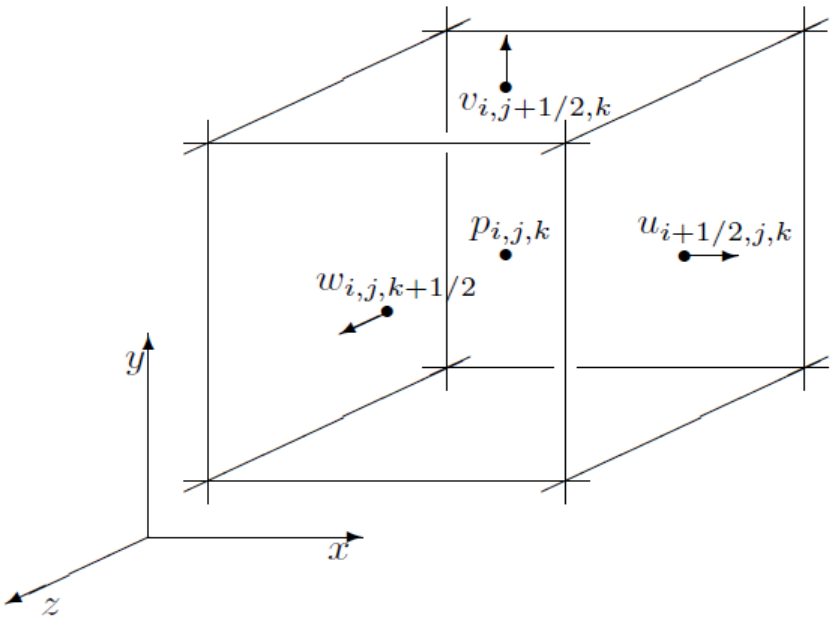

Figure 4: One cell from the three-dimensional MAC grid

The MAC grid is used to have accurate central differences to calculate derivatives. The derivative at a MAC grid point i is:

$$\left(\frac{\partial q}{dx}\right)_i \approx \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x}$$

This is unbiased and accurate to $O(\Delta x^2)$. Moreover, it does not skip over any values of q , so if we set this equal to zero we can only have q constant, resulting in a correct null-space. This means that the set of functions where the formula evaluates to zero does not contain more than just the constant functions it should be restricted to.

3.5 Advection

3.5.1 Semi-Lagrangian Advection

In order to get the new value of q at some point $\vec{x}$ in space, we could conceptually find the particle that ends up at $\vec{x}$ and look up its value of q . We can find this value as we know where the particle ends up and that it is moving through the velocity field $\vec{u}$. We will say that the location in space of the grid point we are looking at is $\vec{x}_G$. We want to find the new value of q at that point, which we will call q_G^{n+1} . We know that if a hypothetical particle with old value q_P^n ends up at $\vec{x}_G$, when it moves through the velocity field for the time step Δt , then $q_G^{n+1} = q_P^n$. By using one step of forward Euler (in our implementation we use the more accurate higher-order Runge-Kutta methods): $\vec{x}_P = \vec{x}_G - \Delta t \vec{u}(\vec{x}_G)$

3.5.2 Interpolation

So now we know where this particle comes from but most likely $\vec{x}_P$ is not on the grid, so we do not have the exact value, but we can get a good approximation by interpolating from q^n at nearby grid points.

$$q_G^{n+1} = interpolate(q^n, \vec{x}_p)$$

A critical factor affecting the stability of the numerical methods is Δt . Semi-Lagrangian advection is unconditionally stable in respect to Δt . Wherever the particle starting point ends up, we interpolate from old values of q to get the new values for q , so q stays bounded. If we only care about real-time rates of the simulation, we can pick Δt equal to the frame duration. Most of the times we limit Δt so that the furthest a particle trajectory is traced is three grid cell widths: $\Delta t \leq \frac{3\Delta x}{u_{max}}$, where $\vec{u}_{max}$ is an estimate of the maximum velocity in the fluid.

In the interpolation step of semi-Lagrangian advection we are taking a weighted average of values from the previous time step. With each advection step, we are doing an averaging

operation. Averaging tends to smooth out or blur sharp features, as it introduces numerical dissipation. When we use the simple semi-Lagrangian method to try to solve the advection equation without viscosity, our results look like we are simulating a fluid with viscosity. We could use the Catmull-Rom interpolation in the semi-Lagrangian method for advection which boosts the accuracy to second order (from first order from linear interpolation) and significantly reduces the numerical dissipation. However, it does not have quite the same guarantees for stability as linear interpolation[18].

3.5.3 MacCormack Method

The semi-Lagrangian advection scheme used by Stam[47] is useful for animation because it is unconditionally stable, meaning that large time steps will not cause the simulation to "blow up". However, it can introduce unwanted numerical smoothing, making water look viscous or causing smoke to lose detail. To achieve higher-order accuracy, we use a MacCormack scheme, presented in [45], that performs two intermediate semi-Lagrangian advection steps. Given a quantity φ , a semi-Lagrangian advection scheme A and A^R which indicates that advection is reversed-time is run backward for that step- then higher-order accuracy is obtained using the following sequence of operations:

1. $\hat{\varphi}^{N+1} = A(\varphi^N)$
2. $\hat{\varphi}^{N} = A^R(\varphi^{N+1})$
3. $\varphi^{N+1} = \hat{\varphi}^{N+1} + \frac{1}{2}(\varphi^N - \hat{\varphi}^N)$

where $\hat{\varphi}^N$ and $\hat{\varphi}^{N+1}$ are intermediate quantities, and φ^{N+1} is the final advected quantity.

Unlike the standard semi-Lagrangian scheme, this MacCormack scheme is not unconditionally stable. Therefore, a limiter is applied to the resulting value φ^{N+1} , ensuring that it falls within the range of values contributing to the initial semi-Lagrangian advection. Regarding the implementation, this means that we must locate the nodes closest to the sample point, and clamp the final value to fall within the minimum and maximum values found on these nodes.

In our implementation we started the implementation of the MacCormack method, but unfortunately due to the limited amount of time we had to finish the project and different focus that we decided to have during this project-which was to improve the speed of the simulation- we did not finish it.

3.6 Pressure Solve

3.6.1 Projection and Linear System Solve

The project routine will execute the following steps:

1. subtract off the pressure gradient from the intermediate velocity field $\vec{u}$:

$$u^{n+1} = \vec{u} - \Delta t \frac{1}{\rho} \nabla p \tag{12}$$

2. ensure that the result:
   - satisfies incompressibility inside the fluid:

$$\nabla \cdot \vec{u}^{n+1} = 0 \tag{13}$$

   - satisfies solid wall boundary conditions:

$$\vec{u}^{n+1} \cdot \hat{n} = \vec{u}_{solid} \cdot \hat{n}. \tag{14}$$

Equation(12), on boundary faces of the fluid region, involves pressures in grid cells that lie outside of the fluid region. If the boundary is a free surface then we assume that pressure outside the fluid is zero, so we set these $p_{i,j,k}$ equal to zero (Dirichlet boundary condition). If the boundary is solid wall we set the fluid velocity equal to solid velocity for these cells, so the $\vec{u} \cdot \hat{n} = \vec{u}_{solid} \cdot \hat{n}$ condition is met.

The divergence in three dimensions is: $\nabla \cdot \vec{u} = \frac{\partial u}{dx} + \frac{\partial v}{dy} + \frac{\partial w}{dz}$. By using central differences of the MAC grid cells we can calculate Equation(13). Combining Equations (12) and (13) we get the following Poisson problem:

$$-\frac{\Delta t}{\rho} \nabla \cdot \nabla p = -\nabla \cdot \vec{u} \tag{15}$$

If we use the MAC grid to numerically approximate the above equation in two dimensions then we will get:

$$\frac{\Delta t}{\rho}\left(\frac{4\,p_{i,j,k} - p_{i+1,j,k} - p_{i-1,j,k} - p_{i,j+1,k} - p_{i,j-1,k} - p_{i,j,k+1} - p_{i,j,k-1}}{\Delta x^2}\right) = -\left(\frac{u_{i+1,j,k} - u_{i-1,j,k}}{\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{\Delta x}\right) \tag{16}$$

In three dimensions this equation will follow the same pattern.

By observing Equation(16) we find the following patterns:

- for the free surface cell boundary condition, we just delete the corresponding p from the equation
- for the solid cell boundary condition, we delete the corresponding p and reduce the coefficient in front of $p_{i,j,k}$, which is at most four in two dimensions and six in three dimensions. The coefficient in front of $p_{i,j,k}$ is equal to the number of non-solid grid cell neighbors
- we increment the negative divergence measured on the right-hand side with a term involving the difference between fluid and solid velocity

Using these patterns we create a large system of linear equations for the unknown pressure values:

$$Ap = b \tag{17}$$

where A is the pressure coefficient matrix, p consists of the pressure unknowns, and b a vector consisting of the negative divergences in each fluid grid cell.

In order to solve the above system we use the MICCG(0) (modified incomplete Cholesky conjugate gradient, level zero). The conjugate gradient (CG) is an iterative method, meaning that we start with a guess at the solution and in each iteration improve on it, stopping when we think we are accurate enough.

The preconditioned conjugate gradient (PCG) is essentially the same algorithm but uses the following facts to speed up the iterative process: The solution of $Ap = b$ is the same as the solution of $MAp = Mb$, for any invertible matrix M . If M is approximately the inverse of A , so that MA is really close to being the identity matrix, then CG should be able to solve the preconditioned equations $MAp = Mb$ really fast. (as $Ip = b$ is just $p = b$)
This preconditioner M , will be one of the incomplete Cholesky family using the Cholesky factorization: if A is symmetric positive definite then do it so that the two triangular matrices are transposes of each other: $A = LL^T$.
Finally, we use a modified version of incomplete Cholesky to create our preconditioner which will scale better for our particular A.

The $project(\Delta t, \vec{u})$ routine does the following:

1. Calculate the negative divergence b (the right-hand side) with modifications at solid wall boundaries
2. Set the entries of A
3. Construct the MIC(0) preconditioner
4. Solve $Ap = b$ with MICCG(0)
5. Compute the new velocities $\vec{u}^{n+1}$ according to the pressure-gradient update to $\vec{u}$

Finding the pressure values of the fluid volume is called a pressure projection, because both algebraically and physically it satisfies the necessary conditions. A projection is a special type of linear operator such that if you apply it twice, you get the same result as applying it once. In our case, the resulting velocity field, $\vec{u}^{n+1}$, has discrete divergence equal to zero. So if we repeat the pressure step with this as input, we will first evaluate $b = -\nabla \cdot \vec{u} = 0$, which would give constant pressures and no change to the velocity, thus providing the same result and proving that this routine is a projection.

3.7 Smoke

Our fluid in this case is the air in which the smoke particles are suspended. To model the most important effects of smoke, we need two extra fluid variables:

1. the temperature T of the air
2. the concentration s of smoke particles (soot).

We use units of Celsius for temperature and keep s a number between zero (no smoke) and one (as thick as possible). Temperature and soot particles are both advected with the fluid, using the material derivatives:

$$\frac{DT}{Dt} = 0 \quad \text{and} \quad \frac{Ds}{Dt} = 0 \ .$$

We have to study which effect T and s have on velocity. We know that hot air rises and cool air sinks. Similarly it seems plausible that air laden with heavier soot particles will be pulled downwards by gravity. We can model this by replacing the acceleration $\vec{g}$ due to gravity in the momentum equation with a buoyant acceleration:

$$b = [\alpha s - \beta(T - T_{amb})]\vec{g} \tag{18}$$

where α and β are non-negative coefficients, and T_{amb} is the ambient temperature (e.g. 20 C) This formula reduces to zero wherever $s = 0$ and $T = T_{amb} = 30C$.

3.7.1 Smoke sources

We generally want to define some regions that at each timestep we add smoke and heat. This can be implemented at each grid point inside a source as an update after advection:

$$T^{new}_{i,j.k} = T_{i,j.k} + (1 - e^{r_T \Delta t})(T_{target} - T_{i,j,k}) \tag{19}$$

$$s^{new}_{i,j.k} = s_{i,j.k} + r_s \Delta t \tag{20}$$

where r_t and r_s are used to control the rate at which we add smoke and heat (both should be zero outside of the sources) and T_{target} gives the target temperature at the source.

3.7.2 Heat and smoke diffusion

Heat and smoke concentration both can diffuse as well, where very small-scale phenomena such as conduction or Brownian motion, together with slightly larger scale processes such as

turbulent mixing, serve to smooth out steep gradients. This can be modeled with a Laplacian term like viscosity:

$\frac{DT}{Dt} = k_t \nabla \cdot \nabla T$ and $\frac{DT}{Dt} = k_t \nabla \cdot \nabla T$,

where k_T and k_S are non-negative diffusion constants.

For example temperature diffusion in three dimensions when discretized will be:

$$T_{i,j,k}^{new} = T_{i,j,j} + \Delta t \, k_t \times \frac{\left(T_{i+1,j,k} + T_{i-1,j,k} + T_{i,j+1,k} + T_{i,j-1,k} + T_{i,j,k+1} + T_{i,j,k-1} - 6\, T_{i,j,k}\right)}{\Delta x^2}$$

3.7.3 Vorticity confinement

The vorticity confinement technique developed by [48] is a modification of the Navier-Stokes equations by a term that tries to preserve vorticity. [18] introduced it to graphics, introducing a Δx factor so that in the limit (as the grid is refined) the term disappears and we get back the true fluid solution. The underlying idea is to detect where vortices are located and add a body force to boost the rotational motion around each vortex. In this context, a vortex is loosely speaking a peak in the vorticity field, a place that is spinning faster than all the fluid nearby. We can construct unit vectors $\vec{N}$ that point to these maximum points simply by normalizing the gradient of $\|\vec{\omega}\|$:

$$\vec{N} = \frac{\nabla \|\vec{\omega}\|}{\|\nabla \|\vec{\omega}\|\|} \tag{21}$$

Now $\vec{N}$ points towards the center of rotation of a vortex, and $\vec{\omega}$ itself points along the axis of rotation, so to get a force vector that increases the rotation, we just take a cross-product:

$$f_{conf} = \varepsilon \Delta x (\vec{N} \, x \, \vec{\omega}) \tag{22}$$

The ε here is a parameter that can be adjusted to control the effect of vorticity confinement. The Δx factor, as mentioned above, makes this physically consistent: as we refine the grid and Δx tends to zero, the erroneous numerical dissipation of vorticity also tends to zero so our fix should too.

3.7.4 Divergence Control

By using this buoyancy model we use the assumption that fluid density is a function of temperature and smoke concentration (the soot actually dissolves in the air). In the thermal

expansion problem, fundamentally we do want the fluid to expand as it heats up and contract as it cools down. In other words, where the change in temperature is $\frac{DT}{Dt} \neq 0$, we do not want a divergence-free velocity field. For example, if the temperature increases, leading to a decrease in density, we need positive divergence to enact that thermal expansion. We can do that by defining a control field $d(\vec{x})$ at the grid cell centers equal to the desired rate of fractional volume change $\frac{\Delta V}{V}\Delta t$ throughout the volume of the fluid and then by solving for pressure to enforce that divergence. In our numerical discretization in which we use a MAC grid during the pressure solve, we add d to the fluid cells in the right-hand side of the linear system of equation (15).

3.8 Surface tracking

In the case of liquids, like water, we need to find a way to track the fluid's surface. This is done by using a implicit surface representation or a combination of different implicit representations.

3.8.1 Tracker particles

We begin the simulation by filling the volume of water with particles and viewing them as a sampling of the water geometry (and if there are sources adding water during the course of the simulation, we seed particles from them as well). Within each advection step we move the particles according to the grid velocity field, so that they naturally follow where the water should be going. Finally, any cell containing a marker particle is water, and the rest of the non-solid cells are left empty by default. During rendering we want a smooth surface between water and air so we construct a smooth implicit surface wrapped around the particles, e.g.:

$$\varphi(\vec{x}) = \|\vec{x} - \bar{X}\| - \bar{r}, \tag{23}$$

where $\bar{X}$ is a weighted average of nearby particle locations and $\bar{r}$ is a similar weighted average of nearby particle radii. Finally, the surface is defined as the points $\vec{x}$ where $\varphi(\vec{x}) = 0$, the zero isocontour or level set of φ.

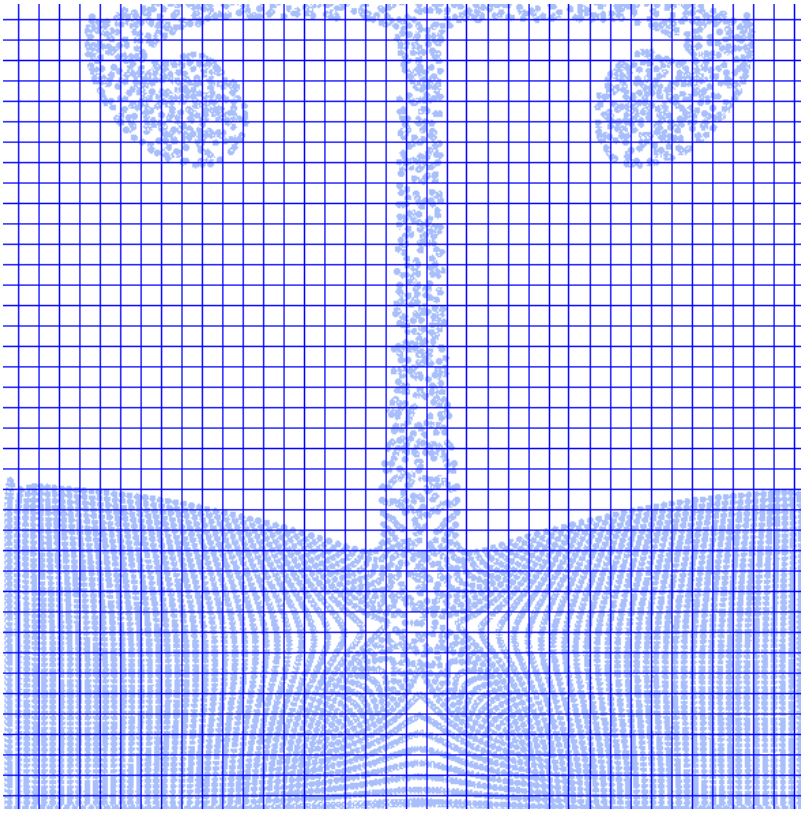

Figure 5: Two dimensional simulation which uses marker particles to track and render the fluid

3.8.2 Level Set Method

In mathematics, a level set of a real-valued function f of n variables is a set of the form $L_c(f) = \{(x_1,\ldots,x_n) \mid f(x_1,\ldots,x_n) = c\}$. That is, a set where the function takes on a given constant value c .

Instead of building an implicit function around particles, like in the previous method, we can work with the grid directly. We define the implicit surface function $\varphi_{i,j,k}$ at the centers of simulation grid cells. Interpolation can be used to estimate $\varphi(\vec{x})$ in between cell centers. The surface is taken to be the points where $\varphi(\vec{x}) = 0$, the inside of the surface is where $\varphi(\vec{x}) < 0$, and the outside of the surface is where $\varphi(\vec{x}) > 0$.

Signed distance

Given any closed set S of points, the distance function for the set is:

$$distance(\vec{x}) = min_{\vec{p}\in S}(\|\vec{x}-\vec{p}\|) \tag{24}$$

that is, it is the distance to the closest point in S . If S divides space into a well-defined inside and outside, then the signed distance function is :

$$\varphi(\vec{x}) = \begin{cases} distance(\vec{x}) : \vec{x} \ is \ outside, \\ -distance(\vec{x}) : \vec{x} \ is \ inside \end{cases}$$

Both equations (23) and (38) can be used to implicitly describe the underling surface: S is the set of points where $distance(\vec{x})$ or $\varphi(\vec{x})$ are zero, the zero level set or isocontour.

Signed distance properties

1. For example, at some point $\vec{x}$ inside the surface, let $\vec{n}$ be the unit-length direction toward the closest point on the surface. For positive small enough, the signed distance $\varphi(\vec{x}+\varepsilon\vec{n})$ must be $\varphi(\vec{x})+\varepsilon$ because if we move along this direction $\vec{n}$, my closest point on the surface will not change, and the distance to it changes by exactly how much we move. Therefore:

$$\nabla\vec{\varphi}\cdot\hat{n} = 1 \tag{25}$$

2. The fastest way to increase or decrease the distance to the surface is by moving along the direction to the closest point. Thus, the gradient of φ , must is the direction $\vec{n}$ to the closest point on the surface:

$$\hat{n} = \nabla\varphi \tag{26}$$

3. Putting the two previous equations together:

$$\|\nabla \vec{\varphi}\| = 1 \tag{27}$$

4. For any given point the closest point on the surface is located at:

$$closestPoint(\vec{x}) = \vec{x} - \varphi(\vec{x})\nabla\varphi(\vec{x}) \tag{28}$$

Calculating the signed distance

In order to calculate the signed distance field we use a geometric method in which we directly compute the distance between each cell and the surface.

The geometry-based signed distance construction algorithm-which efficiently propagates information about the surface out to grid points far away, without requiring expensive geometric searches for every single grid point- is given below:

1. Find the closest points on the surface for the nearby grid points, setting their signed distance values accordingly. Set the other grid points as unknown.

2. Loop over the unknown grid points (i, j, k) in a chosen order:

   i. Find all neighboring grid points that have a known signed distance and closest surface point.

   ii. Find the distances from $x_{i,j,k}$ to those surface points. If is closer $x_{i,j,k}$ than a neighbor, mark the neighbor as unknown again.

   iii. Take the minimum of the distances, determine if (i, j, k) is inside or outside, and set its signed distance value accordingly.

The loop order used to traverse through the grid points is determined by using the fast marching method which is based on the realization that grid points should only get information about the distance to the surface from points that are closer. We thus loop over the grid points going from the closest to the furthest. This can be facilitated by storing unknown grid points in a priority queue shorted by the current estimate of their distance.

Advecting Level Set

We then have to move the surface by advecting the level set. This is done by solving: $\frac{D\varphi}{dt} = 0$. Advecting a signed distance field does not in general preserve the signed distance property. We thus need to periodically recalculate signed distance.

Until now we have resolved the free surface boundary condition by setting $\varphi = 0$ at water-air boundary. We also need to decide the solid wall boundary condition for φ . A possible approach is to extrapolate φ from the water-air region into the solid, virtually extending the surface. The most straightforward approach to extrapolation is to directly use the closest point definition. When extrapolating velocity we are working on staggered grid locations, not the grid points where we computed closest elements so we can simply average from the neighboring grid cell centers, or find the closest of the two elements stored at the neighboring grid cell centers.

Current Fluid Simulation Loop

1. If necessary reinitialize the fluid/air signed distance function φ to signed distance.
2. Extrapolate the divergence-free velocity field from the last time step outside the fluid, to get an extended velocity field $\vec{u}^n$.
3. Add body forces such as gravity or targeting to get an interim velocity field.
4. Advect the interim velocity field and φ in the divergence-free $\vec{u}^n$ velocity field.
5. Solve for and apply pressure to get a divergence-free velocity in $\vec{u}^{n+1}$ the new fluid region.

3.8.3 Particle Level Set

In Particle Level Set (PLS) a Eulerian level set formulation is augmented with helper marker particles to track material boundaries, like the free surface of a liquid. PLS begins with a grid-sampled level set $\varphi_{i,j,k}$. As only the values of φ near the zero isocontour define the surface, only there we will add particles to accurately advect these grid cells. To avoid bias, particles are added on both sides of the interface, φ positive and negative. Each particle, in addition to its position, includes its own estimate of the φ value. A particle's initial φ_p value can be interpolated from the grid. Both the grid level set values and the particles are advected forward each time step. A series of error-correction passes are applied, where the particles attempt to correct numerical dissipation errors in the grid, but features that ultimately cannot be represented on the grid are removed from the particles. This error-correction process begins with identifying escaped particles: particles where φ interpolated from the grid is a different sign from their own φ_p and of larger magnitude. These are cases where an air particles ended up inside the water, or a water particle ended up inside the air, and have gone further than their own radius. Lastly, we delete particles that are too far from the zero level set and add new particles in each of the cells close to the material boundary that have less particles than a certain maximum value.

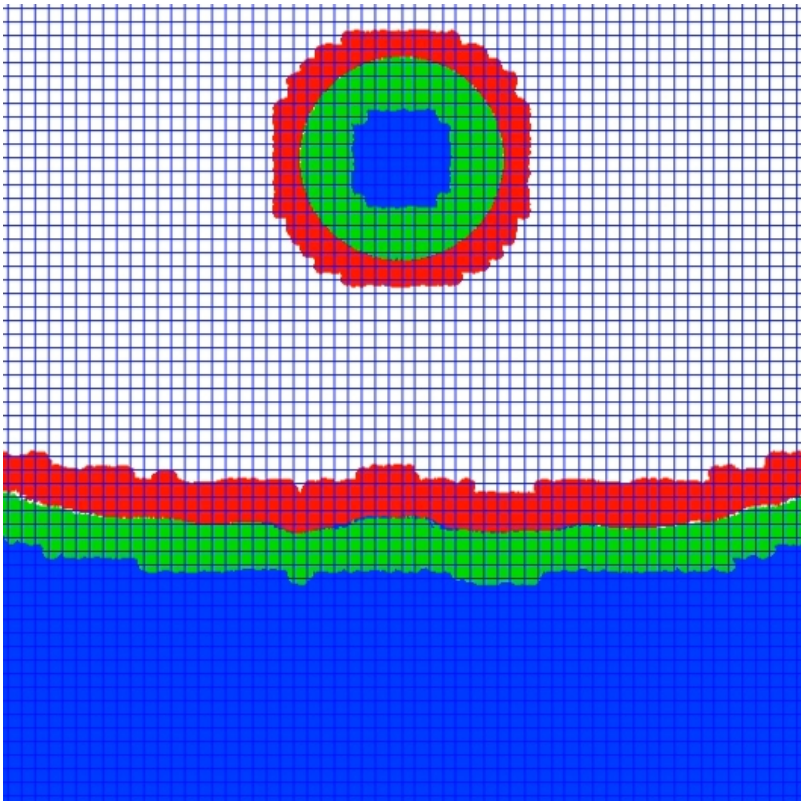

Figure 6: Two dimensional fluid simulation that uses Particle Level Set method to track the fluid. Red particles are seeded in air and green particles are seeded inside water.

3.9 Accurate Pressure Solves

3.9.1 Ghost fluid method

Even if we can track an accurate water surface, the pressure solve only sees a block voxelized surface, with some cells marked as water and some as air. Thus the velocity field, and from there the motion and shape of the surface itself, will experience significant voxel artifacts. In order to improve the visual plausibility of our simulation we have to provide a more elegant solution for the free surface boundary condition instead of just setting $p = 0$ at the water-air boundaries.

Between two grid cells (i, j) with $\varphi_{i,j} \leq 0$ and $(i+1, j)$ with $\varphi_{i,j} > 0$, instead of setting $p_{i+1,j} = 0$, it would be more accurate to say that $p = 0$ at the water-air interface, which is somewhere between $\varphi_{i,j}$ and $\varphi_{i+1,j}$.

Linearly interpolating between $\varphi_{i,j}$ and $\varphi_{i+1,j}$ gives the location of the interface at $(i+\theta\Delta x, j)$ where:

$$\theta = \frac{\varphi_{i,j}}{\varphi_{i,j} - \varphi_{i+1,j}} \tag{29}$$

Linearly interpolating between the real pressure $p_{i,j}$ and a fictional "ghost" pressure $p^G_{i+1,j}$ then setting it equal to zero at the interface, gives :

$$\begin{aligned} (1-\theta)\, p_{i,j} + \theta\, p^G_{i+1,j} &= 0 \Rightarrow \\ p^G_{i+1,j} &= -\frac{1-\theta}{\theta}\, p_{i,j} \Rightarrow \\ p^G_{i+1,j} &= \frac{\varphi_{i+1,j}}{\varphi_{i,j}}\, p_{i,j} \end{aligned} \tag{30}$$

Recall the velocity update (where $(i+1, j)$ is an air cell):

$$\begin{aligned} u^n_{i,j} &= u_{i,j} - \frac{\Delta t}{\rho} \nabla p \Rightarrow \\ u^n_{i,j} &= u_{i,j} - \frac{\Delta t}{\rho} \frac{p^G_{i+1,j} - p_{i,j}}{\Delta x} \end{aligned} \tag{31}$$

Now plug the ghost pressure in the above equation:

$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho}\frac{p_{i+1,j}^{G} - p_{i,j}}{\Delta x} \Rightarrow$$

$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho}\frac{\frac{\varphi_{i+1,j}}{\varphi_{i,j}}p_{i,j} - p_{i,j}}{\Delta x} \Rightarrow \quad (32)$$

$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho}\frac{\varphi_{i+1,j} - \varphi_{i,j}}{\varphi_{i,j}}\frac{p_{i,j}}{\Delta x}$$

By introducing the update in equation (30) in the pressure solve and the update of equation (32) in the velocity update we provide a better solution for the free surface boundary condition and thus treat the voxelized treatment of the interface.

Another alternative is to split each cell into subcells and calculate the actual density of each one of them using the real water and air densities. Then, by interpolating the density of each subcell we can get the overall density of each cell and set $p=0$ where we detected an air cell (by leaving p untouched we can have a more accurate solve for both water and air pressures, and thus also get a realistic simulation of air bubbles).

3.9.2 Surface tension

The physical chemistry of surface tension is conceptually simple. Water molecules are more attracted to other water molecules than to air molecules, and vice versa. Thus, water molecules near the surface tend to be pulled in towards the rest of the water molecules and vice versa. In a way they are seeking to minimize the area exposed to the other fluid, bunching up around their own type. The simplest linear model of surface tension can in fact be phrased in terms of a potential energy equal to a surface-tension coefficient γ times the surface area between the two fluids. The force seeks to minimize the surface area.

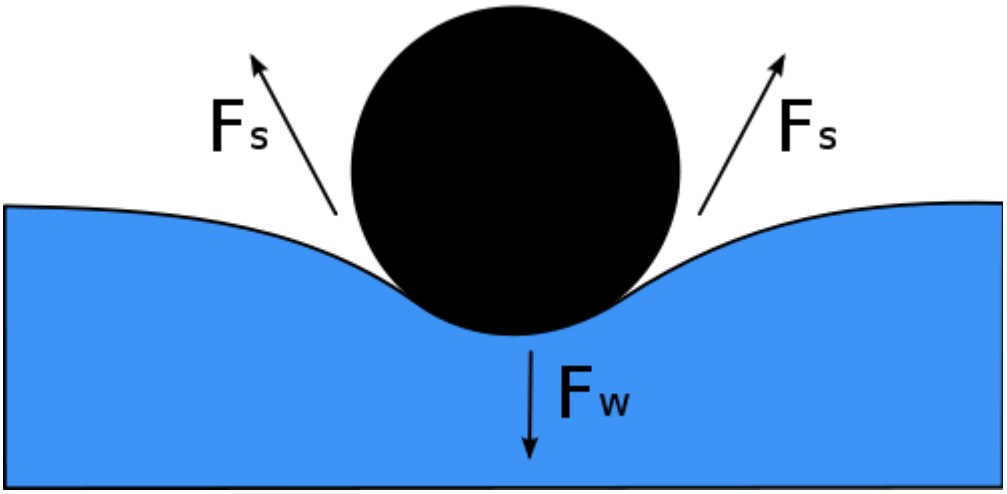

Figure 7: Diagram shows, in cross-section, a needle floating on the surface of water. Its weight, F_w , depresses the surface, and is balanced by the surface tension forces on either side, F_s , which are each parallel to the water's surface at the points where it contacts the needle. Notice that the horizontal components of the two F_s arrows point in opposite directions, so they cancel each other, but the vertical components point in the same direction and therefore add up to balance F_w

The surface area of the fluid is simply the integral of 1 on the boundary:

$$A = \iint_{\partial\Omega} 1 \tag{33}$$

Remembering our signed distance properties from the 3.8.2 Level Set Method, this is the same as:

$$A = \iint_{\partial\Omega} \nabla\varphi \cdot \hat{n} \tag{34}$$

Now we use the divergence theorem in reverse to turn this into a volume

integral:

$$A = \iiint_{\Omega} \nabla \cdot \nabla \varphi \tag{35}$$

Consider a virtual infinitesimal displacement of the surface, δx . This changes the volume integral by adding or subtracting infinitesimal amounts of the integrand along the boundary. The resulting infinitesimal change in surface area is

$$A = \iint_{\partial\Omega} (\nabla \cdot \nabla \varphi) \delta x \cdot \hat{n} \tag{36}$$

Thus, the variational derivative of surface area is $(\nabla \cdot \nabla \varphi)\hat{n}$. Our surface- tension force is proportional to this, and since it is in the normal direction we can think of it in terms of a pressure jump at the air water interface pressure only applies in the normal direction). Since air pressure is zero in our free surface model, we have that the pressure at the surface of the water is :

$$p = \gamma\kappa \tag{37}$$

where $\kappa = \nabla \cdot \nabla \varphi$ is mean curvature of the surface, that measures how curved a surface is.

Here between two grid cells (i, j) with $\varphi_{i,j} \leq 0$ and $(i+1, j)$ with $\varphi_{i+1,j} > 0$, instead of setting $p_{i+1,j} = 0$, it would be more accurate to say that $p = \gamma\kappa$ at the water-air interface instead of zero.

3.10 Coupling Fluids and Solids

3.10.1 One way coupling

The simulator as described so far covers the one way coupling between fluid and solid by resolving the solid wall boundary condition in the case of a moving solid. In this respect, we already have a solid to fluid one way coupling. There also exist solids that take their motion from the fluid, but do not affect it. This is the second form of one way couplind, namely the fluid to solid coupling. In this case the position $\vec{x}_i$ of the solid follows the fluid velocity field:

$$\frac{d\,\vec{x}_i}{dt} = \vec{u}(\vec{x}_i, t) \tag{38}$$

Instead of only focusing on the solid's position, we also can update their orientations by using the angular velocity measured around the origin is given by:

$$\vec{\Omega} = \frac{1}{2}(\nabla \times \vec{u}) \tag{39}$$

Furthermore, solids can have some inertia. Then for small particles in the flow, we add a simple drag force of the form: $\vec{F}_i = D(\vec{u} - \vec{u}_i)$. D is used to control the amount of the drag force we will add.

3.10.2 Weak Coupling

For objects large or heavy enough to significantly affect the fluid flow, but light enough to be affected in turn by the fluid, we need methods for simulating both. One common approach to implementing this two-way coupling is sometimes termed weak coupling. We interleave the solid and fluid simulation steps. When using solids we get the following algorithm for each time step:

1. advect the fluid, and update the solid positions (and orientations if relevant).
2. add gravity to all velocities
3. solve for the pressure to make the fluid incompressible, enforcing the solid-wall boundary condition with the current solid velocities held fixed.
4. update the solid velocities from forces due to the new fluid pressure and from contact/collision

In our implementation we added the weak coupling method supporting circles and spheres in two and three dimensions respectively.

3.11 Terrain

During the project we also wanted to be able to use arbitrary terrain formations so that the fluid could interact with. We decided to use a heightmap like technique in order to make the terrain easy to build. The following steps are necessary to include terrain in one simulation scene:

1. create a square png image with each dimensions matching the resolution of the simulation grid we will use (e.g. for a 64x64x64 grid we create a 64x64 image)

2. we paint in grayscale the white image in order to create the terrain we want. (white being the maximum and black the minimum height of the grid)

3. we load the respective png image as the simulation begins. Additionally, we store the heightmap that we just loaded to raw file containing the cell heights in a format that is readable by pbrt's heightmap shape. During the rest of the simulation, we treat all the corresponding terrain cells as solid cells so we can have correct solid-fluid interactions.

4. we use the heightmap shape of pbrt to load and render the terrain along with the fluid

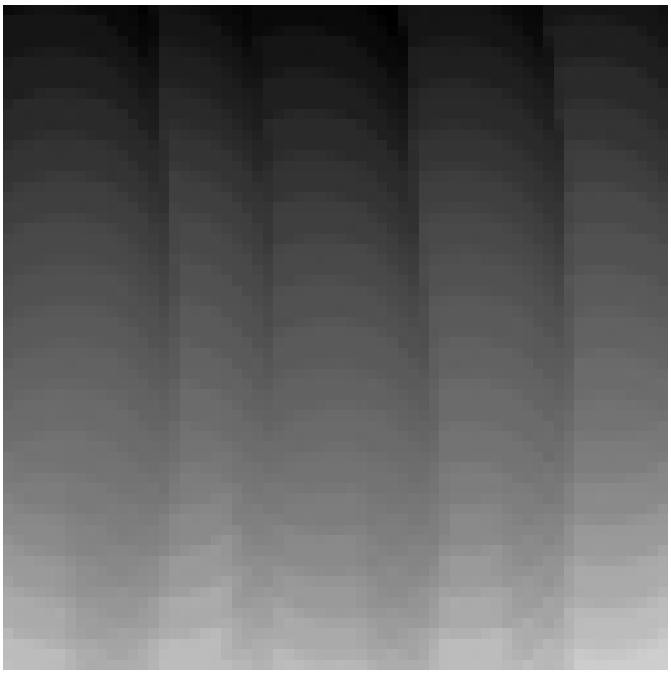

Figure 8: Example of height map

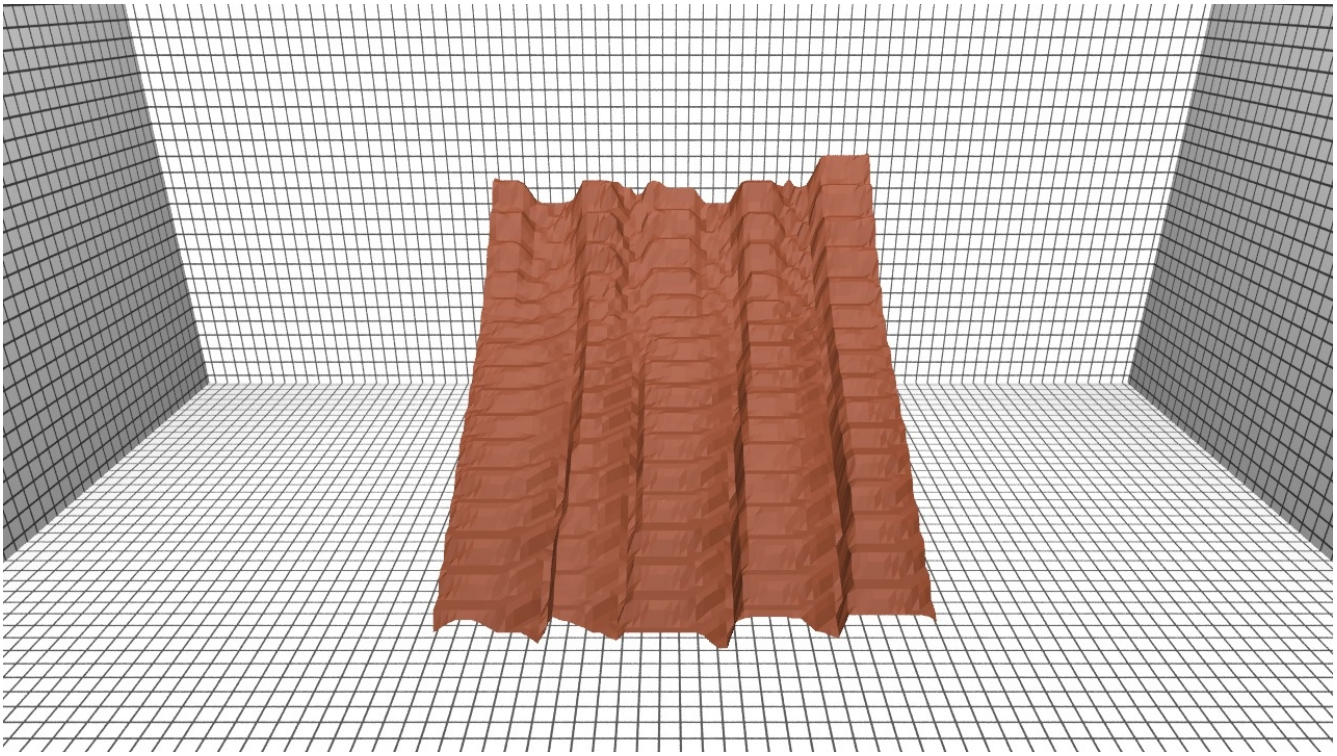

Figure 9: The above heightmap rendered in pbrt as the scene's terrain

3.12 Implementation

The code for this Master's Thesis was written in C++ on Ubuntu 12.04 64bit operating system. The project was written and tested on a laptop with an Intel® Core™ i7 CPU Q 740 and 4GB RAM.

Below you can find the classes used in the code, along with their description and correspondence to the fluid simulation theory presented earlier in this document.

3.12.1 Classes

Grid class: This class implements a MAC grid. It contains all the functions that use variables stored on the grid.

LevelSet class: This class implements a signed distance field used to track the surface of the fluid. Furthermore, it handles the particles used in the case of Particle Level Set method. It can be used to specify the initial fluid volume in the simulation space. (e.g. Load a triangle mesh and convert it to level set information)

Object class: This class implements a solid object in the simulation space.

Renderer class: This class implements the rendering of the fluid. It has many different options (e.g. fluid cells, marching cubes surface, velocity direction etc.). Almost all features exist in both 2D and 3D.

Point class: This class implements a point or vector in the simulation space.

Png folder: The files in this folder contain the implementation of picoPNG, a png decoder made freely available by Lode Vandevenne.

ParticleSystem class: This class implements a particle system, which basically is collection of particles used for the same purpose. (e.g. in the Particle Level Set method)

Utilities folder: This file implements some function that are used by more than one class. (e.g. sign of a number, bilinear/trilinear interpolation)

Solver class: This class implements the MICGG(0) algorithm to be used in the fluid simulation loop. It includes both my implementation and the implementation made freely available from Robert Bridson. The user can specify which solver should be used. Both provide similar performance.

FluidApp class: This class implements the fluid simulation loop.

3.12.2 Rendering options

Below you can find 2D examples of the different rendering options that are available.

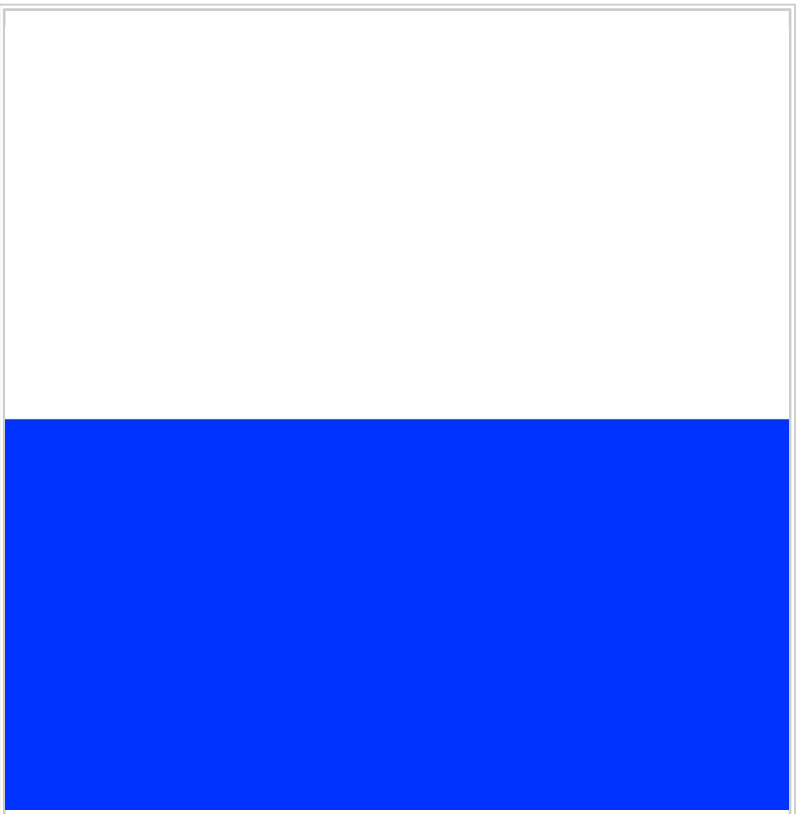

Figure 10: Two dimensional fluid simulation without grid

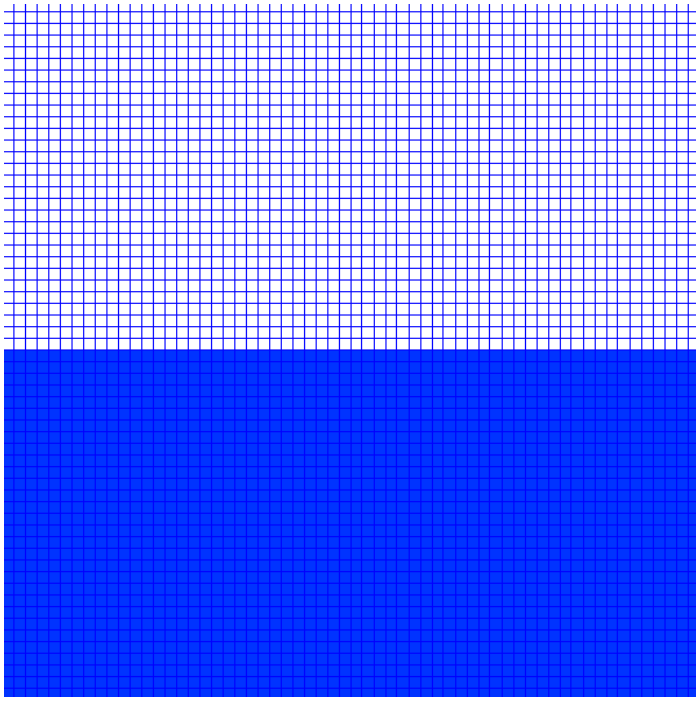

Figure 11: Two dimensional fluid simulation with grid

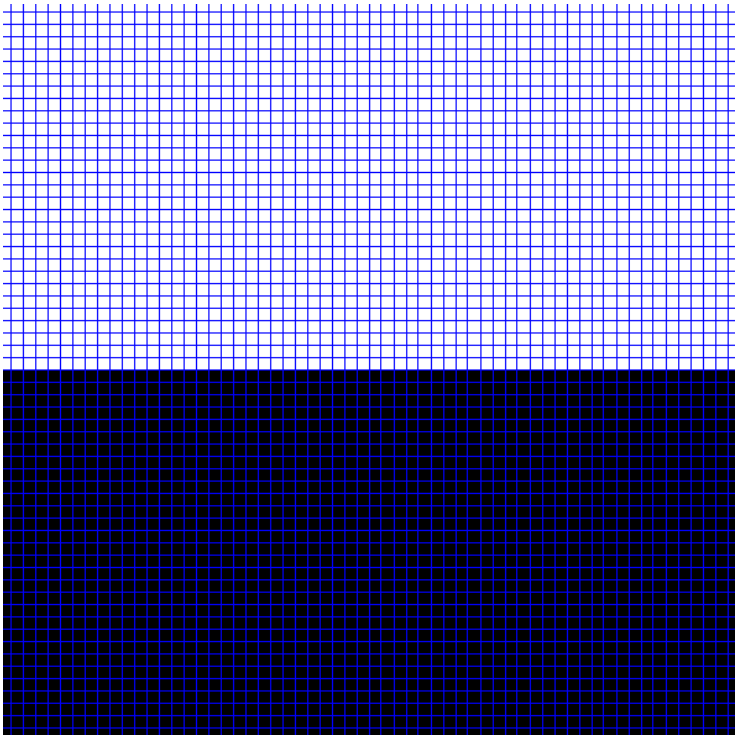

Figure 12: Two dimensional fluid simulation. Black cells indicate fluid while white cells indicate air.

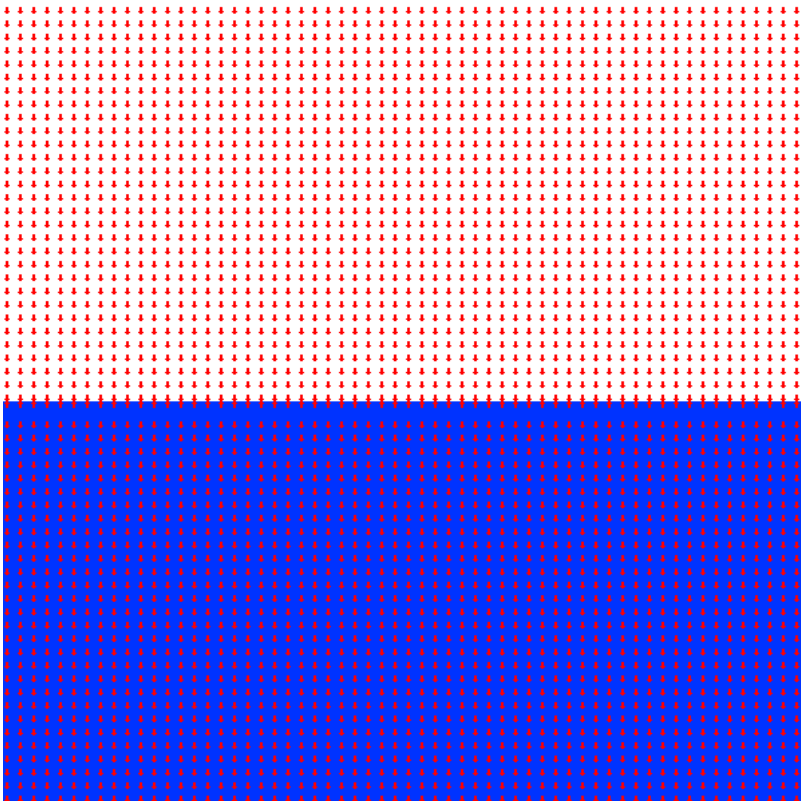

Figure 13: Two dimensional fluid simulation. The red arrows show the direction of the velocity in each cell center.

3.12.3 Results

Two-dimensional fluid simulation

Glut and OpenGL are used to render the simulations in 2D.

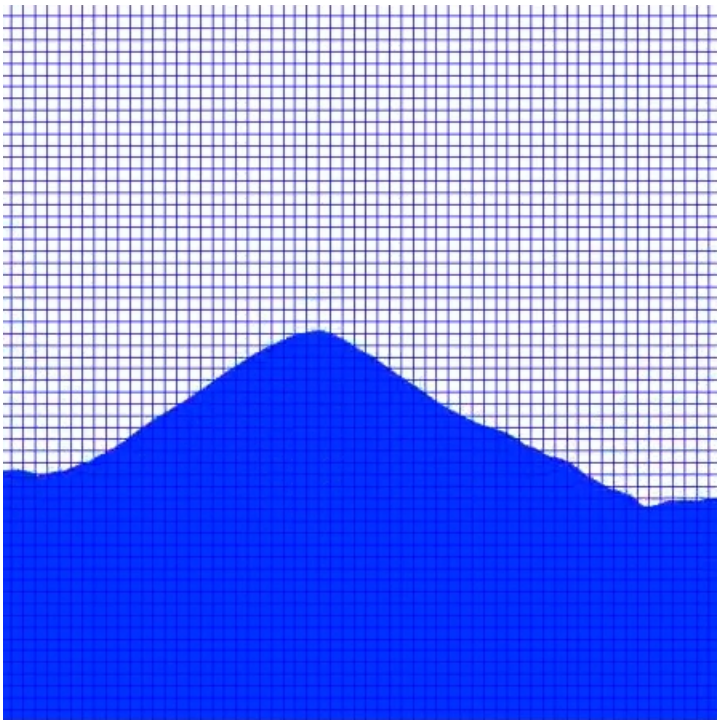

Figure 14: Ball dropping in water tank

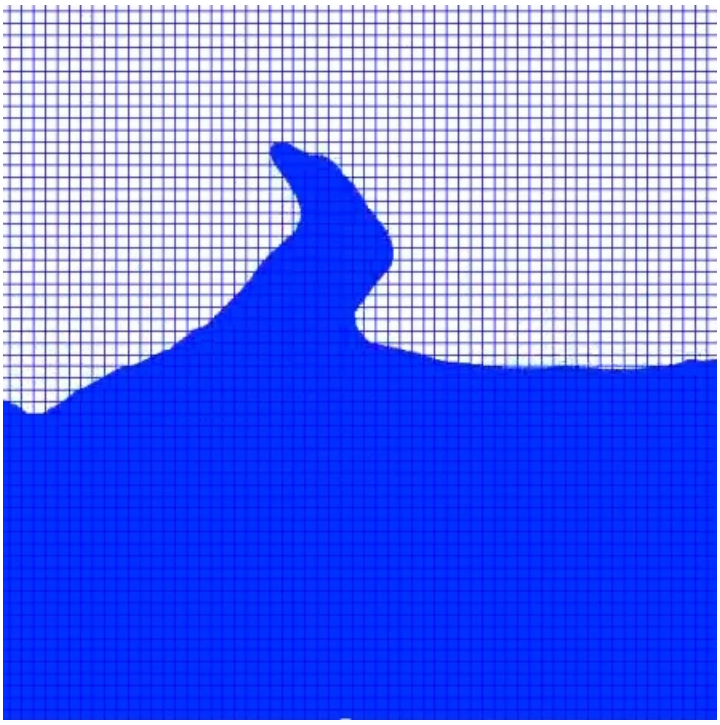

Figure 15:Ball dropping in water tank

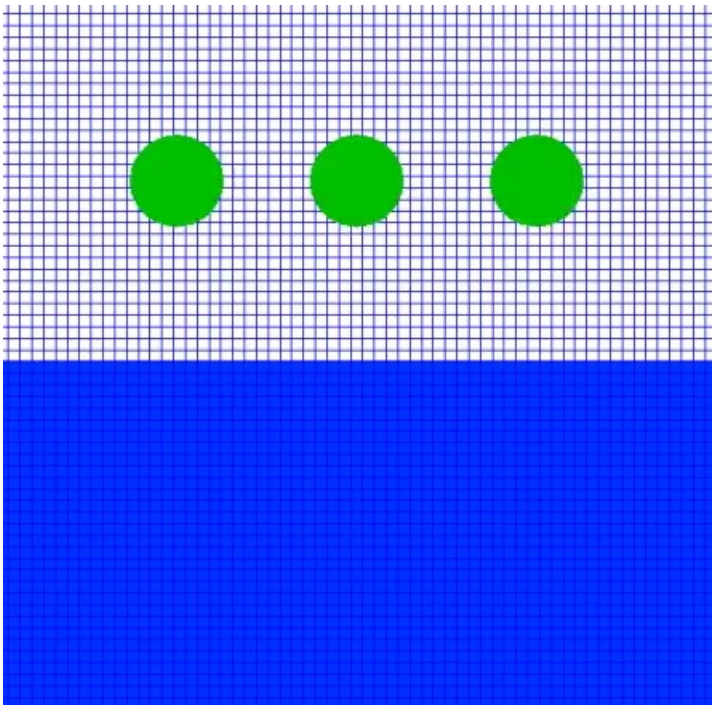

Figure 16: Solid balls drop in a water tank

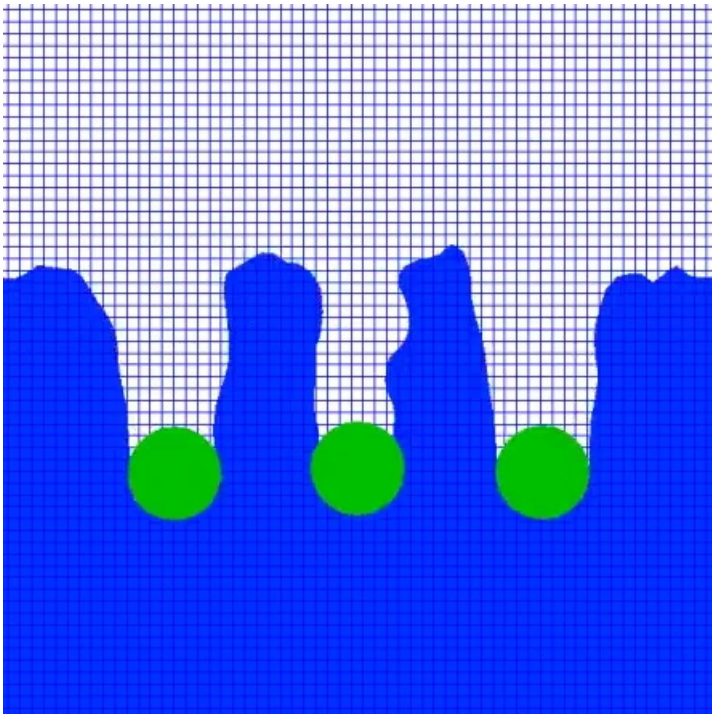

Figure 17: Solid balls drop in a water tank

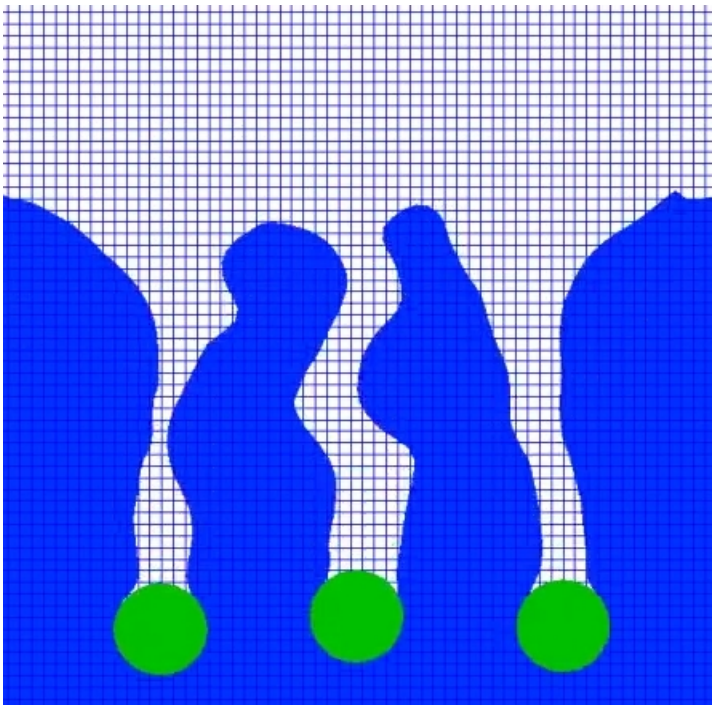

Figure 18: Solid balls drop in a water tank

Three-dimensional fluid simulation

Physically based ray tracer (pbrt) is currently used to render the simulations in 3D. As the name suggests, pbrt is an open-source 3D renderer and relies on external 3D modeling programs to create the scene that needs to be rendered. In addition, the grid-based implicit surface shape plugin for pbrt made by Robert Bridson, is used to define the fluid volume in pbrt given the signed-distance values that describe the surface of the fluid, as described earlier in the level set method.

During the simulation we export the signed-distance field of each frame to a file. After we finish the simulation, we use each saved file as an input to pbrt in order to produce one image per frame. Finally, we combine all these rendered images to generate one video file for each one of the simulations.

Water

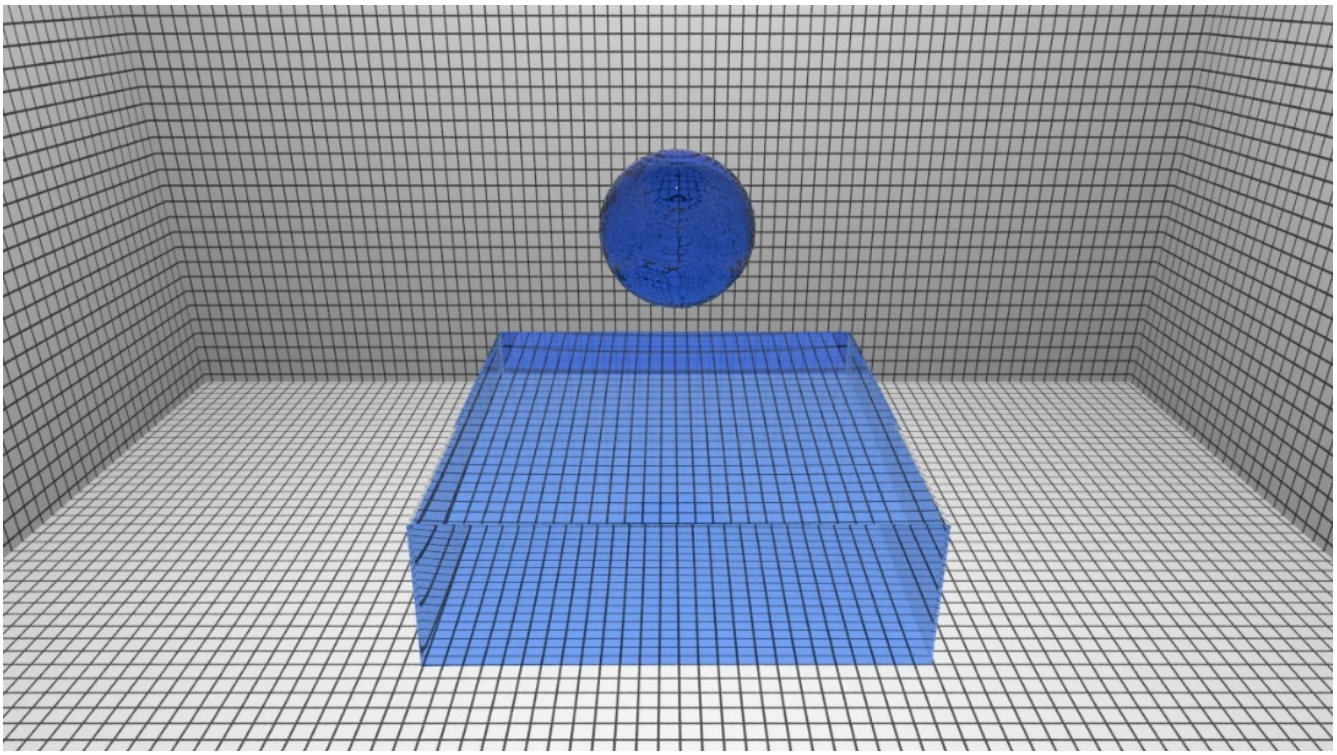

Figure 19: Ball dropping in water tank

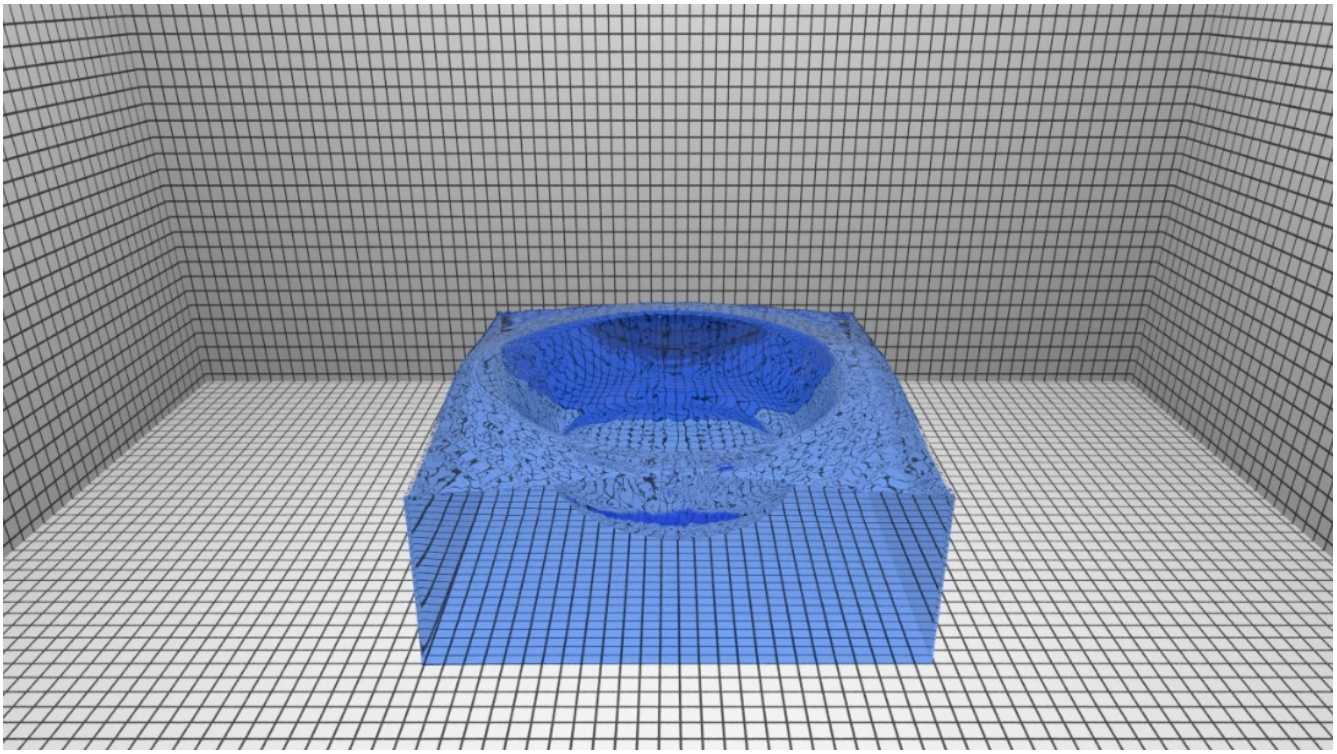

Figure 20: Ball dropping in water tank

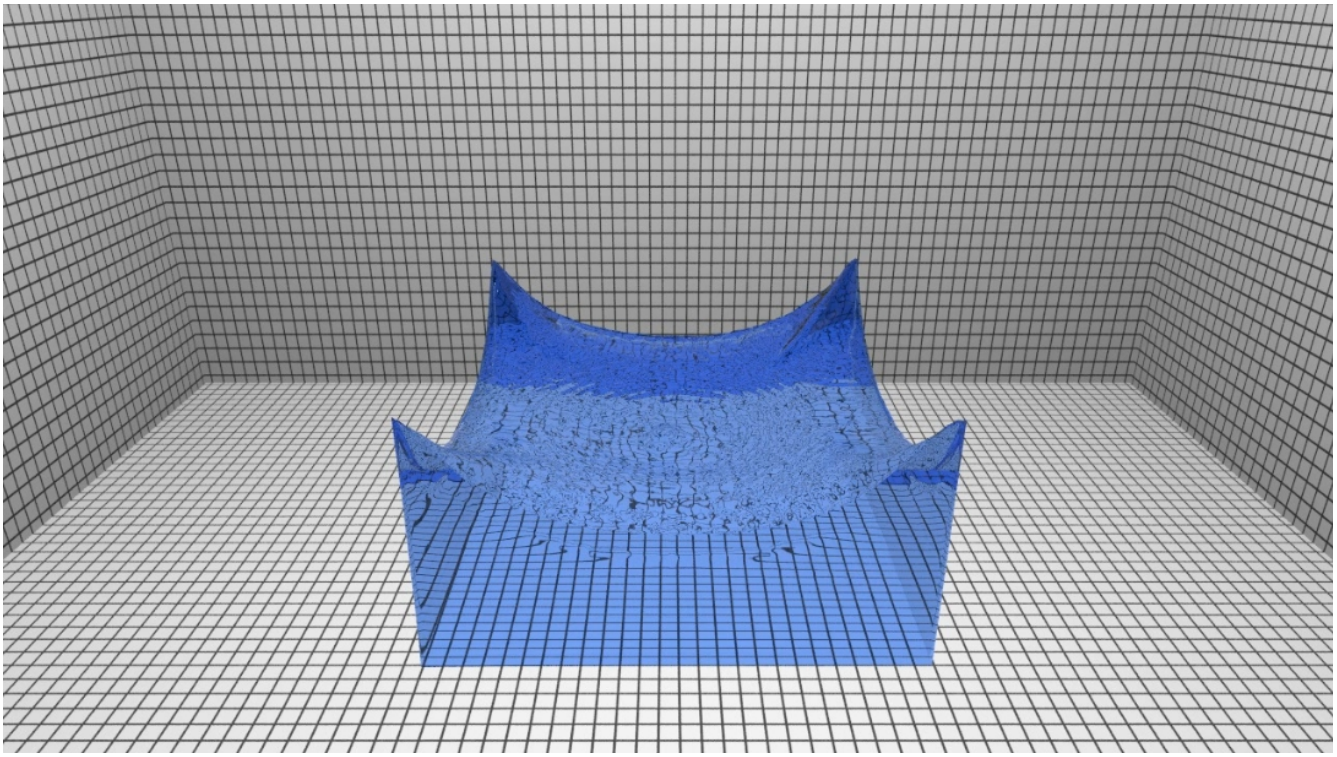

Figure 21: Ball dropping in water tank

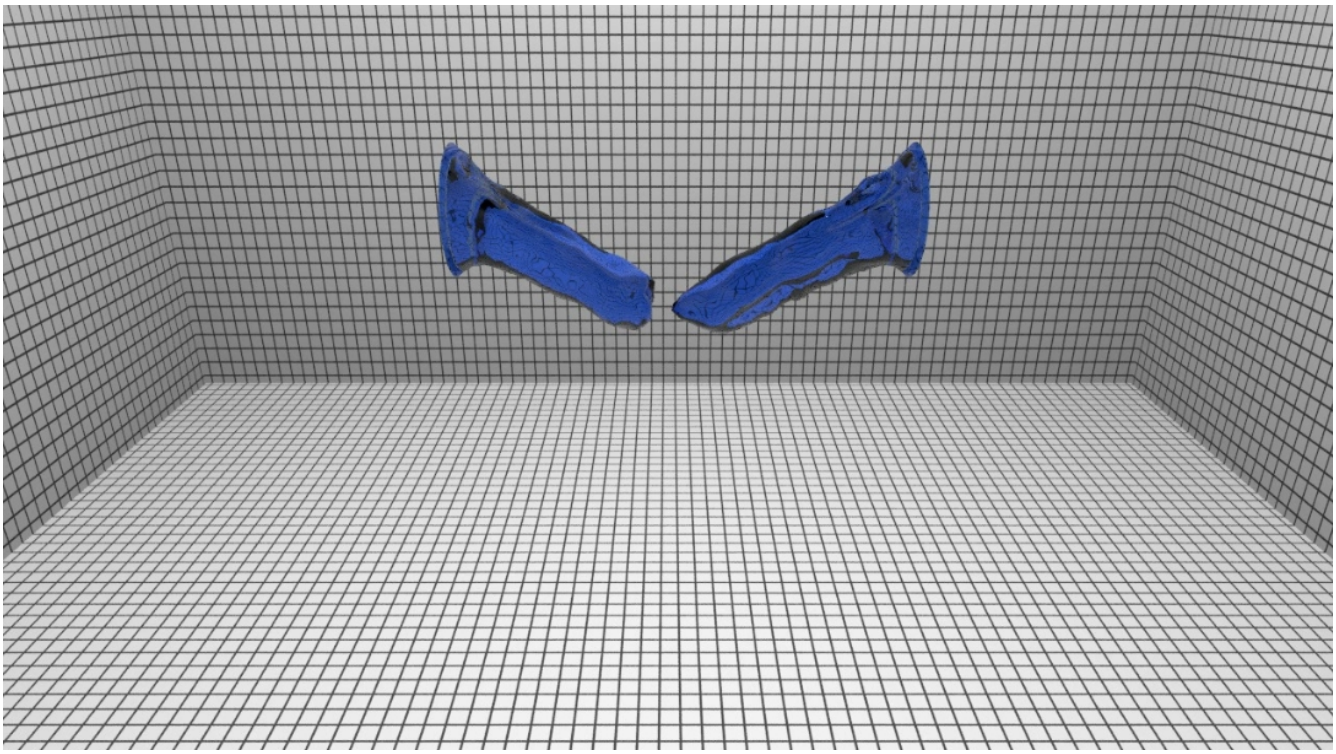

Figure 22: Two water sources

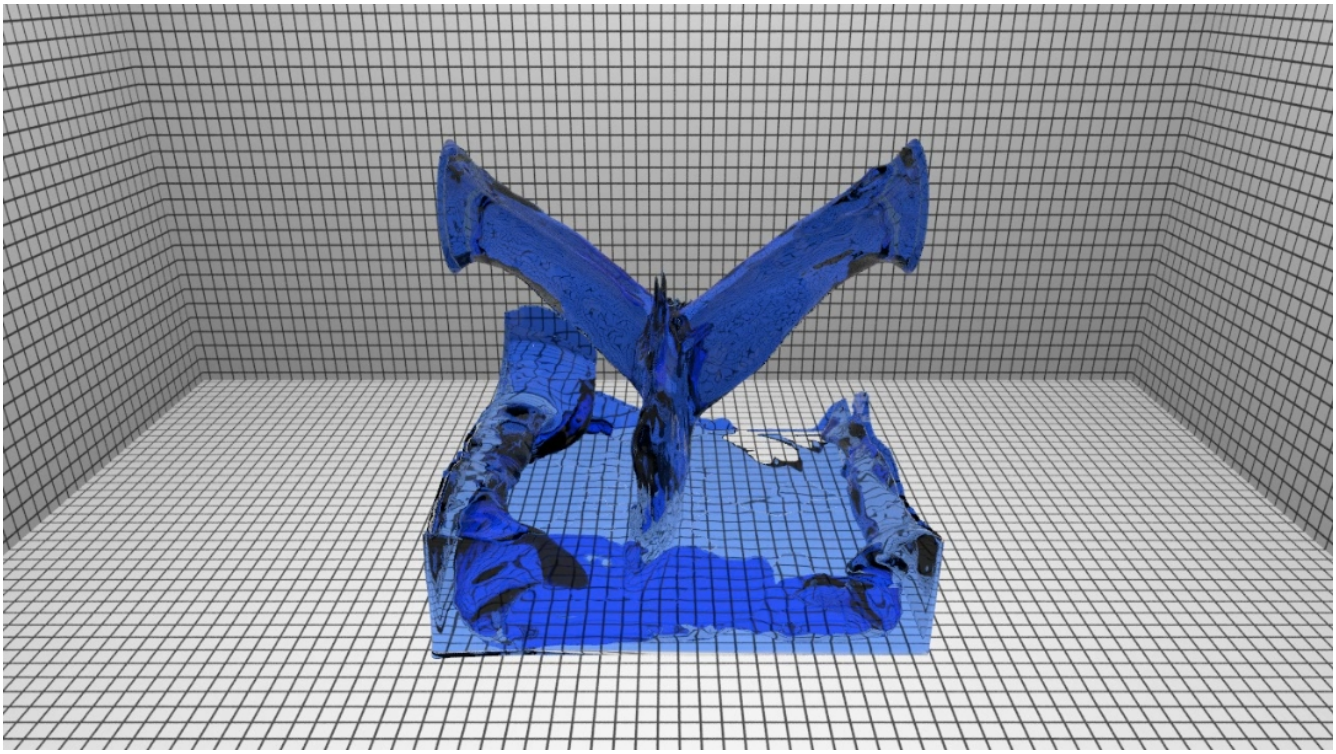

Figure 23: Two water sources

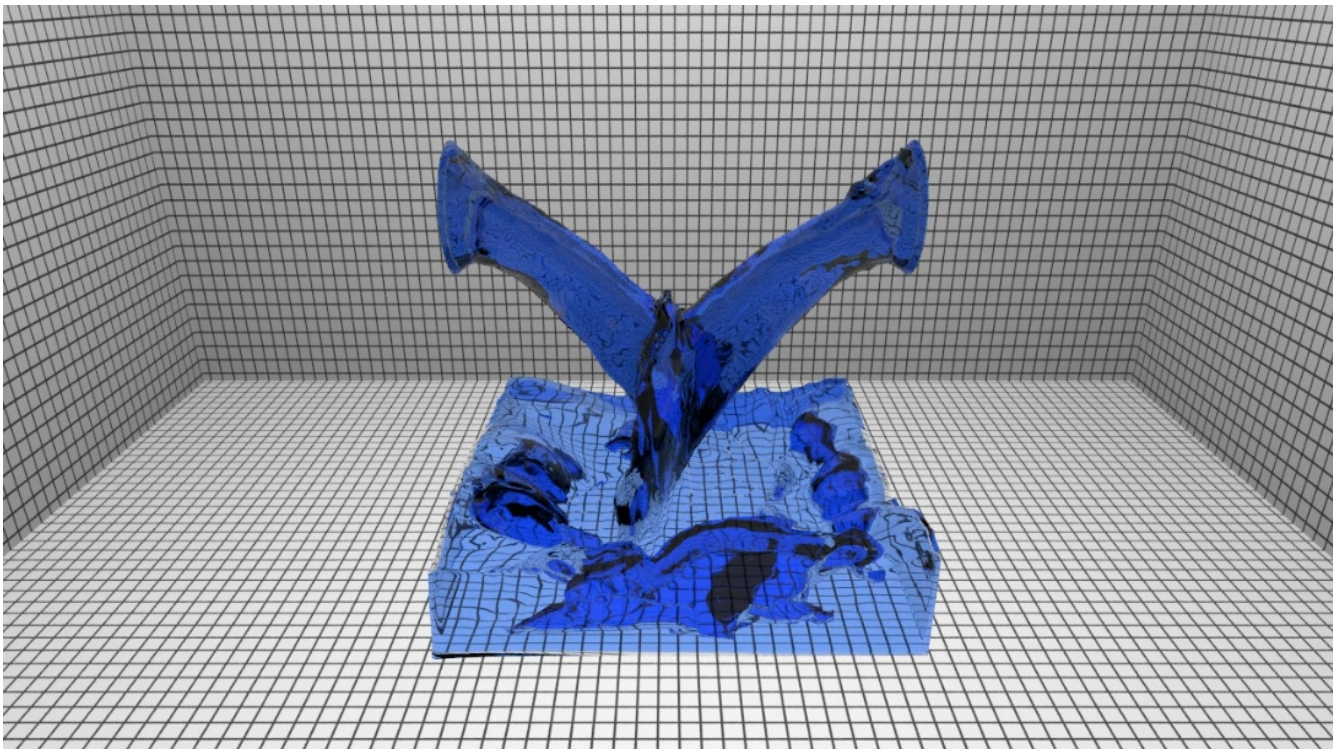

Figure 24: Two water sources

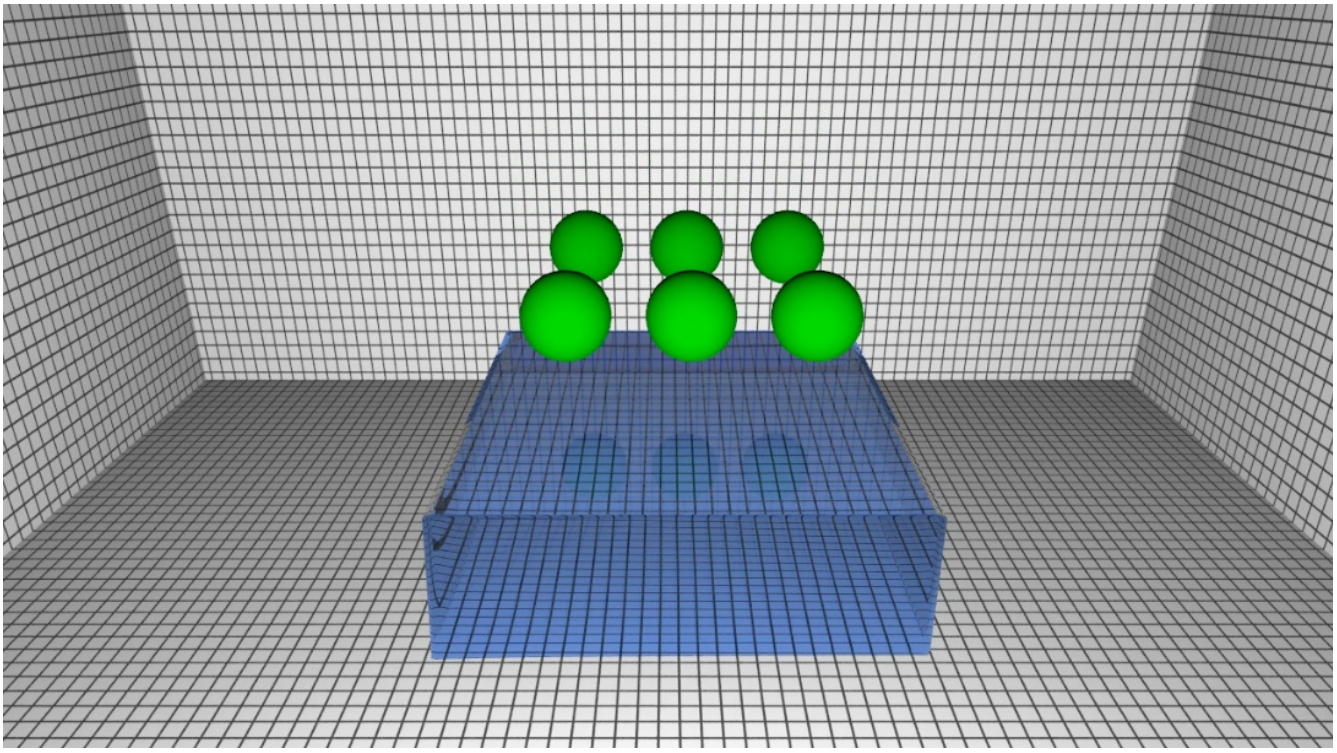

Figure 25: Solid balls drop into a water tank

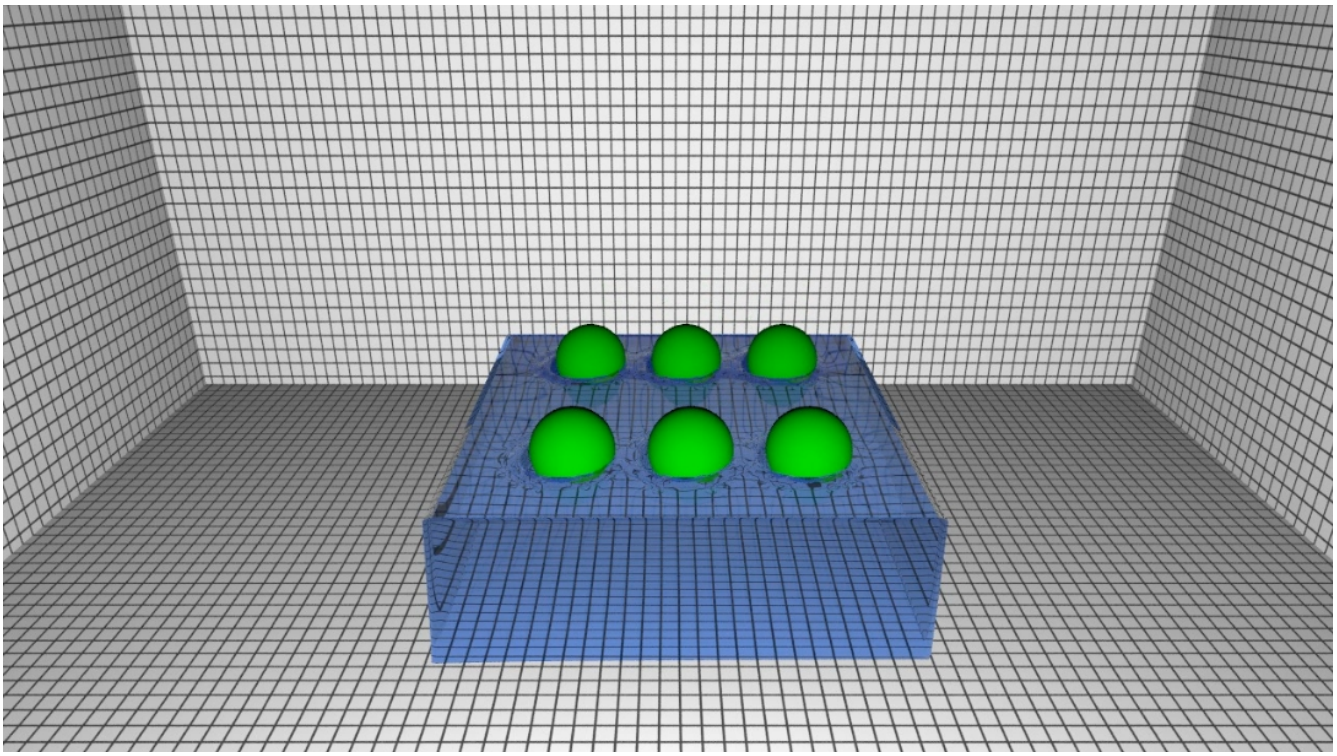

Figure 26: Solid balls drop into a water tank

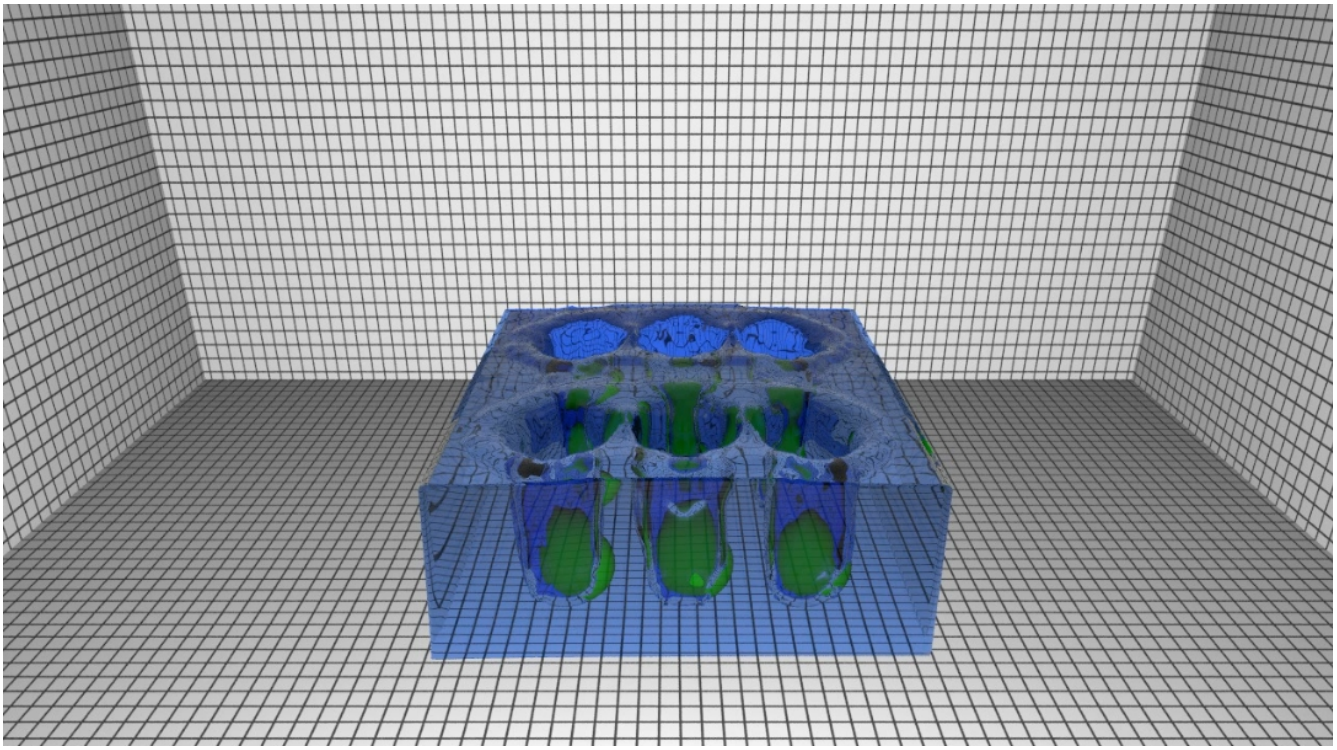

Figure 27: Solid balls drop into a water tank

Smoke

Here are some 2D and 3D smoke simulation results from my fluid simulator. Smoke is not part of my official thesis plan, but as smoke simulation involves a couple more steps compared to the base simulation, I found it interesting to add it to the simulator. Diffusion of smoke and heat concentration along with vorticity confinement are implemented in the simulator. pbrt's volume grid is used for the rendering.

Figure 28: Two-dimensional smoke simulation

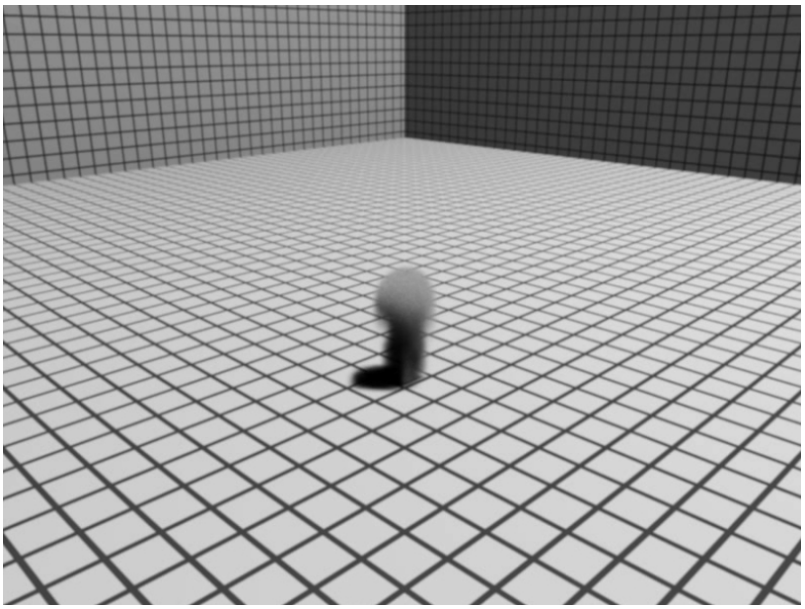

Figure 29: Three-dimensional smoke simulation

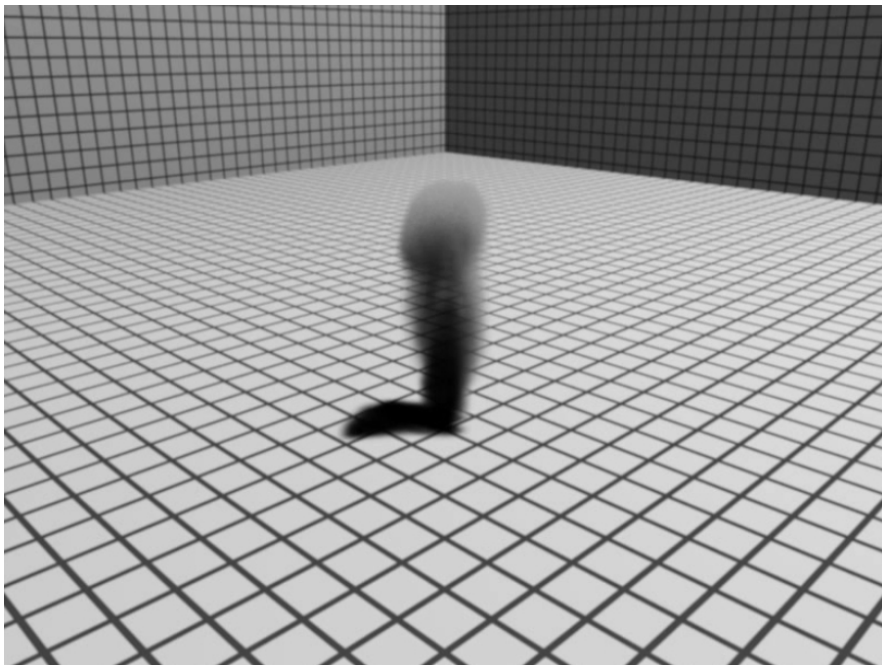

Figure 30: Three-dimensional smoke simulation

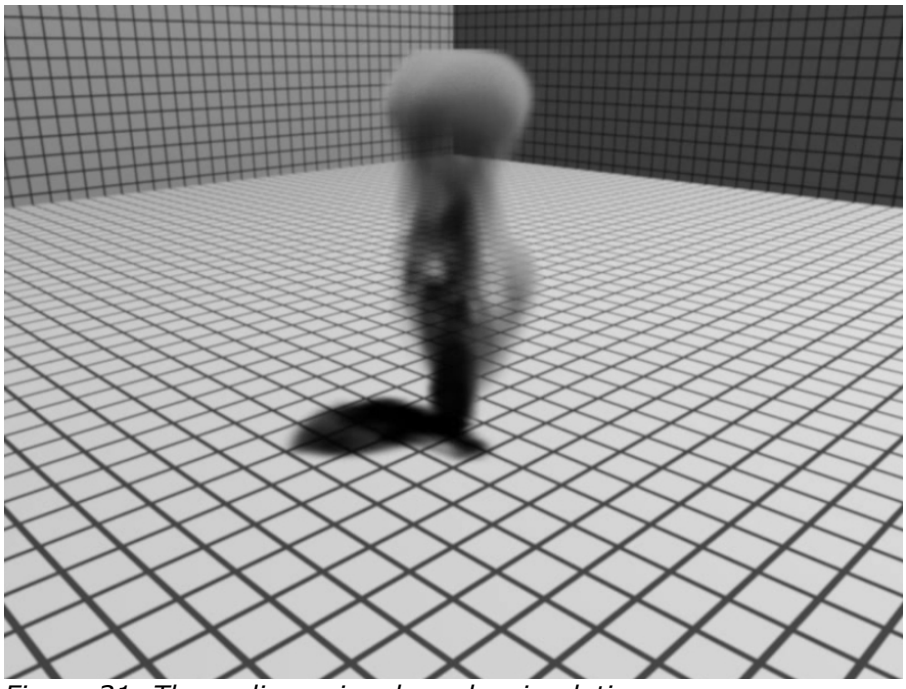

Figure 31: Three-dimensional smoke simulation

Chapter 4

Work focus

4.1 Hybrid grid structure

In this chapter the "Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid" [11] paper will be discussed. Among the contributions this paper introduced, the hybrid grid representation will be discussed in detail. Furthermore, we describe how this hybrid representation is extended to a more general model and added to the fluid simulator we developed. Finally, we present the simulation timings to show the performance increase our method introduced to the base fluid simulator that we developed.

4.1.1 Original method

"Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid" proposes a new Eulerian fluid simulation method, which allows real-time simulations of large scale three dimensional liquids. To reduce computation time they use a hybrid grid representation composed of regular cubic cells on top of a layer of tall cells. With this layout water above an arbitrary terrain can be represented without consuming an excessive amount of memory and compute power, while focusing effort on the area near the surface where it most matters. Additionally, the grid representation for a GPU implementation of the fluid solver was optimized. To further accelerate the simulation, a specialized multi-grid algorithm for solving the Poisson equation was introduced and solver modifications to keep the simulation stable for large time steps were proposed.

In this method liquids are simulated by solving the inviscid Euler equations, subject to the incompressibility constraint. The equations are solved in the domain specified by a scalar level-set field φ in the region where $\varphi < 0$.

4.1.2 Discretization

The discretization of the simulation domain is a specialized tall cell grid in which, from bottom to top, each column consists of terrain, one tall cell and a fixed number of regular cells. Figure 32 shows an example of the tall cell grid in 2D. The terrain height and the height of the tall cell are discretized to be a multiple of the grid spacing Δx . These height values are stored in two arrays. For regular cells, all the physical quantities like velocity, level set value and pressure are stored at the cell center. For tall cells, these quantities are stored at the center of the topmost and the bottommost subcells. In terms of implementation, a quantity q is stored in a compressed 3D array, $q_{i,j,k}$, of size $(Bx, By+2, Bz)$ where Bx and Bz are the number of cells along the x and z axis respectively, B_y is the constant number of regular cells along the y -axis per column and the $+2$ comes from the top and the bottom values stored in per tall cell. In addition, two 2D arrays of size (Bx, Bz) are

used to store the terrain height $H_{i,k}$ and the tall cell height $h_{i,k}$. The y -coordinate of the uncompressed position of an array element $q_{i,j,k}$ is given by :

$$y_{i,j,k} = \begin{cases} H_{i,k} + 1 & if\ j = 1 (tall\ cell\ bottom) \\ H_{i,k} + h_{i,k} & if\ \ j = 2 (tall\ cell\ top) \\ H_{i,j} + h_{i,k} + j - 2 & if\ j \geq 3\ \ (regular\ cell) \end{cases}$$

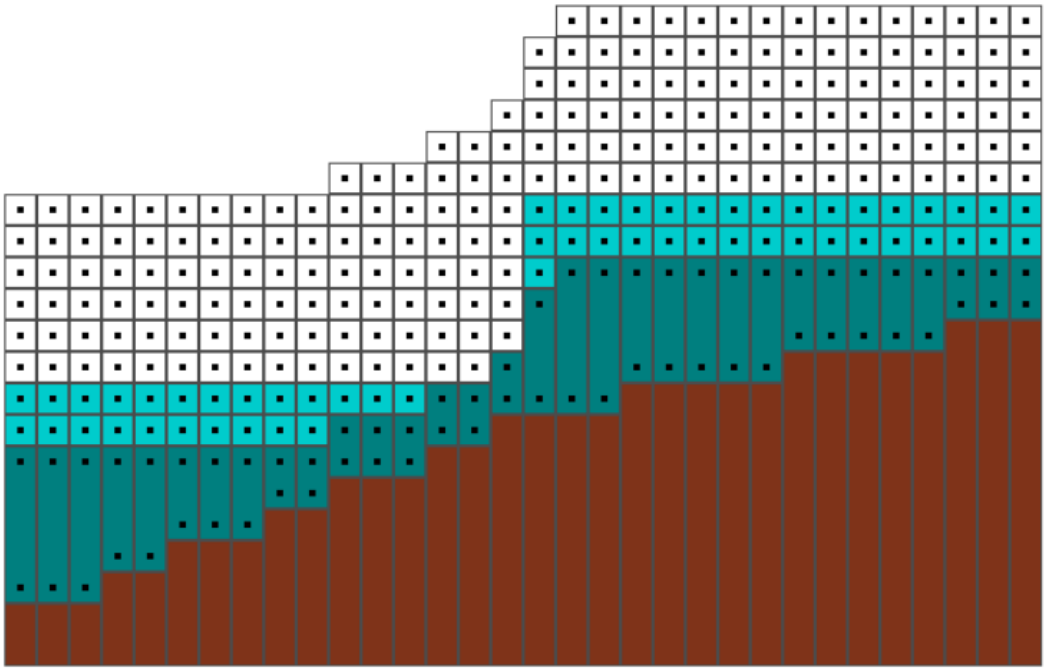

Figure 32: 2D cross section of the tall cell grid. Each column stores the terrain height, one tall cell and a constant number of regular cubic cells. Physical quantities are stored at the center of regular cells and at the top and the bottom of tall cells.

A quantity stored in the compressed array at position i, j, k is denoted with $q_{i,j,k}$ without parentheses, and a quantity at the uncompressed world location $(x\Delta x,\ y\Delta x,\ z\Delta x)$ as $q_{(x,y,z)}$ with parentheses. Depending on the y -coordinate, there are four cases for evaluating $q_{(x,y,z)}$ based on the values stored in the compressed array.

1. If $y \leq H_{x,z}$ the value of $q_{(x,y,z)}$ is the value below the terrain.

2. If $H_{x,z} < y \leq H_{x,z} + h_{x,z}$ the requested quantity lies within the tall cell. In this case, we linearly interpolate from the top and the bottom sub-cells of the tall cell:
$$q(x, y, z) = \frac{y - H_{x,z}}{h_{x,z}} q_{x,2,z} + (1 - \frac{y - H_{x,z}}{h_{x,z}}) q_{x,1,z}$$

3. If $H_{x,z} + h_{x,z} < y < H_{x,z} + h_{x,z} + B_y$, the quantity must be looked up from the regular cells in the compressed array as : $q(x, y, z) = q_{x,(y - H_{x,z} - h_{x,z} - 2),z}$

4. Otherwise $q_{(x,y,z)}$ gets the value above air

This definition of $q_{(x,y,z)}$ hides the tall cell structure of the grid. Once implemented, the grid can be accessed as if it was a regular grid composed of cubical cells only, which simplifies what follows significantly. A quantity at an arbitrary point in space can be computed using tri-linear interpolation of the nearest $q_{(x,y,z)}$'s.

4.1.3 Remeshing

After advection, liquid cells are identified as those where $\varphi \leq 0$. At this point the new number of cells above the terrain that should be grouped into one tall cell for each column (i,k) must be calculated. There are a few desirable constraints that may conflict each other:

1. There must be at least G_L regular cells below the bottom most liquid surface to capture the 3D dynamics of the liquid.

2. There must be at least G_A regular cells above the top most liquid surface, to allow water to slosh into the air in the next time steps.

3. The heights of adjacent tall cells must not differ by more than D units to reduce the volume gain artifacts.

We first iterate through each pair (i,k) and compute the maximum and minimum y -coordinate of the top of the tall cell that satisfy constraints (1) and (2), respectively. Next the temporary variable $y_{i,k}^{tmp}$ is initialized to be the average of the two extrema. To reduce the differences in height of adjacent tall cells several smoothing passes on $y_{i,k}^{tmp}$ are run. During the smoothing $y_{i,k}$ is clamped so that it always satisfies conditions (1) and (2), giving preference to condition (2) by enforcing it after condition (1). Finally, we iterate through (i,k) again and enforce condition (3) in a Jacobi-type fashion using : $y_{i,k}^{tmp'} = min(y_{i,k}^{tmp}, \max_{|i'-i|+|k'-k|=1} y_{i,k}^{tmp} + D)$

Finally height is set: $h_{i,k}^{new} = y_{i,k}^{temp} - H_{i,k}$. The authors in their examples used $8 \leq G_L \leq 32$, $G_A = 8$, $3 \leq D \leq 6$ and between one and two Jacobi iterations.

The algorithm attempts to make compromise among the constraints but may not satisfy all of them. Once the new heights of the tall cells are known, all the physical quantities are transfered to the new grid. For regular cells, the values at the corresponding locations from the old grid are copied or interpolated linearly if the location was occupied by a tall cell in the

previous time step. For tall cells, a least square fit is done to obtain the values at the bottom and the top of the cell.

4.2 Our method: A more general hybrid grid structure

The authors of the above paper introduced the tall cell grid structure in order to speed up their GPU-based simulation. This addition, together with a specialized multi-grid solver for the Poisson equation and further grid modifications for the GPU implementation, lead to a real-time GPU-based liquid simulation.

We wanted to use this hybrid grid structure concept into the CPU-based solver we developed and research how it affects the overall performance. Furthermore, we generalized this tall cell structure to further speed up the simulation.

4.2.1 Details

We extended the tall cell grid method to support non-regular large cells on all sides of the simulation domain, instead of only using a layer of tall cells at the bottom of the simulation. Consequently, we define four layers of large cells in the two-dimensional simulation and six layers in the three-dimensional one. Figure 33 and Figure 34 show an example of this grid structure.

Instead of using a collocated grid, we use a MAC grid storing the velocity components at the center of the facets of the grid cells. The pressure along with the level-set values are stored at the center of the each grid cell. Regarding the large cells (both horizontal and vertical) we only store the simulation quantities for the first and last regular cells of each of these large cells. If we need the value of some quantity of cell in between these two, we interpolate between the two known values.

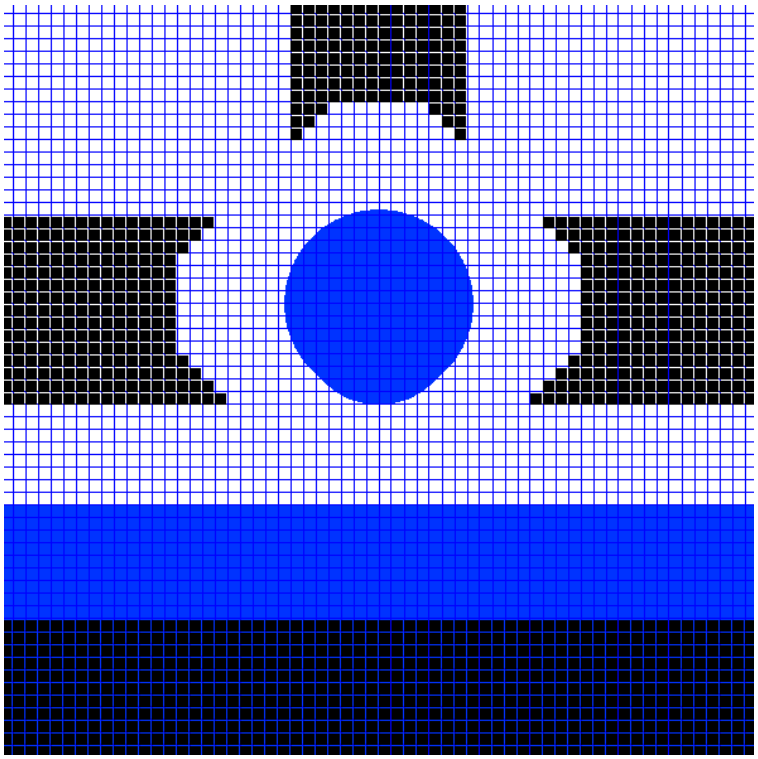

Figure 33: Screenshot of our generalized hybrid cell grid. Regular fluid and air cells are blue and white respectively. The non-regular cells (either horizontal or vertical) are black.

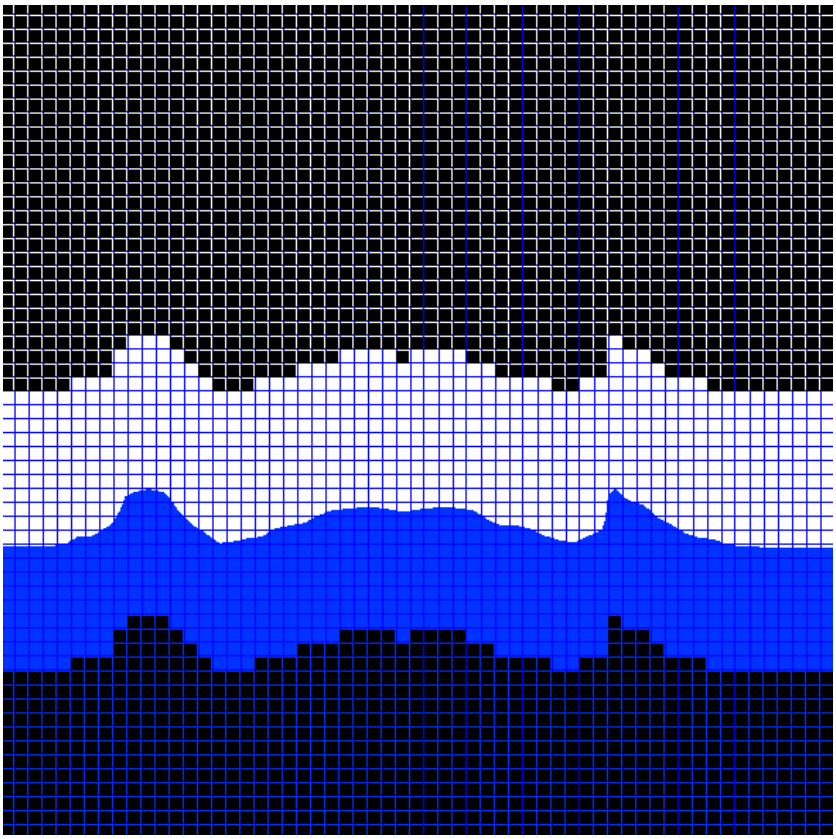

Figure 34: Screenshot of our generalized hybrid cell grid. Regular fluid and air cells are blue and white respectively. The non-regular cells (either horizontal or vertical) are black.

4.2.2 Timings

Size	Time (sec)	Time Hybrid grid (sec)	Increase (%)
40x40x40	0.0722879	0.0341782	52.72
60x60x60	1.05866	0.337175	68.15
64x64x64	1.46029	0.477248	67.32
80x80x80	2.94668	2.02317	31.34
96x96x96	6.19246	3.89172	37.15
128x128x128	19.6494	12.0576	38.64

Table 1: Time comparison between the simulation that uses the standard grid structure and the simulation that used the hybrid grid structure we developed.

In table 1, we see that the performance increase that the hybrid grid structure introduces ranges from 31% to 68% with an average increase of 49%. Depending on the height of the larger cells in both the vertical and horizontal layers of the hybrid grid, we can achieve the increase in speed, but at the same time we have to ensure that enough cells remain between the layers of larger cells and the fluid surface in order to preserve the details of the surface. In all the examples measured for this thesis, we enforced that between each large non regular cell and the interface must exist at least $\frac{N}{6}$ regular cells, where N is the grid resolution of each example.

4.3 Multi-Threaded Advection

As we wanted to speed up the simulation, we first had to measure the time complexity of the individual parts of the simulation loop.

Size	Fast March (%)	Advection (%)	External Forces (%)	Presure Projection (%)	Velocity update (%)
40x40x40	5	52	2	38	2
64x64x64	5	45	0.05	46	0.05
96x96x96	6	39	0.8	52	0.8
128x128x128	6	31	0.8	61	0.7

Table 2: Time complexity of the individual parts of the simulation loop (rendering time is not taken into account)

The most time consuming part of the simulation loop is the advection step and the pressure projection.

In order to speed up the advection step we noticed that it does not require any synchronization or exchange of information between neighboring cells of the grid. As a result of this, it was possible to make the advection step multi-threaded by dividing the grid in smaller sub-grids (Figure 35). Then, we assign the advection step to an individual thread for each one of these sub-grids.

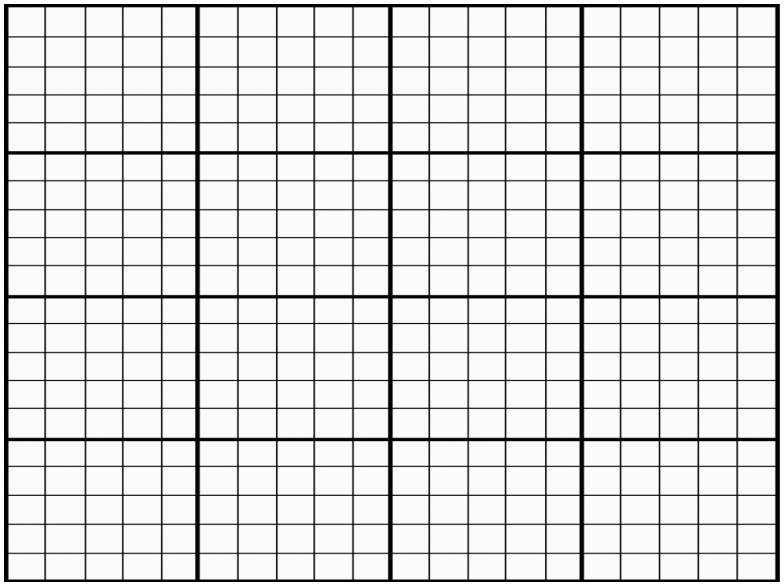

Figure 35: Example of a regular grid split into several sub-grids

4.3.1 Timings

Size	Time (sec)	Time Threads (sec)	Increase (%)
40x40x40	0.0722879	0.0319796	55.76
60x60x60	1.05866	0.361465	65.86
64x64x64	1.46029	0.625294	57.18
80x80x80	2.94668	2.05838	30.15
96x96x96	6.19246	4.20663	32.07
128x128x128	19.6494	11.7257	40.33

Table 3: Time comparison between base and multi-threaded simulation loop

4.4 Hybrid Grid Structure and Multi-Threaded Advection combined

We also combined the hybrid grid structure we developed with the multi-threaded approach we added to the advection step of the simulation in order to investigate if it could further speed up the simulation.

Size	Time (sec)	Time – Threads and Hybrid grid (sec)	Increase (%)
40x40x40	0.0722879	0.0234495	67.56
60x60x60	1.05866	0.291389	72.48
64x64x64	1.46029	0.456211	68.76
80x80x80	2.94668	1.78187	39.53
96x96x96	6.19246	3.50514	43.4
128x128x128	19.6494	10.5098	46.51

Table 4: Time comparison between base and multi-threaded hybrid grid simulation loop

4.5 Result comparison

Below you can find a comparison of a tank being filled with water between our results and another paper in terms of quality.

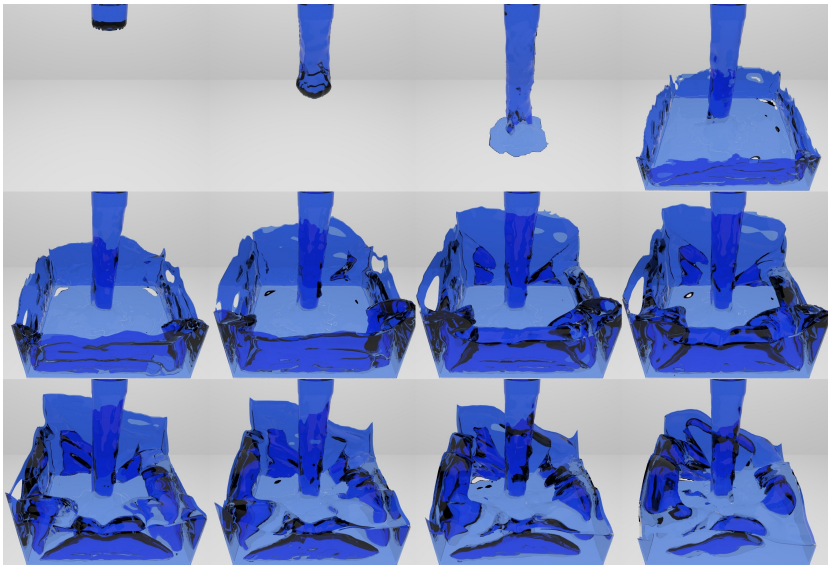

Figure 36: Tank filled with water. Results from my Master's Thesis.

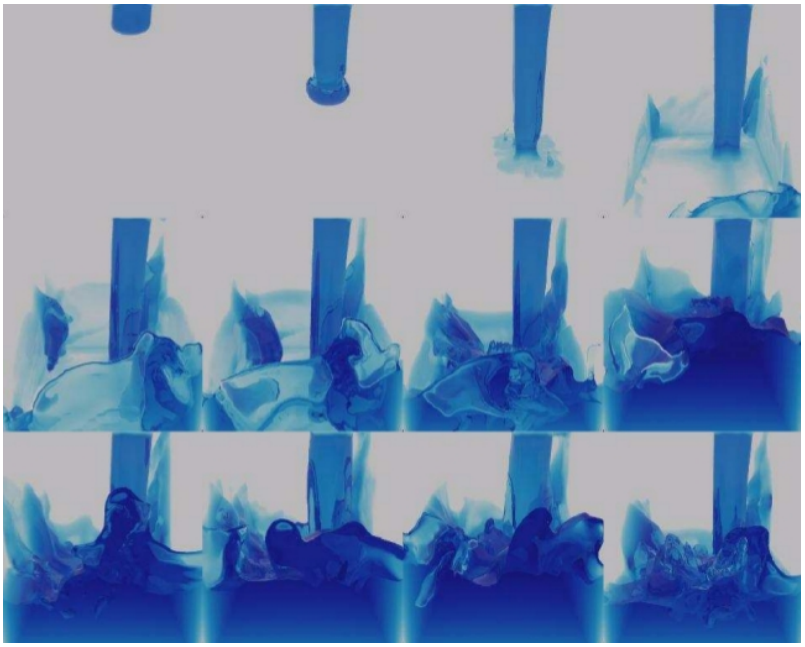

Figure 37: Tank filled with water. Result taken from [28]

Both results seem realistic, but in Figure 37 we observe that more small scale details appear at the edges of the water when compared to our results. We believe that is due to the sharper interpolation methods that the authors in [28] use. We use regular trilinear interpolation methods for interpolations of velocities and signed-distance values of the implicit surface representation, instead of a sharper cubic Hermite spline interpolation like Catmull-Rom.
It is also important to note, that the results on Figure 37 are rendered by a custom ray tracing implementation of the authors which is built especially for the purposes of this particular fluid and set-up. On the other hand, we used pbrt which is able render a large set of materials and scenes but does not have any special support for water or other kinds of fluids other than a generic glass material.

4.5.1 Surface tension

A significant amount of time of this Master's Thesis was spent investigating how surface tension can be included in the simulation. We did not get the results we were hoping at the beginning, but since surface tension is responsible for many small scale phenomenas and fine details in the real world, we think that we should show the work we did towards this direction.

Derivation

Following the derivation of the pressure at the interface between two neighboring fluids previously found in the "Surface Tension" chapter, we still have to find a way to add this more accurate boundary pressure calculation into the simulation.

We decided to do that by incorporating the surface tension force into the ghost fluid method that described earlier. In order to do that we take the ghost pressures in the air so that we linearly interpolate to $\gamma\kappa$ at the point where $\varphi = 0$, rather than interpolating to zero. The mean curvature κ can easily be estimated at that point by using the standard central difference for the Laplacian, on trilinearly interpolated values of φ .

Like in the "Surface Tension" chapter, we have to derive the two equations that need to be changed, namely the ghost pressure value of the boundary air cells and the velocity update equation.

$$\begin{aligned} (1-\theta)\, p_{i,j} + \theta\, p^{G}_{i+1,j} &= \gamma\kappa \;\Rightarrow \\ p^{G}_{i+1,j} &= -\frac{1-\theta}{\theta}\, p_{i,j} + \frac{1}{\theta}\gamma\kappa \;\Rightarrow \\ p^{G}_{i+1,j} &= \frac{\varphi_{i+1,j}}{\varphi_{i,j}}\, p_{i,j} + \frac{\varphi_{i,j} - \varphi_{i+1,j}}{\varphi_{i,j}}\gamma\kappa \end{aligned} \tag{40}$$

By plugging equation(40) into equation (31):

$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho \Delta x}(p_{i+1,j}^{G} - p_{i,j}) \Rightarrow$$
$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho \Delta x}\left[\frac{\varphi_{i+1,j}}{\varphi_{i,j}} p_{i,j} + \frac{\varphi_{i,j} - \varphi_{i+1,j}}{\varphi_{i,j}} \gamma\kappa - p_{i,j}\right] \Rightarrow \quad (41)$$
$$u_{i,j}^{n} = u_{i,j} - \frac{\Delta t}{\rho \Delta x}\left[\frac{\varphi_{i+1,j} - \varphi_{i,j}}{\varphi_{i,j}} p_{i,j} + \frac{\varphi_{i,j} - \varphi_{i+1,j}}{\varphi_{i,j}} \gamma\kappa\right]$$

Finally, the mean curvature $k = \nabla \cdot \nabla \varphi$ at a surface point x can be be estimated by using the laplacian:

In two dimensions:

$$x_{i,j}^{curv} = \frac{1}{\Delta x^2}(\varphi_{i+1,j} + \varphi_{i-1,j} + \varphi_{i,j+1} + \varphi_{i,j-1} - 4\,\varphi_{i,j}) \quad (42)$$

In three dimensions:

$$x_{i,j,k}^{curv} = \frac{1}{\Delta x^2}(\varphi_{i+1,j,k} + \varphi_{i-1,j,k} + \varphi_{i,j+1,k} + \varphi_{i,j-1,k} + \varphi_{i,j,k+1} + \varphi_{i,j,k-1} - 6\,\varphi_{i,j,k}) \quad (43)$$

Unfortunately, after these changes were added into the pressure project and the velocity update functions respectively, no significant visual change could be observed to the final simulation. We spent a fair amount of time trying to figure out why these changes did not work.

Firstly, we verified the correctness of our derivations and then tried several values for the surface tension γ constant. This constant depends on the two neighboring fluids and in the case of water and air at normal conditions is approximately 0.073 J/m^2 . Neither this nor larger values introduced any visual differences to the simulation.

Below you can see some real world scenes of water formations that happen due to surface tension. and thus can be estimated using a computer only by simulators that support it.

Figure 38: Breakup of a moving sheet of water bouncing off of a solid object

Figure 39: A metal paper clip floats on water. Several can usually be carefully added without overflow of water.

Below you can find a comparison between droplet simulation and a real droplet dropping into a pool of water.

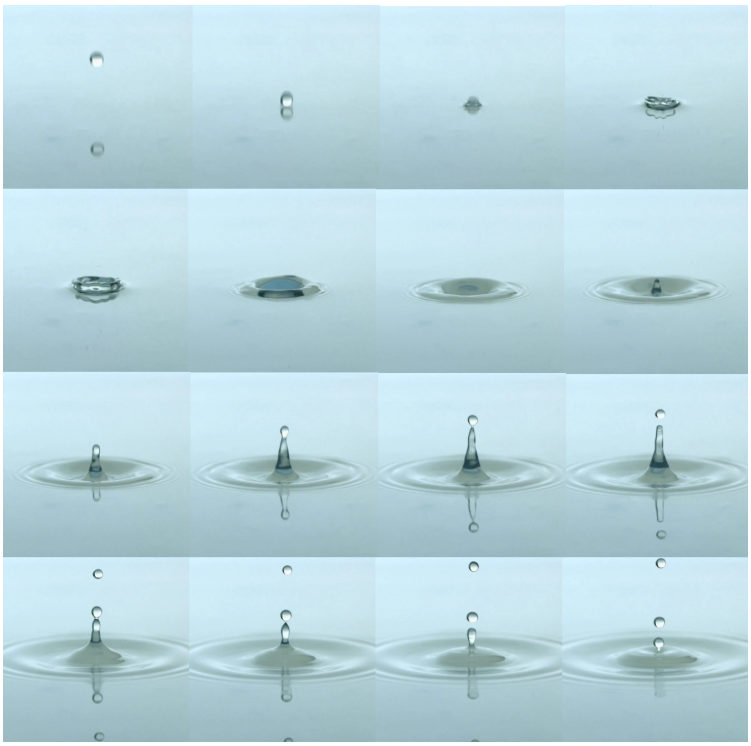

Figure 40: Water droplet in a pool of water

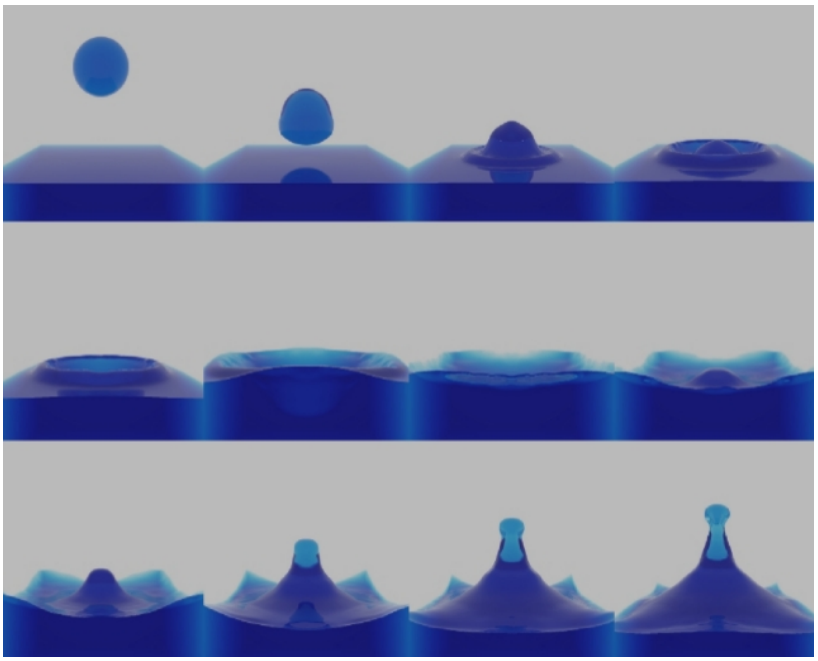

Figure 41: Simulation of a water droplet using the Ghost fluid method with surface tension. Image taken from [28]

In the two above figures we can see that while the simulation seems realistic enough to the human observer, when compared to a real world example, it lacks important details. This mainly happens at the thin parts of the fluid that surface tension tends to separate it to smaller drops.

Surface tension can drastically improve the visual fidelity of a simulation with the small fine details it introduces and thus is an important part of a fluid simulation software. However, it is a complex subject of ongoing research in both the physics and graphics fields.

Conclusions and Future work

5.1 Conclusions

In this Master's Thesis we developed a three dimensional fluid simulator, which uses a staggered grid and follows the Eulerian viewpoint together with a semi-Lagrangian advection technique. We focused our contributions into speeding up the simulation by adding the following methods to the base simulator:

1. hybrid-grid structure using a combination of layers of large cells defined on all sides of the simulation domain

2. multi-threaded advection step

3. the two above methods combined

The above additions lead to performance increase that ranges between 30% and 70% with an average increase of around 51% (an average increase of around 49% for method 1, an average increase of around 47% for method 2 and an average increase of around 56% for method 3).

Performance-wise we think that making this type of Eulerian simulation run in most cases twice as fast, achieves the goals that we had set at the beginning of the project.
As far as the quality of the simulation is concerned, given the complexity of the fluid simulation field and the amount of research that is constantly undergoing, we think that our results are visually pleasing and believable to the human observer.

5.2 Future Work

Due to the complex nature of Computational Fluid Dynamics, there are many different steps of the simulation that can be improved in more than one aspects.

5.2.1 Simulation quality

Advection

The unconditionally stable semi-Lagrangian advection can be replaced by a MacCormack type scheme in order to reduce the unwanted numerical smoothing which makes water look viscous or causes smoke to lose detail. We already started developing a MacCormack type scheme but, as we wanted to focus on performance and time was limited, we decided to stop its implementation and consider it as a possible future addition.
Furthermore, modifications can be made to the standard semi-lagrangian scheme so that it fully conserves momentum which can lead to more stable simulations when using coarser grids and larger time steps.

Surface tracking

Several methods can be combined to the level set method to improve the surface representation without increasing the grid resolution.

- The triangle mesh of the surface can be generated and, then, updated whenever the level set is updated. Similarly to the particle level set method we then have extra information about the surface, which in combination with the signed-distance values of the grid can lead to a more precise fluid surface.

- A volume of fluid (VOF) method can be used to provide more information about the fluid interface. In VOF methods, the free surface is maintained by storing for each cell the amount of fluid it contains. This amount ranges between zero if the cell is empty and one if the cell is full of fluid.

Surface tension

We can finish our implementation of surface tension in order to calculate more accurately the pressure at the interface between neighboring fluids and thus get finer details at the surface of the fluid we are simulating.

5.2.2 Simulation speed

Multigrid solver

A variety of more dynamic grid structures can be used to improve the speed of the simulation. For example, the hybrid structure we presented earlier, speeds up the overall simulation but there is a certain limit in the increase it can introduce. This limit is due to the MICCG(0) Poison solver we are using which treats both the regular and the larger horizontal and vertical cells equally. In the presence of this hybrid grid structures, the resulting linear system is non-symmetric and the Conjugate Gradients method cannot be used. On the other hand, even though non-symmetric, we have a constant number of coefficients that need to be stored per cell. In order to solve this system more efficiently we can use a multigrid. Solver. By using a multigrid solver, we can both efficiently use hybrid grid structures and accelerate further the simulation as the pressure projection tends to be the more time consuming part of the whole simulation loop.

Data parallel architecture

In order to further improve the performance, we could develop the whole fluid simulation on a parallel architecture such as the GPU. In a GPU implementation, cell attributes (velocity, pressure, and so on) are stored in several 3D textures. At each simulation step, we update these values by running computational kernels over the grid. A kernel is implemented as a pixel shader that executes on every cell in the grid and writes the results to an output texture. However, because GPUs are designed to render into 2D buffers, we must run kernels once for each slice of a 3D volume.

Appendix

Divergence Theorem

The Fundamental Theorem of Calculus (that the integral of a derivative is the original function evaluated at the limits) can be generalized to multiple dimensions in a variety of ways. The most common generalization is the divergence theorem discovered by Gauss:

$$\iiint_{\Omega} \nabla \cdot \vec{u} \;=\; \iint_{\partial\Omega} \vec{u} \cdot \hat{n}$$

That is, the volume integral of the divergence of a vector field $\vec{u}$ is the boundary integral of $\vec{u}$ dotted with the unit outward normal $\hat{n}$. This actually is true in any dimension (replacing volume with area or length or hypervolume as appropriate). This provides our intuition of the divergence measuring how fast a velocity field is expanding or compressing: the boundary integral above measures the net speed of fluid entering or exiting the volume.

References

[1] T. D. Aslam. "A Partial Differential Equation Approach to Multidimensional Extrapolation." J. Comp. Phys. 193 (2004), 349–355.

[2] B. Adams, M. Pauly, R. Keiser, L. J. Guibas. 2007. Adaptively sampled particle fluids. In Proc. SIGGRAPH, 48.

[3] *A. W. Bargteil, T. G. Goktenin, J. F. O'Brien, and J. A. Strain, 2005. A semi-lagrangian contouring method for fluid simulation. ACM Transactions on Graphics.*

[4] C. Batty, F. Bertails, and R. Bridson 2007. A fast variational framework for accurate solid-fluid coupling. In Proc. SIGGRAPH, 100.

[5] J. U. Brackbill, and H. M. Ruppel. "FLIP: A Method for Adaptively Zoned, Particle-in-Cell Calculuations of Fluid Flows in Two Dimensions." J. Comp. Phys. 65 (1986), 314–343.

[6] R. Bridson, 2008. "Fluid Simulation for Computer Graphics." A K Peters, Ltd.

[7] T. Brochu and Bridson, R. 2009. Robust topological operations for dynamic explicit surfaces. SIAM Journal on Scientific Computing 31, 4, 2472–2493.

[8] M. Carlson, P. J. Mucha, and G. Turk. 2004. "Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid." ACM Trans. Graph. (Proc. SIGGRAPH) 23:3 (2004), 377–384.

[9] N. Chentanez, T.G. Goktenin, B. E. Feldmanm and J. F. O'Brien. 2006. Simultaneous coupling of fluids and deformable bodies. In Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 83–89.

[10] N. Chentanez, M. Müller-Fischer. 2010. Real-time simulation of large bodies of water with small scale details. In Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation.

[11] N. Chentanez, M. Müller, Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid, ACM Transactions on Graphics (SIGGRAPH 2011), 30(4), pp 82:1-82:10

[12] R. Courant, E. Issacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. Comm. Pure and Applied Math, 5:243–255, 1952.

[13] K. Crane, I. Llamas, and S. Tariq. 2007. Real-time simulation and rendering of 3d fluids. In GPU Gems 3, H. Nguyen, Ed. Addison Wesley Professional, August, ch. 30

[14] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. "A Hybrid Particle Level Set Method for Improved Interface Capturing." J. Comp. Phys. 183 (2002), 83–116.

[15] D. Enright, S. Marschner, and R. Fedkiw. "Animation and Rendering of Complex Water Surfaces." ACM Trans. Graph. (Proc. SIGGRAPH) 21:3 (2002), 736–744.

[16] D. Enright, D. Nguyen, F. Gibou, and R. Fedkiw, 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In In Proc. 4th ASME-

JSME Joint Fluids Eng. Conf., number FEDSM200345144. ASME, 2003–45144.

[17] [6] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi- Lagrangian particle level set method. Computers and Structures, 83:479–490, 2005.

[18] R. Fedkiw, J. Stam, and H. W. Jensen. "Visual Simulation of Smoke." In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 15–22. New York: ACM, 2001.

[19] [Feldman et al. 05] B. E. Feldman, J. F. O'Brien, B. M. Klingner, and T. G. Goktekin. "Fluids in Deforming Meshes." In Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim., pp. 255–259. Aire-la-Ville, Switzerland: Eurographics Association, 2005.

[20] N. Foster and R. Fedkiw. "Practical Animation of Liquids." In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 23–30. New York: ACM, 2001.

[21] N. Foster and D. Metaxas. "Realistic Animation of Liquids." Graph. Models and Image Processing 58 (1996), 471–483.

[22] N. Foster and D. Metaxas. "Modeling the Motion of a Hot, Turbulent Gas." In SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 181–188. Reading, MA: Addison Wesley, 1997.

[23] F. Gibou, R. Fedkiw, L.-T. Cheng, and M. Kang. "A Second- Order-Accurate Symmetric Discretization of the Poisson Equation on Irregular Domains." J. Comp. Phys. 176 (2002), 205–227.

[24] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. 2005. Coupling water and smoke to thin deformable and rigid shells. In Proc. SIGGRAPH, 973–981 .

[25] F. Harlow and J. Welch. "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface." Phys. Fluids 8 (1965), 2182–2189.

[26] F. H. Harlow. "The Particle-in-Cell Method for Numerical Solution of Problems in Fluid Dynamics." In Experimental Arithmetic, High- Speed Computations and Mathematics, pp. 269 269. Providence, RI: American Math. Society, 1963.

[27] N. Holmberg, and B. C. Wunsche. 2004. Efficient modeling and rendering of turbulent water over natural terrain. In Proc. GRAPHITE, 15–22.

[28] M. Kang, R. Fedkiw, and X.-D. Liu. "A Boundary Condition Capturing Method for Multiphase Incompressible Flow." J. Sci. Comput. 15 (2000), 323–360.

[29] B. M. Klinger, B. E. Feldman, N. Chentanez, and J. F. O'Brien. 2006. Fluid animation with dynamic meshes. In Proc. SIGGRAPH, 820–825.

[30] B. Long, and E. Reinhard. 2009. Real-time fluid simulation using discrete sine/cosine transforms. In Proc. ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, 99–106.

[31] [Losasso et al. 04] F. Losasso, F. Gibou, and R. Fedkiw. "Simulating Water and Smoke with an

Octree Data Structure.” ACM Trans. Graph. (Proc. SIGGRAPH) 23 (2004), 457–462.

[32] M. Lentine, W. Zeng, and R. Fedkiw. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. In Proc. SIGGRAPH, 114:1–114:9.

[33] A. McAdams, E. Sifakis, and J. Teran 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation.

[34] [19] C. Min and F. Gibou. A second order accurate projection method for the incompressible Navier-Stokes equation on non-graded adaptive grids. J. Comput. Phys., 219:912–929, 2006.

[35] J. Molemaker, J. M. Cohen, S. Patel, and J. Noh. 2008. Low viscosity flow simulations for animation. In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 9– 18 .

[36] M. Muller, D. Charypar, and M. Gross. 2003. “Particle-Based Fluid Simulation for Interactive Applications”. Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation, pp. 154-159.

[37] M. Muller, 2009. Fast and robust tracking of fluid surfaces. In Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation.

[38] S. Osher, and Sethian. “Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton--Jacobi Formulations”. J.A. Journal of Computational Physics, 79, pp. 12-49, 1988.

[39] S. Osher and R. Fedkiw. Level Set Methods and Dynamic Implicit Surfaces. New York: Springer-Verlag, 2002.

[40] S. Premoze, T. Tasdizen, J. Bigler, A. E. Lefohn, and R. T. Whitaker. 2003. Particle-based simulation of fluids. Comput. Graph. Forum 22, 3, 401–410.

[41] C. S. Peskin. “The Immersed Boundary Method.” Acta Numerica 11 (2002), 479–517.

[42] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw. 2008. Two-way coupling of fluids to rigid and deformable solids and shells. ACM Trans. Graph. 27 (August), 46:1–46:9.

[43] J. Sethian. “A Fast Marching Level Set Method for Monotonically Advancing Fronts.” Proc. Natl. Acad. Sci. 93 (1996), 1591–1595.

[44] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):910–914, 2005.

[45] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. 2008. An unconditionally stable MacCormack method. J. Sci. Comput. 35, 2-3, 350–371.

[46] B. Solenthaler, and R. Pajarola. 2009. Predictive- corrective incompressible sph. In Proc. SIGGRAPH, 1–6.

[47] J. Stam. “Stable Fluids.” In SIGGRAPH 99 Conference Proceedings, Annual Conference Series, pp. 121–128. New York: ACM, 1999.

[48] A. Staniforth and J. Cote. Semi-Lagrangian integration schemes for atmospheric models: A review. Monthly Weather Review, 119:2206– 2223, 1991.

[49] J. Steinhoff and D. Underhill. Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. Phys. of Fluids, 6(8):2738–2744, 1994.

[50] T. Takahashi, H. Ueki, A. Kunimatsu, and H. Fujii. 2002. The simulation of fluid-rigid body interaction. In ACM SIGGRAPH conference abstracts and applications, 266–266.

[51] N. Thurey, and U. Rude. 2004. Free Surface Lattice- Boltzmann fluid simulations with and without level sets. Proc. of Vision, Modelling, and Visualization VMV, 199–207.

[52] N. Thurey, M. Müller-Fischer, S. Schirm, and M. Gross. 2007. Real-time breaking waves for shallow water simulations. In Proc. Pacific Conf. on CG and App., 39–46.

[53] N. Thurey, and U. Rude. 2009. Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids. Computing and Visualization in Science 12 (5).

[54] Y.-H. R. Tsai. "Rapid and Accurate Computation of the Distance Function Using Grids." J. Comput. Phys. 178:1 (2002), 175–195.

[55] J. Tsitsiklis. "Efficient Algorithms for Globally Optimal Trajectories." IEEE Trans. on Automatic Control 40 (1995), 1528–1538.

[56] O. Šťava, B. Benes, M. Brisbin, and J. Krivanek. 2008. Interactive terrain modeling using hydraulic erosion. In Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 201–210.

[57] C. Wojtan , C., N. Thurey, M. Gross, and G. Turk. 2010. Physics-inspired topology changes for thin fluid features. In Proc. SIGGRAPH, no. 4, 1–8.

[58] H. Zhao. "A Fast Sweeping Method for Eikonal Equations." Math.Comp. 74 (2005), 603–627.

[59] Y. Zhu and R. Bridson. "Animating Sand as a Fluid." ACM Trans. Graph. (Proc. SIGGRAPH) 24:3 (2005), 965–972.