[1 Introduction 2](#_Toc193778436)

[1.1 Motivation 2](#_Toc193778437)

[1.2 How to simulate fluids 2](#_Toc193778438)

[1.3 Related work 3](#_Toc193778439)

[1.4 Used techniques 4](#_Toc193778440)

[2 Fluid simulation 5](#_Toc193778441)

[2.1 Chapter overview 5](#_Toc193778442)

[2.2 Basics of fluid mechanics 5](#_Toc193778443)

[2.3 Basics of smoothed particle hydrodynamics 9](#_Toc193778444)

[2.4 Particle based, mathematical model of fluid motion 10](#_Toc193778445)

[2.5 Smoothing kernels 13](#_Toc193778446)

[2.6 Basic simulation algorithm 14](#_Toc193778447)

[2.7 Implementation 16](#_Toc193778448)

[2.8 Environment and user interaction 18](#_Toc193778449)

[2.9 Multithreading optimization 19](#_Toc193778450)

[2.10 Results 19](#_Toc193778451)

[3 Visualization 20](#_Toc193778452)

[3.1 Optical characteristics of water 20](#_Toc193778453)

[4 Conclusion 21](#_Toc193778454)

[4.1 Summary 21](#_Toc193778455)

[4.2 Improvements and alternatives 21](#_Toc193778456)

[Appendix 22](#_Toc193778457)

[References 22](#_Toc193778458)

[Glossary 23](#_Toc193778459)

[Derivation of the gradient and Laplacian of the smoothing kernels 23](#_Toc193778460)

# Introduction

## Motivation

Fluids like liquids and gases are ubiquitous parts of the environment we live in. For instance we all know how it looks like when milk gets filled into a drinking glass. In realtime computer graphics, where we traditionally try to reproduce parts of our world as visually realistic as possible, it’s unfortunately hard to simulate such phenomena. Computational fluid dynamics is a relatively old and well known research topic, but most applications (like i.e. in aerodynamics research) aim at results that are as accurate as possible. Therefore the simulations are mostly calculated offline and realtime visualization is mostly used only to render precomputed data sets, if at all.

|  |  |
| --- | --- |
| 2.png  3.png  Figure : Example for offline simulation Source:[APK07] | 1.png  Figure : Example for realtime simulation Source: [MCG03] |

Realtime applications that do allow the user to interact with authentically (but not necessarily accurately) simulated and rendered fluids (like i.e. water) are today rare. For all types of virtual realities, like surgical training environments or computer games, there’s always demand to cover more aspects of our world and so realtime simulation and rendering of fluids is an interesting field of study. In 2003 Müller, Charypar and Gross sparked additional interest in realtime fluid simulation, with a paper that proposed a relatively simple, particle based fluid-model, that fits well for realtime applications [MCG03]. Since then many different aspects of realtime particle based fluid simulation where covered in a couple of papers from authors around the world. This thesis gives an overview on the topic, as it discusses my implementation of a particle based fluid simulation and a suitable water renderer.

## How to simulate fluids

In the nineteenth century Claude Navier and George Stokes created the fundamentals of modern fluid dynamics as they formulated the well known Navier-Stokes equations. With these equations, which describe the conservation of momentum, together with two additional equations for mass and energy conservation, it’s possible to simulate the fluid flow. As the formulas tend to get very complicated for less common fluids, they are mostly written for Newtonian fluids which include a variety of common liquids and gases (water, air...).

Simulations apply numerically methods to solve the (in most cases) resulting nonlinear partial differential equations. One common way to do this is to treat the fluid as a continuum, discretize the spatial domain into a grid and use finite differences or the finite volume method. In the literature grid based fluid models are called Eulerian models. For the use within virtual environments grid based methods, as a matter of principle, have the drawback of a bounded simulation space.

Particle based methods (in literature: Lagrangian model, from Lagrangian mechanics) in contrast represent the fluid as a discrete set of particles and simulate the fluid flow through solving the particle dynamics. For realtime applications this brings some advantages over grid based methods:

* simpler calculation (mass conservation can be omitted, convective term can be omitted, cp. [MCG03])
* no numerical diffusions in the convection terms (diffusion directions are not influenced by the grid layout)
* surface reconstruction is likely to be easier
* fluid can spread freely in space (no boundary through the grid)

For those reasons (especially the last) this thesis focuses on a Lagrangian method based on smoothed particle hydrodynamics (SPH) [Mon05] which became very popular for this kind of applications. The idea behind SPH is that every particle distributes the fluid properties in its neighborhood using radial kernel functions. To evaluate some fluid property at a given point one must simply sum up the properties of the neighboring particles, weighted with the appropriate smoothing function.

## Related work

The first investigations in smoothed particle hydrodynamics where made in 1977 by Gingold and Monaghan (who coined the term) [GM77] and independently by Lucy [Luc77]. Its first usages took place mainly in the astronomy sector to simulate large scale gas dynamics, but later it also has been applied to incompressible flow problems like beach wave simulation, sloshing tanks and bow waves of ships.

While in realtime computer graphics first the Eulerian approach was favored, Müller, Charypar and Gross [MCG03] where one of the first who showed, that a SPH based Lagrange method also suits very well to interactive applications. Later many papers used SPH to simulate fluids (especially liquids) in realtime and brought adaptations and improvements both for the simulation as well as for the rendering of liquids.

Papers on realtime SPH simulation:

* [KC05] proposes to avoid the particle neighborhood problem by sampling the fluid properties from grids witch sum up the weighted properties from all particles
* [KW06] compares the performance of an octree based (linear time for neighbor search, but large costs for the update of the structure) versus a “staggered grid” based solution to the neighbor problem
* in [MST04] Müller et al. show how particle based fluids can interact with deformable solids
* [AIY04] sketches how to use a CPU generated neighbor map so that the property summation for each particle can be handled on the GPU which reaches twice the performance of their CPU only simulation
* [Hei07] uses the Ageia PhysX engine (one of its developers is Matthias Müller) for a SPH based simulation of smoke

Papers with relevance for realtime liquid rendering:

* [MCG03] suggests direct point splatting of the particles or marching cubes rendering [LC87] of the isosurface (which implies that an isovolume must be created for each frame)
* [KW03] presents a GPU executed isovolume raycaster; in combination with a efficient method for building the isovolume on the GPU this way the iso surface could be visualized
* [CHJ03] introduces iso-splatting, a point based isosurface visualization technique; same as with [KW03] applies here
* [Ura06] demonstrates a GPU version of the marching tetrahedra algorithm (variation of marching cubes); same as with [KW03]
* [KW06] uses a 2.5D “carped visualization” for the special case of rivers and lakes

## Used techniques

The goal with this thesis was to provide a realtime application that simulates a water-like liquid in a form that is “believable” in terms of movement behavior and optical appearance. The SPH simulation therefore focuses not on physical accuracy. It’s a straightforward implementation of the lightweight SPH model presented in [MCG03], optimized to run on actual multi-core consumer CPUs. To speed up the neighbor search it stores the particles according to their position in a dynamic grid, with a cell size equivalent to the maximal radius of support. The particle interactions are evaluated directly on pairs of particles (simultaneous for both particles). Chapter 2 discusses the theoretical foundations and the implementation details of the simulation.

For visualization three techniques are provided: The first directly renders the particles as point sprites, which is mainly useful for debug and tuning of the fluid behavior. The second, which is nearly entirely CPU-based, uses the marching cubes algorithm to construct a triangle mesh representing the isosurface. This technique was implemented to experiment with efficient isovolume construction methods and to test how well a marching cubes / triangle based approach fits for the purpose of liquid visualization. The last and most sophisticated technique uses the GPU to construct an isovolume within a 3D texture and renders the isosurface directly with a raycasting shader. The raycasting enables the visualization of effects like multiple refractions and reflections, which are characteristic for the optical appearance of liquids. Chapter 3 explains each visualization technique in detail.

|  |  |  |
| --- | --- | --- |
| 1.jpg  Figure : Sprite visualization | 2.jpg  Figure : Marching cubes visualization | 3.jpg  Figure : GPU raycasting visualization |

# Fluid simulation

## Chapter overview

## Basics of fluid mechanics

Fluid mechanics normally deals with macroscopic behavior at length and time scales where intermolecular effects are not observable. In this situation fluids can be treated as continuums where every property has a definite value at each point in space. Mathematically this can be expressed through functions that depend on position and time (i.e. vector or scalar fields). Properties are macroscopic observable quantities that characterize the state of the fluid. The most relevant properties for the movement of fluids are mass, density, pressure and velocity. The mass specifies “how much matter there is” and is relevant for the inertia of the fluid. The mass density measures the mass per volume and is defined as:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: very small length, but significant greater than the molecule spacing; : volume

Pressure is a scalar quantity that’s defined as the force acting in normal direction on a surface (normal stress): . Differences in the pressure field of a fluid (= force differences) result in a flow from areas of high to areas of low pressure, while in regions with constant pressure those forces are balanced.

The velocity is a measure for how fast and in which direction the fluid passes a fixed point in space. It’s perhaps the most important property of the fluid flow. The velocity field effects most other properties either directly (i.e. dynamic pressure) or indirectly (i.e. because of advection). In viscous fluids (all real fluids are viscous to some amount) it’s also relevant for the viscosity forces which are together with pressure forces the most relevant fluid forces.

Viscosity compensates the differences in flow velocity over time (comparable to friction). In case of a fluid with a “constant” viscosity (later more on that) it’s a measure for how much momentum is transferred between adjacent regions with different flow speeds and is thereby responsible for shear stress (tangential force on a surface). Viscosity as a constant is stated as dynamic viscosity (when the result is a force) or kinematic viscosity (when the result is acceleration).

Surface tension is the last cause of forces that we deal with. It’s a property of the surface of the fluid (the border to another immiscible fluid, a solid or vacuum), that is relevant for the size of the forces that try to minimize the area and curvature of the surface. A simple explanation for the cause of Surface tension is that the cohesive forces (attractive forces between molecules of the same type) between molecules on the surfaces are shared with less neighbor molecules than in the inner of the fluid, which results in a stronger attraction of the molecules on the surface. It is mentioned here for completeness although it’s not further discussed in the basics subchapter (we will deal with it later in 2.4).



Figure : Cause of surface tension

Now that we know the meaning of most magnitudes, let’s see how the motion of a fluid could be described mathematically. Let’s start with Newton’s second law:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Note: vectors are written in bold (), scalars in italics ().

It states that the acceleration of an object depends on its mass and the force that acts on it. This could also be interpreted as conservation of momentum: Without external forces () there’s no change of velocity () and the momentum stays constant.

In classical (Newtonian) dynamics Newton’s second law is usually interpreted from the Lagrangian point of view, meaning that a moving object is observed. With fluids this would mean that the observation area follows the fluid flow, so that always one and the same “amount of fluid” is being watched. Alternatively in the Eulerian point of view the area of observation is locally fixed, so that the fluid passes by and the watched amount of fluid may be a different one at each moment. The Eulerian observer therefore not only sees changes due to variances in the currently watched amount of fluid, but also changes due to the fact that the watched amount of fluid may be a different one every moment.



Figure : Lagrangian versus Eulerian point of view

In an Eulerian description (which is more common in classical fluid dynamics) the acceleration therefore must be a special time derivative of the velocity, which takes into account the movement of currents in fluids in both of it forms: Diffusion and advection (together: convection). It is called substantial derivative (synonyms: substantive d., convective d., material d.) and defined as follows:

|  |  |  |
| --- | --- | --- |
|  |  |  |

written in Cartesian coordinates in three dimensions; : del operator; : components of velocity; : components of position; : an arbitrary quantity (vector or scalar)

The partial derivative expresses the “local” changes in the currently observed amount of fluid (i.e. due to diffusion or external influences) while the term represents the changes due to advection (transport of properties together with the matter). By replacing the acceleration in with the substantive derivative of the velocity we get:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: gradient of the velocity (the Jacobian matrix)

(2.1) states that the mass of the fluid inside the observed control volume depends on its density, therefore we write:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Now we will focus on the forces acting on the fluid. It can be distinguished between internal forces produced by the fluid itself and external forces like gravity or electromagnetic forces:

|  |  |  |
| --- | --- | --- |
|  |  |  |

The most important external force is gravity which is in fact stated as gravitational acceleration. Synonym we will describe the external forces as force density field that directly specifies acceleration (remember that the mass depends on the density in our case):

|  |  |  |
| --- | --- | --- |
|  |  |  |

In order to provide a simple expression for the fluid forces, we assume that we deal with a Newtonian fluid that satisfies the incompressible flow condition. A viscid fluid is called Newtonian when the viscous stress is proportional to the velocity gradient (cp. [Pap99]). For Newtonian fluids the equation describes the relation between shear stress , dynamic viscosity constant and the velocity gradient perpendicular to the direction of share [BE02]. This means in common words that, in contrast to non-Newtonian fluids, the viscosity is a constant and does not change under different shear rates. The fluid flow is called incompressible when the divergence of the velocity field is zero (), meaning that there are no sources or sinks in the velocity field. As a counter example think of air that expands because it’s heating up. Note that also flows of compressible fluids (all real fluids are compressible to some extent) can satisfy the incompressible flow condition (i.e. regular air flow till ~ mach 0.3). If the fluid fulfills all this conditions, we can simply spilt fluid forces into forces due to pressure differences (normal stresses) and in viscosity forces due to velocity differences (shear stresses):

|  |  |  |
| --- | --- | --- |
|  |  |  |

The pressure forces depend only on the *differences* in pressure and let the fluid flow from areas of high to areas of low pressure. We model them with the negative gradient of the pressure field , which points from high to low pressure areas and has a magnitude proportional to the pressure difference:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Because of our assumption of an incompressible flow, the viscosity force becomes a relative simple term:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: dynamic viscosity; : the Laplacian operator, sometimes also written

For a mathematical derivation of the term above see i.e. chapter 5 in [Pap99] or [WND]. Here it should only be remarked, that the Laplacian is an operator that measures how far a quantity is from the average around it and therefore the force expressed by (2.10) smoothes the velocity differences over time. This is what viscosity is supposed to do. By combining the last two formulas we end up with the Navier-Stokes momentum equation for incompressible, Newtonian fluids often simply referred to as *the* Navier-Stokes equation:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Navier-Stokes equation

This equation is the basis of a bunch of fluid simulation models. In 2.4 we will combine it with the basic principles of smoothed particle hydrodynamics (2.3) to form the mathematical model of the fluid simulation presented in this thesis. The sense of its rather descriptive derivation in this subchapter was to make the equation plausible in each of its parts and as a whole. The derivation therefore was intentional not mathematically strict and left out some concepts that are relevant for other forms of the equation (like the stress tensor ). In the literature (i.e. [Pap99]) numerous mathematical strict derivations can be found if needed. This subchapter made clear that the Navier-Stokes equation is simply a formulation of Newton’s second law and a statement of momentum conservation for fluids.

## Basics of smoothed particle hydrodynamics

Smoothed particle hydrodynamics is a technique developed by Gingold and Monaghan [GM77] and independently by Lucy [Luc77] for the simulation of astrophysical gas-dynamics problems. As in other numerical solutions to fluid dynamic problems, the value of a physical quantity at a given position must be interpolated from a discrete set of points. SPH derives from the integral interpolation:

|  |  |  |
| --- | --- | --- |
|  |  |  |

is a radial symmetric smoothing function (also called kernel) with smoothing length (also called core radius). One could say that the interpolation uses the smoothing kernel to spread a quantity from a given position in its surroundings. In practice the kernel is even () and normalized () and tends to become the delta function for tending to zero (if would be the delta function, would reproduce exactly). This thesis follows the example of [MCG03] to treat as the radius of support, so all used smoothing functions will evaluate to zero for .

1.wmf

Figure : 1D example for a smoothing kernel

With SPH the interpolation points are small mass elements, which aren’t fixed in space (like the grid points in the Euler method) but move with the fluid. For each such fluid particle a position , velocity , mass and density is tracked. The value of a quantity at a given position can be interpolated from the particle values using the summation interpolant (derived from the integral form):

|  |  |  |
| --- | --- | --- |
|  |  |  |

The mass-density-coefficient appears because each particle represents a volume of . As an interesting example (2.13) applied to the density gives:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Which shows that with SPH the mass density is estimated by smoothing the mass of the particles.

In practice not all particles must participate in the summation. As the smoothing kernel only has a finite radius of support, all particles with a greater distance to the evaluated point can be omitted.

An advantage of SPH is that spatial derivatives (which appear in many fluid equations) can be estimated easily. When the smoothing kernel is differentiable the partial differentiation of (2.13) gives:

|  |  |  |
| --- | --- | --- |
|  |  |  |

The gradient therefore becomes:

|  |  |  |
| --- | --- | --- |
|  |  |  |

According to [MCG03] this could also be applied to the Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  |  |

There also exist some different SPH formulations for the gradient and Laplacian that will not be further discussed here. Chapter 2.2 in [CEL06] gives a good overview of other useful formulations. Monaghan also suggests alternatives to (2.17) in chapter 2.3 of [Mon05].

This rules cause some problems when they are used to derive fluid equations for particles. The derivate does not vanish when is constant and a number of physical laws like symmetry of forces and conservation of momentum are not guaranteed. When the time has come, we will therefore have to adjust the particle fluid equations slightly to ensure physical plausibility.

## Particle based, mathematical model of fluid motion

Now the core concepts of SPH from 2.3 will be applied to the Navier-Stokes equation introduced in 2.2 in a straightforward way, to form a mathematical model for particle based fluid simulation that’s simple enough to be suitable for realtime usage. This subchapter is entirely based on the [MCG03] paper, which introduced the lightweight simulation model used in this thesis.

In the model presented here each particle represents a small portion of the fluid. The particles carry the properties mass (which is constant and in this case the same for all particles), position and velocity. All other relevant quantities will be derived from that using SPH rules and some basic physical equations.

Grid based, Eulerian fluid models need an equation for the conservation of momentum like the Navier-Stokes equation (2.11) and at least one additional equation for conservation of mass (sometimes one for energy conservation too) like the continuity equation:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Continuity equation

The mass of each particle and the count of particles are constant, so mass conservation is guaranteed automatically. The momentum equation is all that’s needed to describe the movement of the fluid particles. A Lagrangian model doesn’t have to take advection of currents into account (see the comparison in 2.2) and thus the substantial derivative of the velocity field in the Navier-Stokes equation can be replaced with an ordinary time derivative of the particle velocity. What we get is a momentum equation for a single fluid particle:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: force acting on particle   
: density at position of particle   
: pressure gradient at position of particle   
: velocity Laplacian at position of particle

For the acceleration of a particle we get therefore:

|  |  |  |
| --- | --- | --- |
|  |  |  |

In (2.14) we have already seen how we could calculate the density at the particles position using the SPH rule (2.13):

|  |  |  |
| --- | --- | --- |
|  |  |  |

The external force density field rightmost in (2.19) directly specifies acceleration when the density factor vanishes after the division in (2.20). All what’s left for a complete description of the particle movement based on the Navier-Stokes equation are the terms for pressure and viscosity.

According to the SPH rules the pressure term would look like as follows:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Unfortunately the resulting force is not symmetric. This could be easily seen when only two particles interact. Because the gradient of a radial smoothing kernel is zero at its center, particle only uses the pressure of particle and vice versa. The pressure varies at different positions and thus the pressure forces would be different for the two particles. [MCG03] suggests to balance the forces by using the arithmetic mean pressure of the two interacting particles:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Till now the pressure at the particle positions was an unknown. Müller et al. proposes to use the ideal gas state equation to derive the pressure directly from the density:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: gas constant depending on temperature; : rest density

Appling the SPH rule to the viscosity term yields the following equation:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Which again results in asymmetric forces for two particles with different velocities. The viscosity forces depend only on velocity differences, not on absolute velocities, therefore the use of velocity differences is a legitimate way of balancing the force equation:

|  |  |  |
| --- | --- | --- |
|  |  |  |

This means the viscosity force in our model accelerates a particle to meet the relative speed of its environment.

Now we have a simple model for the forces acting on the particles that contains everything what’s expressed by the Navier-Stokes equation. But there’s an additional fluid force relevant for the scenario we’d like to describe, that’s not covered by the momentum equation. Fluids interacting with solid environments often produce small splashes and puddles with much free surface, where the surface tension force plays a noticeable role. As described in 2.2 the surface tension forces try to minimize the surface of the fluid body, to achieve an energetically favorable form. The bigger the curvature of the surface is, the bigger should be the surface tension forces that push the border particles towards the fluid body. In order to find the particles at the surface and calculate the surface tension forces, the color field method is used in [MCG03]. A color field is 1 at particle positions and 0 everywhere else. The smoothed color field has the form:

|  |  |  |
| --- | --- | --- |
|  |  |  |

The gradient of the color field gives us two kinds of information: Its length becomes huge only near the surface, which helps us identifying surface particles and its direction points towards the center of the fluid body, which is a good choice for the direction of the surface force. The surface curvature, which is a magnitude for the size of the force, could be expressed trough the Laplacian of the color field:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Using the color field gradient as force direction and “marker” for surface particles and the curvature as magnitude for the force size leads to the following equation for the surface tension force:

|  |  |  |
| --- | --- | --- |
|  |  |  |

: surface tension coefficient, depends on the materials that form the surface

is near to zero for inner particles, so the surface tension is only getting evaluated when it exceeds a certain threshold to avoid numeric problems. It should be mentioned that this surface tension model can be error-prone under some circumstances, so also other models proposed in the literature (i.e. in [BT07]) may be worth an evaluation.

## Smoothing kernels

The smoothing kernels used in the interpolations have great influence on speed, stability and physical plausibility of the simulation and should be chosen wisely. As every kernel is radial symmetric, it’s normally specified only as function of the length of : . It should be even (), normalized () and differentiable as often as needed. Despite of these requirements one is free to specify the kernel in every form that’s suitable for its task. In the literature there exist many different ways to specify them, from splines, over exponential functions up to Fourier transformation generated kernels. [Mon05] contains a good overview of the most common techniques.

In this thesis the kernels proposed in [MCG03] are used. The first is the Poly6 kernel:

|  |  |  |
| --- | --- | --- |
|  |  |  |

with:

it has the gradient:

|  |  |  |
| --- | --- | --- |
|  |  |  |

and the Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Note that in the appendix there’s the section “Derivation of the gradient and Laplacian of the smoothing kernels”

Its advantage is that appears only squared, so the computation-intense calculation of square roots can be avoided. The Poly6 kernel is used for everything except the calculation of pressure and viscosity forces. With pressure forces the problem is that the gradient goes to zero near the center. Therefore the repulsive pressure force between particles vanishes when they get too close to each other. This problem is avoided through the use of the Spiky kernel, which has a gradient that doesn’t vanish near the center:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Gradient:

|  |  |  |
| --- | --- | --- |
|  |  |  |

With viscosity the problem of the Poly6 kernel is that its Laplacian becomes negative really fast. A particle, that’s faster than its environment, could therefore be accelerated by the resulting viscosity forces, while it should actually get slowed down. In the viscosity calculation thus the “Viscosity” kernel is used, which’s Laplacian stays positive everywhere:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Gradient:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| ip.eps | is.eps | iv.eps |

Figure : Used smoothing kernels

(from left to right) along the x-axis for   
thick lines: kernel, thin l.: absolute value of gradient, dashed l. Laplacian

## Basic simulation algorithm

Subchapter 2.4 described how the fluid forces acting on the particles could be derived directly from the particle positions and velocities. This enables us to specify the basic algorithm for the fluid simulation:

Listing : Basic simulation algorithm

|  |
| --- |
| **while** simulation is running  h ← smoothing-length  init density of all particles  clear pressure-force of all particles  clear viscosity-force of all particles  clear color-field-gradient of all particles  clear color-field-laplacian of all particles  *// calculate densities*  **foreach** particle **in** fluid-particles  **foreach** neighbor **in** fluid-particles  r ← position of particle **–** position of neighbor  **if** length of r **≤** h  **add** mass **\*** W\_poly6(r, h) **to** density of particle *// compare*  **end-if**  **end-foreach**  **end-foreach**  *// calculate forces and color-field*  **foreach** particle **in** fluid-particles  **foreach** neighbor **in** fluid-particles  r ← position of particle **–** position of neighbor  **if** length of r **≤** h  density-p ← density of particle  density-n ← density of neighbor  pressure-p ← k **\*** (density-p **–** rest-density) *//*  pressure-n ← k **\*** (density-n **–** rest-density)  **add** mass **\*** (pressure-p **+** pressure-n) **/** (2 **\*** density-n) *//*  **\*** gradient-W-spiky(r, h) **to** pressure-force of particle  **add** eta **\*** mass **\*** (velocity of neighbor **–** velocity of particle) *//*  **/** density-n **\*** laplacian-W-viscosity(r, h)  **to** viscosity-force of particle  **add** mass **/** density-n **\*** gradient\_W\_poly6(r, h)  **to** color-field-gradient of particle  **add** mass **/** density-n **\*** laplacian\_W\_poly6(r, h)  **to** color-field-laplacian of particle  **end-if**  **end-foreach**  **end-foreach**  *// move particles*  **foreach** particle **in** fluid-particles  gradient-length ← length of color-field-gradient of particle  **if** gradient-length **≥** threshold *//*  surface-tension-force ← -sigma **\*** color-field-laplacian of particle  **\*** color-field-gradient of particle **/** gradient-length  **else**  surface-tension-force ← 0  **end-if**  total-force ← surface-tension-force **+** pressure-force of particle  **+** viscosity-force of particle  *//*  acceleration ← total-force **/** density of particle **\*** elapsed-time + gravity  **add** velocity of particle **+** acceleration **\*** elapsed-time  **to** velocity of particle  **add** velocity **\*** elapsed-time **to** position of particle  **end-foreach**  **end-while** |

The dependencies on the density and the forces lead to a tripartite evaluation scheme. First the density of each particle is evaluated by summation over the contributions of all particles in the neighborhood. In the second step every neighbor exerts forces on the particle and the color field is being built. At last the accumulated forces are used to approximate the movement of the particles in the current time step.

## Implementation

The fluid simulation as well as whole other CPU code of the program was implemented with C++, because today it’s the de facto standard in professional, realtime computer graphics on PCs. The pseudo code in the last chapter describes the real implementation of the simulation component relatively good. The update method, that’s called once for every simulation step, indeed linearly executes the following four tasks:

calculate density at every particle position

calculate pressure forces, viscosity forces and color field values for each particle

move the particles and clear the particle related fields

update the acceleration structures

As stated before the particles carry only the properties position and velocity (the mass is constant and the same for all particles). This is the only information that’s transferred from one simulation step to the next. All other per-particle data, like density and forces is stored in separate arrays. The particle data structure therefore consists of one three component vector for position, one for velocity and an integer index that locates derived particle properties in the respective arrays.

The first task, the density calculation, has to implement the summation interpolation. The summation is the most crucial point for the overall performance of the simulation. The naive summation over all particles in the simulation would result in a computation complexity that’s quadratic in the number of particles, which is impracticable for the amounts of particles we aim at. Therefore it’s necessary to implement the summation as a neighbor search that finds all those neighbor particles, which are near enough to influence a certain particle. Those are all particles with a distance lower than their smoothing length. In this simulation the smoothing length is treated constant and equal for all particles.

This allows the use of a location grid as efficient acceleration structure for the neighbor search. The grid consists of cubic cells with a side length equal to the smoothing length. Each cell contains a reference to a list of all particles, with a position that maps to the space partition associated with the cell, or a null pointer if no such particle exists. The particle positions change with every simulation step. Thus after each step the grid location and cell count dimensions must be updated to fit the space occupied by the particles and the particles must be sorted into the grid again. The neighbor search finds the neighbors of all particles in a particular grid cell. Because the side length equals the smoothing length, all neighboring particles must be contained in the current or one of the maximal 26 adjacent cells. This reduces the time complexity of the summation from to ( being the average number of particles per grid cell) at the cost of the time needed to rebuild the grid ().



Figure : Grid based neighbor search

A further performance gain is accomplished trough storing copies of the particles in the grid cells instead of references. This dramatically lowers the cache miss rate of the CPU, because all particles, that are accessed during the neighbor search for particles within one cell, lie close to each other in system memory.

The neighbor relation between the particles is symmetric () and also the interactions between the neighbors (density accumulation, force exertion) are mostly symmetric. This allows another optimization: Whenever a particle pair contained in the neighbor relation is found, all necessary calculations are performed in both directions, so that every pair must be evaluated only once. The algorithm visits cell after cell. First it checks each particle against all which follow in the same cell. Then it checks all the pairs between the current cell and one half of the neighbor cells. If all neighboring cells would be considered, the whole algorithm would evaluate each cell-neighborship twice. Thus all cells which are located on the opposite site of already checked cells are skipped (see Figure 11). In this manner the algorithm halves the computation complexity and ensures that every pair is found exactly once. The optimization also has the consequence that no particle gets evaluated against itself, which is ok when the density initialization takes care of the self induced density (for the forces and the color field gradient/Laplacian this doesn’t matter at all).

|  |  |
| --- | --- |
| neighbor offsets  in 3D case:  ----↓---- ----↑----  (-1,-1,-1) ( 1, 1, 1)  (-1,-1, 0) ( 1, 1, 0)  (-1,-1, 1) ( 1, 1,-1)  (-1, 0,-1) ( 1, 0, 1)  (-1, 0, 0) ( 1, 0, 0)  (-1, 0, 1) ( 1, 0,-1)  (-1, 1,-1) ( 1,-1, 1)  (-1, 1, 0) ( 1,-1, 0)  (-1, 1, 1) (1,-1,-1)  ( 0,-1,-1) (0, 1, 1)  ( 0,-1, 0) (0, 1, 0)  ( 0,-1, 1) (0, 1,-1)  ( 0, 0,-1) (0, 0, 1)  **( 0, 0, 0)**→**( 0, 0, 0)** |  |

Figure : Skip neighbor cells on the opposite side

The density calculation is not the only task where the summation interpolation and therefore the neighbor search must be performed. In the separate force and color field calculation the same neighborhood relations are needed. Therefore the particle pairs, which are found by the neighbor search during the density computation phase, are stored and reused within the following force and color field stage.

The neighbor search delivers us all particle pairs with a distance below the smoothing length. The C++-method for the density computation calculates the additional density that the two particles impose on each other () and ads it to the total densities of both particles.

Similar the pressure force, viscosity force, color field gradient and color field Laplacian calculations in the second task first compute a common term for both particles according to the appropriate SPH equation. The term gets weighted with the density inverse of the neighbor particle (which is part of all four related SPH equations) and provided with the right direction (in case of a vector) before it’s added to the particles overall values.

After the first two tasks have pair wise evaluated the density, pressure force, viscosity force and the color field values of every particle, the third task processes the particles linearly. The color field gradient and Laplacian is used to calculate the surface tension force, which ads up with pressure and viscosity forces to the total per-particle force in the current time step. Total force divided by mass density results in an acceleration (Newton’s law) that’s added to the constant earth acceleration to get the total acceleration. Current velocity, position and step duration finally lead to the new velocity and position of our particle at the end of the current simulation step, respectively the beginning of the next. Because position and velocity are the only information that’s kept for the next step, all the other property fields get cleared/initialized at the end of the calculation.

The fourth and final step clears the neighbor search grid and rebuilds it from the new particle positions. For that purpose first the new spatial dimensions of the particle cloud are calculated and a properly placed and scaled empty grid is created. Then particle after particle gets sorted into the grid according to its position, whereby new cells are created on demand if a particle falls to a position where no cell exists yet.

## Environment and user interaction

A fluid floating around in empty space is rather untypically in our everyday environment. Thus we want to simulate interaction of the fluid with solid obstacles or containers. Moreover, the name suggests that an interactive, realtime simulation should provide some sort of user interaction with the simulated object. Therefore a liquid fluid has been placed in a virtual water glass that the user can move around with the mouse. This scenario is comparatively easy to simulate and because the fluid can’t flow away, the user gets a steady simulation with which he can interact over a long time. Additionally surely everyone once watched his drink when it’s shaken around in the glass and thus we know very well how the fluid would behave in reality.

The environment interaction in this simulation works only in one direction, meaning that the movement of the simulated glass is entirely controlled by the user with the mouse and the fluid doesn’t exert forces on the glass that would cause it to move. Conceptually the glass is modeled as an infinite long, vertical aligned cylinder as side walls and a horizontal aligned plane as ground of the glass. The collision detection therefore becomes a simple check of the particles distance from the cylinders center line respectively from the bottom plane. A first implementation of the glass interaction only checked if a particle was outside the glass and repositioned it back into the glass along the border normal. However, this doesn’t lead to any physical plausible results, because thus the glass does not influence the fluid density near the border, nor does it participate in the pressure and viscosity computation. The actual implementation simulates the interaction of the glass with the fluid particles with the same SPH methods that are responsible for the particle-particle interactions. Therefore synonym to the density and forces calculation phases for the fluid itself, extra density and forces calculation phases for the glass have been added to the simulations update method. A simulation update is thus now performed in six steps:

1. update densities (particle <-> particle)
2. update densities (glass -> particle)
3. update pressure forces, viscosity forces and color field (particle <-> particle)
4. update pressure and viscosity forces (glass -> particle)
5. move particles, enforce glass boundary, clear fields
6. update the neighbor search grid

Because in some extreme situations the glass emitted pressure forces are not sufficient to keep the particles inside the glass, the fluids move-method (step 5) was equipped with a modified version of the old collision response code. It ensures that the particles don’t leave the glass too far and prevents them from permanently moving away under some extreme rotation conditions.

## Multithreading optimization

Today’s higher end consumer PC’s are all equipped with dual or quad-core CPUs. The performance of a single-process application can only profit from more than one CPU core, when it distributes its computation load among multiple threads. In that way the different cores can execute multiple parts of the computation in parallel, whereas a single-threaded application would only utilize one of the cores.

As a consequence of the simulation’s step based execution scheme, the threads do not work on long running tasks, but instead on short recurring ones. Therefore it must be possible to quickly allocate threads (creation would be too expensive), assign them a task, start their execution and wait until they are all finished with as minimal overhead as possible. For that purpose a worker-thread manager was created, that holds a pool of worker threads (per default as much as physical cores are available to the process) and offers functions for comfortable parallel execution of jobs.

Two major ways to parallelize the program execution exist: Make use of task parallelism or make use of data parallelism. At the beginning of the multi-core era on consumer PC’s, mostly task parallelism was exploited, because it’s comparatively easy to execute distinct parts of a program in parallel. However, task parallelism requires the existence of enough independent heavy-worker tasks to make use of all cores. Furthermore in a realtime application it’s unlikely that each task requires comparable execution times, so some cores will run at full capacity while other are often idle. In the case of this fluid simulation, all performance critical tasks depend on their precursor, so there couldn’t be made any reasonable use of task parallelism at all. The fluid simulation therefore makes use of data parallelism where ever it seems possible and lucrative. This means concretely that the first 5 of the 6 update tasks where parallelized:

The density calculation step begins with the grid based neighbor search. The distinct grid cells thereby provide a natural data separation criterion. Each thread only searches neighbors for particles in grid cells with an index dividable by its own id. This fine grained distribution causes an almost equal utilization of all threads. However, it doesn’t prevent the threads to find pairs with particles in neighbor cells that are handled by a different thread. This principally becomes a problem when the thread adds the additional density to the values for both particles. Because the add-operation (C++: +=) isn’t atomic at the instruction level, a simultaneous add attempt from two threads could lead to a swallow of one of the summands. To overcome this problem, one could use atomic operations at x86-instruction-set level (inline assembler; CMPXCHG-instruction) or provided by the operating system (Win32-API; InterlockedIncrement-function). However, the summation is very performance critical, so the memory barriers needed for those commands would cause an immense performance hit and with the vector values in the later phases, things would get complicated. The good news is that with many particles the probability for such a collision is very low and its consequences (losing the contribution of one particle) are not dramatically for the overall simulation. Thus the density array is only marked as “volatile” to prevent the worst multithread-errors because of caching and further possible collisions are treated as an acceptable risk. Every thread stores its own particle pair list for the later forces step, so that the same data distribution among the threads is used there. The pressure and viscosity force arrays as well as the color field arrays are also simply marked as volatile, but not further synchronized.

The tasks for glass-related density and force calculation as well as the movement task simply let every thread linearly compute on the same count of particles. Those three tasks don’t need any synchronization at all, because they always operate on distinct data.

The last task, the sort of the particles into the neighbor search grid is performed single-threaded. Because a failure with the insertion of the particles into the lists in the cells would cause major trouble to the simulation, a strong synchronization associated with a performance hit would be necessary. Performance improvements here wouldn’t make a great difference anyhow, because the insertion into the grid does need only ~5% of the total computing time of the simulation.

The work on the multithread ability of the simulation did pay off. In the tests the program version with the multithreaded fluid simulation engine achieved an 83% better overall performance on a quad-core CPU (Intel Core 2 Quad Q6600 @ 3.24 GHz) than the pure single threaded version (the frames per second of the entire application inclusive sprite rendering were measured ).

## Results

The fluid simulation produces satisfying results in terms of performance and believability of the liquid’s behavior.

The performance can be expressed in numbers: With 1728 particles (12³) and simulation of all possible forces (pressure, viscosity and surface tension) the application runs with ~330 frames per second on a PC with 3.2 GHz quad-core CPU and 2 GB RAM (measured inclusive sprite rendering which ads no measureable overhead). With 10648 particles (22³) still 53 FPS are achieved. At 27000 particles (30³) the frame-rate drops down to 14. All experiments where run with simulation time steps depended on the real elapsed time to provide a constant time behavior for the viewer.

A measure for the plausibility of the behavior is harder to find. First it should be mentioned that the application is capable to simulate all major effects that could be observed when a real liquid is shaken around in a glass: vortex formation, wave breaking, wave reflection, drop formation and drops that slowly drain down along the side of the glass

|  |  |
| --- | --- |
| wirbel.png | welle.png |
| tropfen.png | reflektion.png |

Figure : Liquid behavior

# Visualization

## Optical characteristics of water

# Conclusion

## Summary

## Improvements and alternatives

[BT07] Inkompressibilität durch Tait-Euation und hohen speed of sound und zur Oberflächenspannung durch Anziehende Kräfte

[CEL06] Inkompressibilität durch Errechnung eines Null-Divergenz-Geschwindigkeitsfeldes

# Appendix

## References

**[APK07] Adams, et al. 2007.** Adaptively Sampled Particle Fluids. *Proceedings of the 2007 SIGGRAPH conference.* 2007.

**[AIY04] Amada, et al. 2004.** Particle-Based Fluid Simulation on GPU. *ACM Workshop on General-Purpose Computing on Graphics Processors.* 2004.

**[BT07] Becker and Teschner. 2007.** Weakly compressible SPH for free surface flows. *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation.* 2007, pp. 63-72.

**[BM07] Bridson and Müller-Fischer. 2007.** Fluid simulation. *SIGGRAPH 2007 course notes.* 2007.

**[BE02] Burgess and Elst, van. 2002.** MAS209: Fluid Dynamics. *Course Material.* 2002.

**[CHJ03] Co, Hamann and Joy. 2003.** Iso-splatting: A Point-based Alternative to Isosurface Visualization. *Computer Graphics and Applications, 2003. Proceedings.* 2003, pp. 325-334.

**[CEL06] Colin, Egli and Lin. 2006.** Computing a null divergence velocity field using smoothed particle hydrodynamics. *Journal of computational physics.* 2006, 217, pp. 680-692.

**[GM77] Gingold and Monaghan. 1977.** Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices.* 181, 1977, pp. 375-389.

**[Hei07] Heinecke. 2007.** Physikalische Rauchsimulation auf Partikelbasis in Echtzeit mit der PhysX-Engine. [ed.] Technische Universität Dresden. [Minor thesis]. 2007.

**[KW06] Kipfer and Westermann. 2006.** Realistic and interactive simulation of rivers. *ACM International Conference Proceeding Series.* 2006, 137.

**[KC05] Kolb and Cuntz. 2005.** Dynamic Particle Coupling for GPU-Based Fluid Simulation. *Proc. 18th Symposium on Simulation Technique.* 2005, pp. 722-727.

**[KW03] Krüger and Westermann. 2003.** Acceleration Techniques for GPU-based Volume Rendering. *Proceedings of the 14th IEEE Visualization.* 2003, p. 38.

**[LC87] Lorensen and Cline. 1987.** Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques.* 1987, pp. 163-169.

**[Luc77] Lucy. 1977.** A numerical approach to the testing of the fission hypothesis. *Astronomical Journal.* 82, 1977, pp. 1013-1024.

**[Mon05] Monaghan. 2005.** Smoothed particle hydrodynamics. *Reports on Progress in Physics.* 2005, 8.

**[Mon92] —. 1992.** Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics.* 1992, 30, pp. 543-574.

**[MCG03] Müller, Charypar and Gross. 2003.** Particle-Based Fluid Simulation for Interactive Applications. *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation.* 2003, pp. 154-159.

**[MST04] Müller, et al. 2004.** Interaction of Fluids with Deformable Solids. *Computer Animation and Virtual Worlds.* 2004, 15, pp. 159 - 171.

**[Pap99] Papanastasiou, Georgiou and Alexandrou. 1999.** *Viscous Fluid Flow.* s.l. : CRC Press, 1999. ISBN13: 9780849316067.

**[Ura06] Uralsky. 2006.** Practical Metaballs and Implicit Surfaces. *Game Developers Conference 2006 Presentation.* 2006. http://developer.nvidia.com/object/dx10-practical-metaballs.html.

**[WND]** wikipedia.org. *Navier-Stokes equations/Derivation.* [Online] [Cited: 03 10, 2008.] http://en.wikipedia.org/w/index.php?title=Navier-Stokes\_equations/Derivation&oldid=177609104.

## Glossary

## 

Eulerian model 3

Lagrangian model 3

Navier-Stokes equations 2

smoothed particle hydrodynamics 3

Smoothed particle hydrodynamics 9

## Derivation of the gradient and Laplacian of the smoothing kernels

Used calculation rules:

Gradient (of a scalar valued function):

Laplacian (of a scalar valued function; sometimes also written or ):

Chain rule:

Product rule:

Gradient and Laplacian of :

with: ,

Gradient of :

Gradient and Laplacian of :