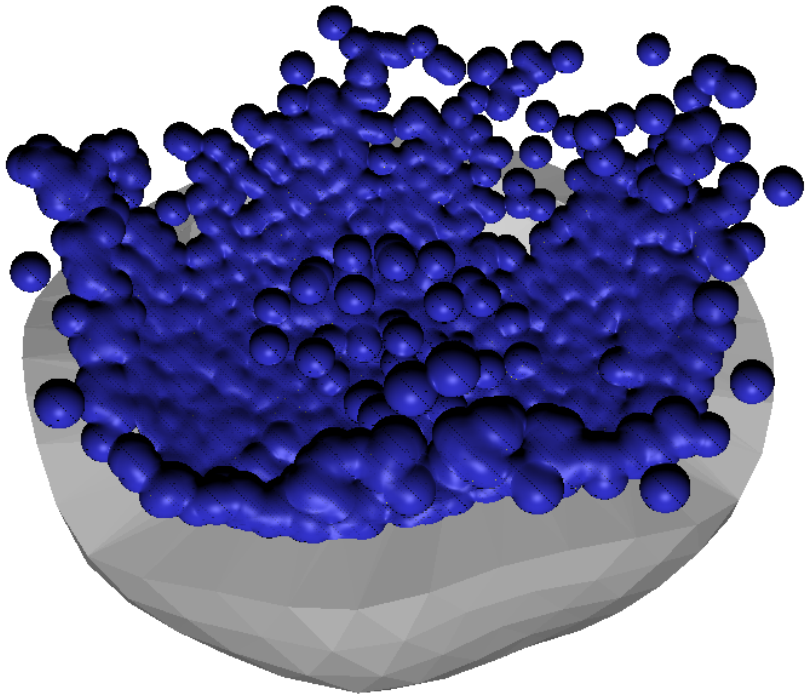
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Graphics Laboratory ETH Zurich

Fluid Effects in Cutting Simulations

Semester Thesis by Roland Angst

2006/2007



Supervised by Dr. Miguel A. Otaduy
Overseen by Prof. Dr.Markus Gross

Abstract

The goal of this semester thesis was to extend an existing software for simulating deforming solids which undergo topological changes with a fluid simulation based on smoothed particle hydrodynamics (SPH). Besides the major issue of handling the deforming solid-fluid interface, a GPU based rendering algorithm suited for realtime applications was implemented.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

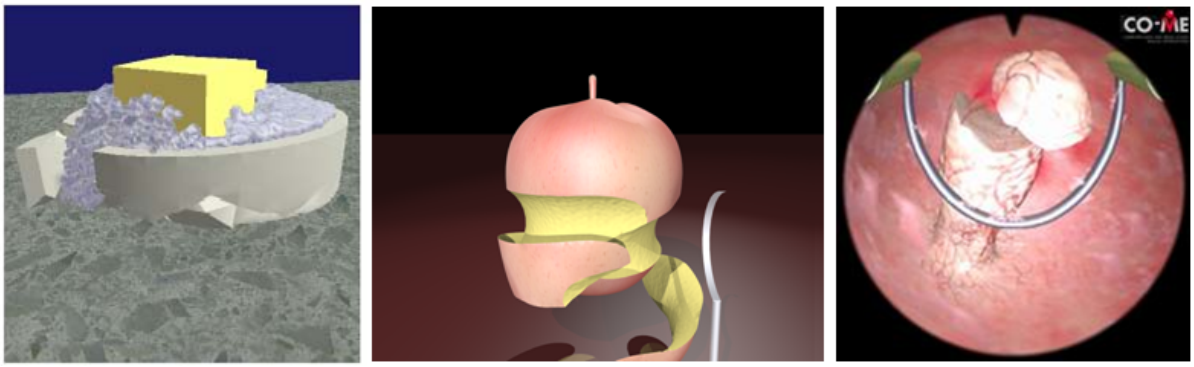
cgl computer graphics lab
Prof. Dr. M. Gross

Semester Thesis SS 2006

Fluid Effects in Cutting Simulations

Introduction

When physical objects break or are cut, it is common to observe fluids that leak through the surface cracks. As an example, cutting of soft tissue during surgical procedures usually leads to bleeding. The realism of physically-based animations for medical training can be enhanced by adding fluid simulation effects to the underlying soft-tissue deformation and cutting simulation. From a computational point of view, some of the challenges of simulating fluid effects along with cutting of deforming solids lie on the handling of fluid-fluid interfaces and interaction with surfaces that deform and change topology.



Task Description

The overall task of the project will be to simulate various fluid effects in conjunction with existing software for simulating deforming solids that undergo topological changes. These effects should include fluids that leak from surface cracks and flow along the surface, and fluids that leak and mix with other fluids. One of the objectives of the project is also to identify and propose research directions for technical and implementation challenges. More specifically, the project can be decomposed into the following main tasks:

- Integration of fluid simulation code with a framework for cutting deforming solids.
- Implementation of leaking in surfaces that change topology.
- Simulation of fluid dripping on surfaces and fluid-fluid interaction demos.

Recommended Skills

- C++ programming skills
- Physically-based simulation lecture

Contact

For further information or application to this project, please contact: Miguel A. Otaduy, IFW D25.2, Tel. 632 7546, otaduy@inf.ethz.ch

Contents

List of Figures

1

Introduction

Fluids are such common in our daily life that we got used to their complex physical behavior. We immediately remark whether a fluid behaves in a natural way or not. Therefore we have high expectation of a fluid simulation which tries to recreate the appearance and dynamics of a fluid. May it be the coffee in a mug, the eddy water in the bathtub, breaking waves at the beach or clouds moving across the sky, all these physical processes can be captured by a set of equations which describe the dynamics of fluids. The most prominent set of equations describing the dynamics of a fluid is known as Navier-Stokes equations. Although these equations are known since a long time they still pose theoretical and practical problems. For example, these equations are listed in the list of Millennium Problems of the Clay Math Institute.

Computational Fluid Dynamics (CFD) is a field in engineering which solely focuses on the simulation of fluids. CFD puts more emphasis on accuracy than on speed. The physical simulation of fluids has also found its way to computer graphics. But in contrast to CFD, computer graphics is more concerned about efficient and plausible solutions rather than highly accurate solutions. Fluid simulations in computer graphics can be distinguished between off line and real-time simulations. Applications of real-time fluid simulations are for example surgical simulations, computer games and previews for more accurate fluid solvers. Special effects in feature films are an example for off line fluid simulations in computer graphics. Fluid simulations impress the most when the fluid interacts with the world around it. For example the fluid should interact with other fluids, with rigid bodies or with deformable solids.

Not only the dynamics of a fluid are difficult to simulate, also the appearance of a fluid is a highly complex process. It is very difficult to model effects like reflection, refraction or caustics in an efficient manner. For a very good introduction to fluid simulations in computer graphics see [BFMF06].

This thesis presents a real-time fluid simulation which interacts with deformable solids. Specif-

ically, a fluid simulation method was implemented in an already existing framework for the simulation of deformable solids. The focus of the fluid and the deformable solid simulation lies in meshless simulation methods. Further the interaction between the fluid and deformable bodies was investigated to finally come up with a collision handling algorithm between the fluid and the deformable solids. A point-based rendering algorithm for the SPH simulation is presented as well.

This thesis is organized as follows. The following two chapters present some theoretical background information. The Navier-Stokes equations are presented first. Then a method to discretize and simulate partial differential equations is presented in the third chapter. This simulation method is then applied to the Navier-Stokes equations in the fourth chapter. The fifth chapter presents the collision handling between the fluid simulation and the deformable solids. The point-based rendering algorithm for the fluid is presented in the sixth chapter. Finally, the thesis is concluded with the presentation of some results and further ideas.

Note that there is no chapter about related work because it was decided to reference important literature and papers in the appropriate section throughout the thesis rather than concentrate all the previous work in one short section.

2

Navier Stokes Equations

To understand the various methods and simplifications made in fluid simulations, it is important to have a rough idea of the governing physical equations. Various different fluid models exist which differ in the assumptions and simplifications made during the derivation of the model. The most widely used fluid model is known as the Navier Stokes model and therefore this chapter mainly focuses on this model. Allthough only a special case of these Navier Stokes equations is normally implemented in most fluid simulations in the realm of computer graphics, it is good to start with the general equations and derive the special cases from this general set of equations. Otherwise it is unclear when the usage of these special cases is justified and the assumptions made for the special cases will become confusing. Therefore the general Navier Stokes equations are presented in this chapter first and then the most important special case for fluid simulations in computer graphics is derived thereof. Note that the information for this introduction was gathered from [HS89] and several web-sites [1] [2].

It is worth to mention that it is not always necessary to simulate a liquid in a physical correct manner. Procedural approaches, like the superposition of sinus waves, are computationally very cheap and can give visually pleasing results for a liquid surface. A slightly more physically inspired method uses the two dimensional wave equation to simulate a liquid surface. Both approaches are employed in many current computer games. But it is obvious that none of them can simulate more complex behaviours like splashing or plausible interactions with other objects, like for example rigid bodies. The Navier Stokes equations on the other hand provide a well established framework which can be used to derive computational methods to simulate physically correct fluids in offline and realtime applications.

This chapter derives the Navier Stokes equations by stating first material specific constitutive

[1] www.navier-stokes.net
[2] www.wikipedia.org

relations and then the balance equations which describe some fundamental conservation principles. The combination of the constitutive relations together with the balance equations finally leads to the Navier Stokes equations. Important special cases are then derived from the general form of the Navier Stokes equations.

2.1 Constitutive Relations

Solids and fluids respond completely different when taken out of the equilibrium state. The mathematical description of this response is captured by a set of relations known as the *constitutive relations, stress rate of strain or stress-strain relations*. These relations are not directly implied by physical laws but are rather derived from a set of *constitutive assumptions*. The state of a fluid is given by the five field quantities density, temperature and velocity (a 3D vector). The constitutive relations for Navier Stokes fluids relate these field quantities to the stress tensor, the heat flux vector, the internal energy and the entropy.

2.1.1 Navier Stokes Constitutive Assumptions

The constitutive relations are derived from a set of assumptions and some physically imposed relations. These assumptions are specific for a Navier Stokes fluid, thus any fluid violating these assumptions is not considered as a Navier Stokes fluid. The first assumption made for many fluid models tells that the stress tensor $\mathbf{T}$, the heat flux tensor $\mathbf{q}$, the internal energy e and the entropy s are functions of the density ρ, the absolute temperature T and the gradients of the velocity and temperature field:

$$\mathbf{T} = \mathbf{T}(\rho, T, \nabla T, \nabla \mathbf{v}) \tag{2.1}$$
$$\mathbf{q} = \mathbf{q}(\rho, T, \nabla T, \nabla \mathbf{v}) \tag{2.2}$$
$$e = e(\rho, T, \nabla T, \nabla \mathbf{v}) \tag{2.3}$$
$$s = s(\rho, T, \nabla T, \nabla \mathbf{v}) \tag{2.4}$$

Note that in contrast to fluids, the stress tensor for solids depends on the gradient of displacements rather than the gradient of velocities.

The second assumption is specific for the Navier Stokes fluid model, namely the dependency of the above equations is required to be linear in both ∇T and $\nabla \mathbf{v}$. Fluid models with this linearity assumption are known as *Newtonian fluids*, the analogon for a solid model is called a Hookean solid. The combination of these assumptions with some fundamental physical principles leads to the constitutive relations.

2.1.2 Navier Stokes Constitutive Relations

The specific constitutive relations for the Navier-Stokes equations are summarized here for the sake of completeness. They read the following:

$$\mathbf{T} = (-p + \varsigma \nabla \cdot \mathbf{v})\,\mathbf{I} + \eta\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T - \left(\frac{2}{3}\nabla\cdot\mathbf{v}\right)\mathbf{I}\right) \tag{2.5}$$

$$= (-p + \lambda \nabla \cdot \mathbf{v})\,\mathbf{I} + \eta\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) \tag{2.6}$$

$$\mathbf{q} = -k\nabla T \tag{2.7}$$

$$e = e(\rho, T) \tag{2.8}$$

$$s = s(\rho, T) \tag{2.9}$$

where $p = p(\rho, T)$ is the thermodynamic pressure ($[F/m^2]$), $\eta = \eta(\rho, T)$ is the shear viscosity ($[kg/(ms)]$), $\lambda = \lambda(\rho, T)$ is the second viscosity ($[kg/(ms)]$) and $k = k(\rho, T)$ is the termal conductivity ($[J/(msK)]$). ς is the so called bulk viscosity which opposes compressional velocity gradients. The bulk viscosity is related to the second viscosity through $\lambda = \varsigma - \frac{2}{3}\eta$.

These quantities are also required to fullfill certain inequalities which are related to the second law of thermodynamics and these inequalities are needed to ensure that fluid frictions always opposes shear gradients and heat flows from warm to cold volumes:

$$\eta \geq 0 \tag{2.10}$$

$$\varsigma \geq 0 \tag{2.11}$$

$$k \geq 0 \tag{2.12}$$

The last requirements ensure a stable thermodynamic equilibrium:

$$c_v \geq 0 \tag{2.13}$$

$$\frac{\partial p(\rho, T)}{\partial \rho} \geq 0 \tag{2.14}$$

where c_v is the specific heat capacity at constant volume. Once the quantities $p = p(\rho, T)$, $c_v = c_v(\rho, T)$, $\eta = \eta(\rho, T)$, $\varsigma = \varsigma(\rho, T)$ and $k = k(\rho, T)$ are defined, the Navier-Stokes equations are completely specified.

Since the major difference between a fluid and a solid is the dependency of the stress on the velocity gradient rather than the displacement gradient, the next subsection gives a detailed derivation of the stress tensor for Navier-Stokes fluids. This derivation should also give a better understanding of the viscosity which is typical for a fluid. Besides the derivation of this stress tensor, the exact derivation of the remaining parts of the Navier Stokes constitutive relations does not reveal any more insights and hence the interested reader is referred to a physical textbook.

2.1.3 Viscosity

The derivation of the stress tensor for a fluid is very similar to the derivation of the stress tensor for a solid, with the major difference that the velocity field is approximated rather than the

displacement field. We start with a Taylor series expansion of the velocity field:

$$\mathbf{v}(\mathbf{r}+d\mathbf{r}) = \mathbf{v}(\mathbf{r}) + \frac{d\mathbf{v}}{d\mathbf{r}}d\mathbf{r} + \cdots \tag{2.15}$$

In the case of Navier-Stokes fluids, the stress tensor is required to be only linearly dependent on the Jacobian of the velocity field $\frac{d\mathbf{v}}{d\mathbf{r}}$. It is well knonw that any matrix can be decomposed into a symmetric and into an antisymmetric part. Applying this decomposition to the Jacobian of the velocity field leads to the following linear approximation of the velocity field:

$$\mathbf{v}(\mathbf{r}+d\mathbf{r}) \approx \mathbf{v}(\mathbf{r}) + \frac{1}{2}\underbrace{\left(\nabla\mathbf{v} - (\nabla\mathbf{v})^T\right)}_{\text{antisymmetric}}d\mathbf{r} + \frac{1}{2}\underbrace{\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right)}_{\text{symmetric}}d\mathbf{r} \tag{2.16}$$

It is interesting to note that the antisymmetric part just corresponds to a rigid rotation of the fluid about $\mathbf{r}$ with an angular velocity ω:

$$\omega = \frac{1}{2}\nabla \times \mathbf{v} \tag{2.17}$$

Because a rigid body movement does not induce any change in the relative positions the antisymmetric part is of no interest for the viscous force. Under the assumption of an isotropic fluid, the symmetric part can be expanded in a coordinate independent manner:

$$\frac{1}{2}\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) = \underbrace{\left(\frac{1}{3}\nabla\cdot\mathbf{v}\right)\mathbf{I}}_{\text{rate-of-expansion tensor}} + \underbrace{\left(\frac{1}{2}\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) - \left(\frac{1}{3}\nabla\cdot\mathbf{v}\right)\mathbf{I}\right)}_{\text{rate-of-shear tensor}} \tag{2.18}$$

A fluid is called isotropic if its properties are independent of direction in space. Note that the rate-of-shear tensor is a traceless, symmetric matrix as can easily be verified. The factor of $\frac{1}{3}$ is only valid for the three dimensional case and has to be set to $\frac{1}{2}$ in two dimensions.

The next step is to transform this rate-of-shear tensor of relative velocity differences to a viscous stress tensor. The stress tensor is then used to compute the viscous forces. Since the Navier-Stokes constitutive assumptions require a linear dependency of the stress tensor $\mathbf{T}_{viscous}$ with respect to the gradient of the velocity field, the most general way to compute the symmetric viscous stress tensor is:

$$\mathbf{T}_{viscous} = \varsigma\left(\nabla\cdot\mathbf{v}\right)\mathbf{I} + \eta\left(\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) - \left(\frac{2}{3}\nabla\cdot\mathbf{v}\right)\mathbf{I}\right) \tag{2.19}$$

It can be shown that this linear combination of a traceless, symmetric matrix with a trace is the most general linear map to define the stress tensor $\mathbf{T}_{viscous}$. Note that this is not yet the final stress tensor. The viscous forces in a fluid stem from velocity differences of fluid molecules. This molecular velocity is composed of two different velocities, namely it is the sum of the fluid velocity and the thermal velocity. But the above stress tensor $\mathbf{T}_{viscous}$ only captures the velocity differences of the fluid as a continuum. Viscous effects of the fluid surface due to the thermal velocity are modeled by a surface tension vector. A simple analysis (see for example p.15 in [HS89]) shows that the magnitude of this tension vector is independent of the orientation of the infinitesimal surface element $d\mathbf{A}$ and points in the normal direction of the surface element.

Therefore these viscous effects are completely specified by a single scalar p, the *hydrostatic pressure*. Because the hydrostatic pressure is independent on the surface orientation, it is simply added to the above stress tensor:

$$\mathbf{T} = -p\mathbf{I} + \mathbf{T}_{viscous} \tag{2.20}$$

Finally, the viscous force is computed in a similar way as Hooke's law of elasticity prescribes the stress force for solids:

$$d\mathbf{F} = \mathbf{T}d\mathbf{A} \tag{2.21}$$

where $d\mathbf{A}$ is an infinitesimal small area represented in the usual vector notation $d\mathbf{A} = \mathbf{n}dA$ where $\mathbf{n}$ is the unit-normal of the area and dA is the surface area. It is interesting to compare Hooke's law with the above stress tensor for viscous forces. Hooke's law tells that the deformation (the strain) is linearly dependent on the responsible force (the stress). In the case of Newtonian fluids, the velocity differences are linearly dependent on the responsible forces (the viscous force).

2.2 Balance Equations

As mentioned previously, balance equations describe some fundamental physical conservation principles. For fluids, three balance equations are of interest. The first one describes conservation of mass, the second the conservation of momentum and the third describes conservation of energy. There are two ways to specify these conservation principles, namely the integral and the local form. The local form is derived from the integral form using (amongst other formulaes) the result of *Cauchy's Theorem*. The integral forms give a more intuitive understanding and they read the following way:

$$\frac{d}{dt}\int_V \rho dV = 0 \tag{2.22}$$

$$\frac{d}{dt}\int_V \rho\mathbf{v}dV = \underbrace{\int_{\partial V} \mathbf{t}d\partial V}_{\text{force acting on surface } d\partial V} + \underbrace{\int_V \rho\mathbf{f}dV}_{\text{body force acting on volume } dV} \tag{2.23}$$

$$\frac{d}{dt}\int_V \rho\left(e + \frac{1}{2}\|\mathbf{v}\|^2\right)dV = \underbrace{\int_{\partial V} (\mathbf{t}\cdot\mathbf{v} - Q)\, d\partial V}_{\text{power associated with surface } d\partial V} + \underbrace{\int_V \rho\,(r + \mathbf{f}\cdot\mathbf{v})\, dV}_{\text{power associated with volume } dV} \tag{2.24}$$

The first equation obviously ensures the conservation of mass. The second equation simply states that the temporal change of the momentum in a volume must equal the applied force. In this case the force is split up into the force due to the traction $\mathbf{t}$ ($[N/m^2]$) acting on the surface $d\partial V$ and into the force due to the internal body force $\mathbf{f}$ ($[N/kg]$). The third equation first appears slightly complicated but it is nothing else than the familiar law stating that the temporal change of energy is equal to the rate at which work is performed or energy is transferred. Naturally, this rate is called the power. The energy of the system is composed of the internal energy e ($[J/kg]$)

and the kinetic energy. The transferred energy is given by the heat energy Q ($[J/(s \cdot m^2)]$) leaving the volume V through the surface ∂V and the so called volumetric energy supply r ($[J/(s \cdot kg)]$). By definition, the sign of the heat energy Q is negative. An example for the volumetric energy supply is a chemical reaction within the material. The work performed during the time dt equals the force vector dotted with the displacement vector along which the force was applied. This displacement vector just equals $\mathbf{v}dt$.

The integral forms provide an intuitive interpretation of the balance equation, but they are not suitable for a numerical simulation. In contrast, the local form provides a set of differential equations which can be integrated in time in a numerical simulation. The derivation of the local form from the integral form is rather involved. First, the integral form must be written in a pure volume integral using Cauchy's Theorem. The key insight is that Cauchy's theorem allows to rewrite the traction vector $\mathbf{t}$ and the heat energy Q using tensor products:

$$\mathbf{t} = \mathbf{T}\mathbf{n} \tag{2.25}$$

$$Q = \mathbf{q} \cdot \mathbf{n} \tag{2.26}$$

$\mathbf{T}$ is the second order stress tensor and $\mathbf{q}$ is the first order heat flux tensor. Note that the tensors are independent of the surface orientation $\mathbf{n}$. This allows to apply the *Divergence or Gauss theorem* to transform the surface integrals into volume integrals. Because the integration limits do not depend on the time the temporal differentiation and the volume integration can be swapped. The resulting volume integrals are valid for any (temporally constant) volume and this implies that the integrands on both sides of the equation must be the same. The final local form looks like this:

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{v} = 0 \tag{2.27}$$

$$\rho\frac{D\mathbf{v}}{Dt} = \nabla \cdot \mathbf{T} + \rho\mathbf{f} \tag{2.28}$$

$$\rho\frac{D}{Dt}\left(e + \frac{1}{2}\|v\|^2\right) = \nabla \cdot (\mathbf{T}\mathbf{v} - \mathbf{q}) + \rho\left(r + \mathbf{v} \cdot \mathbf{f}\right) \tag{2.29}$$

As usual, $\frac{D}{Dt}$ denotes the *material* or *Lagrangian derivative*:

$$\frac{Df(t,\mathbf{x})}{Dt} = \frac{\partial f}{\partial t} + \nabla f \cdot \frac{\partial \mathbf{x}(t)}{\partial t} = \frac{\partial f}{\partial t} + (\mathbf{v} \cdot \nabla) f \tag{2.30}$$

The local form 2.27, 2.28 and 2.29 is not only valid for fluids but also for solids or rigid bodies. This is due to the fact that during the derivation only some general physical conservation principles were used. The constitutive relations are responsible for typical behaviour of a fluid or a solid!

2.3 Navier Stokes Equations

The derivation of the general form of the Navier-Stokes equations is straight forward. By substituting 2.6 and 2.7 into 2.27, 2.28 and 2.29 the following equations result:

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{v} = 0 \tag{2.31}$$

$$\rho\frac{D\mathbf{v}}{Dt} = -\nabla p + \nabla\left(\lambda\nabla \cdot \mathbf{v}\right) + \nabla \cdot \left(\eta\left(\nabla\mathbf{v} + \left(\nabla\mathbf{v}\right)^T\right)\right) + \rho\mathbf{f} \tag{2.32}$$

$$\rho\frac{De}{Dt} = -p\nabla \cdot \mathbf{v} + F + \nabla \cdot \left(k\nabla T\right) + \rho\, r \tag{2.33}$$

F is the *viscous dissipation* and is given by:

$$F = \quad \frac{1}{2}\varsigma\left(tr\left[\frac{1}{2}\left(\nabla\mathbf{v} + \left(\nabla\mathbf{v}\right)^T\right)\right]\right)^2 + \tag{2.34}$$

$$\eta\, tr\left[\left(\frac{1}{2}\left(\nabla\mathbf{v} + \left(\nabla\mathbf{v}\right)^T\right) - \frac{1}{3}tr\left[\frac{1}{2}\left(\nabla\mathbf{v} + \left(\nabla\mathbf{v}\right)^T\right)\right]\mathbf{I}\right)^2\right] \tag{2.35}$$

The viscous dissipation function relates mechanical energy of the fluid to heat energy. More exactly, the viscous dissipation function specifies the rate at which mechanical energy is converted into heat per unit volume [oSTT03].

Some important special cases of these general Navier-Stokes equations are given in the next few subsections.

2.3.1 Inviscid Fluids

An inviscid fluid is defined by setting η, λ and k to zero. This is an approximation for fluids with negligible friction. The Navier-Stokes equations therefore become much easier:

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{v} = 0 \tag{2.36}$$

$$\rho\frac{D\mathbf{v}}{Dt} = -\nabla p + \rho\mathbf{f} \tag{2.37}$$

$$\rho\frac{De}{Dt} = -p\nabla \cdot \mathbf{v} + \rho\, r \tag{2.38}$$

2.37 is the well known *Euler Equation*. This equation shows clearly that the body force counteracts the pressure force.

2.3.2 Incompressible Fluids

For incompressible fluids, the density *along* fluid element paths is constant. Thus if the initial condition for an imcompressible fluid specifies a uniform density the density will remain uniform. But note that an incompressible fluid alone does not imply a uniform density! If the speed

of the fluid is much smaller than the thermodynamic speed of sound the fluid can in general be approximated as an incompressible fluid. Such a fluid is called *stiff* which relates to the fact that even though the density variation is negligibly small the pressure variation can be large.

Often, a slightly different version is used and named as incompressible fluid. In this version, the density ρ, the thermal conductivity k and both viscosities η and λ are each fixed at a constant value and thus are uniform over the whole simulation domain. The Navier-Stokes equations simplify under these conditions to:

$$\nabla \cdot \mathbf{v} = 0 \tag{2.39}$$

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \eta \nabla^2 \mathbf{v} + \rho \mathbf{f} \tag{2.40}$$

$$\rho \frac{De}{Dt} = -p\nabla \cdot \mathbf{v} + F + k\nabla^2 T + \rho\, r \tag{2.41}$$

The important part here is that the equations 2.39 and 2.40 are decoupled from the energy conservation 2.41. This means that the unknown variables (the velocity field $\mathbf{v}$ and the pressure p) can be solved independently from the temperature field T. If necessary, the temperature field T itself can be computed once the values for $\mathbf{v}$ and p are known. For most fluid simulations in computer graphics the temperature field T is not explicitly required and thus computational power can be safed. Note that this does not mean that energy conservation is ignored. The pressure p is a function of the temperature T and density ρ (for example the ideal gas law: $p = \frac{nRT}{V} = \frac{nRT\rho}{m}$) and therefore the computation of the pressure has to take care of the energy conservation. Rather than computing the temperature field T explitly using the energy conservation 2.41 *Tait's equation* is used to compute the pressure directly:

$$p = A(\left(\frac{\rho}{\rho_0}\right)^z - 1) \tag{2.42}$$

A denotes here the speed of sound in the fluid, z is a constant depending on the fluid and ρ_0 is the reference density, mostly the density at standard conditions. Note that the density was originally assumed to be constant. But this leads to two problems. First, there is no *equation of state* for truely incompressible fluids because in reality every fluid is slightly compressible. An equation of state specifies a relationship between state variables, in the current case the relation between the pressure and the density and temperature. Second, even though there exist numerical methods to solve for incompressible fluids, the integration time step implied by the constant density assumption is very small. This justifies the usage of Tait's equation which assumes an *artificial compressibility*. Thus this approach requires the compuation of the density although it was originally assumed to be constant. This is often a source for confusion. How to compute the density will be shown in subsection 4.1.1.

A is acutally a function of the fluid entropy which in turn is again a function of the temperature. But in most fluid simulations in computer graphics, the temperature varies only slightly and therefore A can be set to a constant value determined optimally by the the average temperature during the simulation. This approximation introduces only a small error but allows to avoid the computation of the temperature field T.

Another method to deal with the pressure can be seen in Stam's Stable Fluid approach [Sta99]. In this method, the temperature is completely ignored but the pressure is used as an additional

degree of freedom to ensure 2.39. Simply said, the temperature field T is replaced by another varible, the pressure field p. This is completely fine under the conditions stated above.

The approach using Tait's equation is summarized below once again because this version of the Navier-Stokes equations will be used for the Smoothed Particle Hydrodynamics simulation.

$$\nabla \cdot \mathbf{v} = 0 \tag{2.43}$$

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \eta \nabla^2 \mathbf{v} + \rho \mathbf{f} \tag{2.44}$$

$$p = A(\left(\frac{\rho}{\rho_0}\right)^z - 1) \tag{2.45}$$

3

Smoothed Particle Hydrodynamics (SPH)

This chapter gives first a very rough overview of existing numerical simulation methods for partial differential equations. After that, the smoothed particle hydrodynamics (SPH) method is presented in detail.

3.1 Mesh-Based and Mesh-Free Methods

Numerical simulations for partial differential equations can be classified according to two criteria. First, a simulation is either mesh-based or mesh-free. Mesh-based methods are the more traditional way of simulating physical phenomena. They decompose the simulation domain into a mesh or even a regular grid. The finite element method is an example for a mesh-based simulation method. The decomposition of the domain into a mesh prooves as a very difficult task. For example, the stability of mesh-based methods depends on the decomposition of the domain. Of course, the decomposition can be adapted during the simulation to take care of a new circumstance. But in general, the mesh-based decomposition of the simulation domain is very unflexible, expensive and tedious to update. The second crierion classifies methods into Eulerian or Lagrangian methods. Lagrangian methods make use of the *Lagrangian Derivative* (also known as *Material Derivative*) whereas Eulerian methods use derivatives based on the normal Eulerian coordinate system. Informally, this means that Lagrangian methods simulate and trace particles through the simulation domain whereas Eulerian methods observe the very same point in the simulation domain during the whole simulation. In the following subsections, examples of fluid simulation methods for the different classifications are described shortly.

3.1.1 Eulerian Mesh-Based

Eulerian mesh-based methods are probably the most classical approach taken in fluid simulation. They normally divide the spatial simulation domain into a regular grid and discretize the partial differential equation with respect to this grid. A finite difference discretization is probably the simplest way to discretize the differential operators involved in the partial differential equation. Stam's seminal paper 'Stable Fluids' [Sta99] presents an *unconditionally stable* Eulerian mesh-based method. This means that an arbitrary large integration time step can be used and the simulation will never blow up and explode! The method achieves this unconditional stability mainly due to the usage of the *semi-Lagrangian advection scheme*. Rather than integrating a value at a grid location through time, the advection is solved by tracing a grid location backwards in time to finally interpolate the value from existing grid values. Interpolation is necessary because in general the backward traced grid location will not lie exactly on another grid location. Provided an appropriate interpolation scheme is used, the absolute value of the interpolated value can never become larger than the maximum of the grid values used for the interpolation and thus the simulation is unconditionally stable. The interested reader is referred to Stam's paper for further details.

The regular grid structure of Stam's method makes this simulation suitable for an implementation on GPU. See for example [Fer04] for a very nice description of how this can be done. On the negative side, Stam's method smooths the fluid too much. It is therefore difficult to simulate a nearly inviscid fluid with Stam's method. Many extensions have been proposed in recent years to overcome this drawback.

3.1.2 Lagrangian Mesh-Based

A Lagrangian mesh-based method is for example the *Lattice Boltzmann method*. The basic idea is to trace particles through a lattice mesh. At every grid node, the simulation values are computed based on local dynamics and on the number of particles present at grid nodes. This allows handling of complex boundaries and parallelization. Instead of just modeling a discrete number of particles per grid node, the number of particles is modelled by a density function.

The Lattice Boltzmann method is a rather new, but very promising approach for fluid simulations. For more details, see for example [SBH91].

3.1.3 Lagrangian Meshless

There exist many different methods belonging to this class of simulation methods. A very prominent representative of this class is the SPH method. SPH was developed for simulating astrophysical phenomenons. Gingold, Monaghan [GM77] and Lucy [Luc77] derived the method independently in 1977. Since then, the method was extended and applied to many different problems, for example to simulate solids and fluids. The rest of this chapter gives an introduction to the SPH method.

3.2 Smoothed Particle Hydrodynamics

This section first derives the SPH method and then states some consistency considerations. Then it is shown how the continuous SPH formulation can be discretized to be applicable to a partial differential equation, for example.

3.2.1 Kernel Approximation

A function f can be transformed to an *integral representation* using the $\delta(\mathbf{x})$ Dirac Delta function:

$$f(\mathbf{x}) = \int_{-\infty}^{+\infty} f(\xi)\delta(\mathbf{x} - \xi)d\xi \tag{3.1}$$

Note that this representation is exact. The *kernel approximation* is simply given by replacing the Dirac Delta function (which actually is not really a function but rather a distribution) by a *kernel function* $W(\mathbf{x}, h)$:

$$\hat{f}(\mathbf{x}) = \int_{\Omega} f(\xi)W(\mathbf{x} - \xi, h)d\xi \tag{3.2}$$

h is a parameter controlling the size of the influence volume Ω. The kernel function W obviously has to fullfill certain properties to make this approximation meaningful. In the case of a discretized kernel approximation, the approximation is meaningful if it converges to the true function when the sampling rate goes to infinity (or equivalently when the spacing between samples approaches zero). The convergence behaviour of the SPH approximation is presented in more detail in the next section, here the conditions on the kernel function W are just stated as a summary:

$$W(\mathbf{x}, h) = 0 \qquad \text{if } x \notin \Omega \tag{3.3}$$

$$\int_{\Omega} W(\mathbf{x} - \xi, h)d\xi = 1 \tag{3.4}$$

$$W(\mathbf{x}, h) \to \delta(\mathbf{x}) \qquad \text{for } h \to 0 \tag{3.5}$$

Note that 3.5 is a redundant requirement. The above conditions are mathematically required to ensure certain convergence properties. Two more conditions must be fullfilled additionally to give physical meaningful results:

$$W(\mathbf{x}, h) > 0 \text{ for } x \in \Omega \tag{3.6}$$

$$W \text{ is a monotonically decreasing function} \tag{3.7}$$

These requirements ensure for example positive density and a decreasing force influence with increasing distance between two particles.

Besides these theoretical constraints, some practical issues should be considered. The computational cost of the SPH method depends a lot on the computational cost to evaluate the kernel

function. It is therefore advantageous to design a kernel which avoids the computation of square roots. On current CPUs, two square root computations are more expensive in terms of processor cycles than the inversion of one 4×4 matrix! Further some special attention has to be paid to the derivatives of the kernel function. But note, nothing disallows to use a different kernel function for each simulated value. This fact will ineed be useful for SPH fluid simulations.

3.2.2 Consistency

We define a kernel function approximation $\hat{f}$ to be C^k *consistent* if it can reproduce polynomial functions with a maximal degree k. It is trivial to see that 3.4 ensures C^0 consistency. To derive the requirements for C^1 continuity the affine polynomial function f is written as $f(\mathbf{x}) = \alpha + \mathbf{c}^T\mathbf{x}$. Then C^1 consistency requires:

$$\alpha + \mathbf{c}^T\mathbf{x} = \int_\Omega (\alpha + \mathbf{c}^T\xi)W(\mathbf{x} - \xi, h)d\xi \tag{3.8}$$

Using the unity condition and considering that this equation has to be valid for any $\mathbf{c}$ we get:

$$\mathbf{x} = \int_\Omega \xi W(\mathbf{x} - \xi, h)d\xi \tag{3.9}$$

If the unity condition is multiplied on both sides by $\mathbf{x}$ we get:

$$\mathbf{x} = \int_\Omega \mathbf{x} W(\mathbf{x} - \xi, h)d\xi \tag{3.10}$$

Subtracting 3.9 from 3.10 gives:

$$0 = \int_\Omega (\mathbf{x} - \xi)W(\mathbf{x} - \xi, h)d\xi \tag{3.11}$$

Thus for C^1 consistency the first moment of the weight function must vanish. Note that a function which is symmetric about the origin has a vanishing first moment. In practice, it is not difficult to define a symmetric kernel function. But still, for points near the boundary where normally not the whole smoothing volume is inside the simulation domain it is difficult to define a symmetric kernel function. This fact is the reason for many problems in SPH fluid simulations.

Similar derivations can be used to proof C^k consistency of certain kernel functions. Note that these consistency considerations are valid for the continuous definition of the kernel approximation but this does not imply the same degree of consistency for the discretized kernel approximation.

3.2.3 Discretization

The continuous kernel approximation

$$\hat{f}(\mathbf{x}) = \int_\Omega f(\xi)W(\mathbf{x} - \xi, h)d\xi \tag{3.12}$$

is discretized by a simple nodal quadrature rule which is also known as *particle approximation*:

$$\hat{f}(\mathbf{x}) = \sum_i f_i W(\mathbf{x} - \mathbf{x}_i) \Delta V_i \tag{3.13}$$

where $f_i = f(x_i)$ and ΔV_i is the volume represented by particle i. Therefore every node or particle has to be associated with its volume. This is more difficult than might appear because there is no grid structure available. In SPH fluid simulations the volume ΔV_i of particle i is given by $\Delta V_i = \frac{m_i}{\rho_i}$ where m_i is the mass associated with particle i and ρ_i is the density of particle i. Since the mass of a particle is normally considered to stay constant and the density is a field variable which is computed for every particle at every time step, only a volume initialization is required. During the simulation itself, the volume is computed by the above formula.

The good thing about this representation is that the first spatial derivate of the approximated function becomes a summation over derivatives of the kernel function which can be designed to be cheap to evaluate. For example, if the kernel function is even the derivative becomes:

$$W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h) = W(\|\mathbf{x}_i - \mathbf{x}_j\|, h) = W(r_{ij}, h) \tag{3.14}$$

$$\nabla_i W_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}} = \frac{\mathbf{x}_{ij}}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}} \tag{3.15}$$

$$\nabla \hat{f}(\mathbf{x}_i) = \sum_j f_j \Delta V_j \nabla_i W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{3.16}$$

where r_{ij} is the Euclidian distance between particle i and j. This formula can be derived from the continuous case. Starting with

$$\nabla \hat{\mathbf{f}}(\mathbf{x}) = \int_\Omega \nabla \mathbf{f}(\xi) W(\mathbf{x} - \xi, h) d\xi \tag{3.17}$$

and integration by parts gives:

$$\nabla \hat{\mathbf{f}}(\mathbf{x}) = \int_\Omega \nabla \mathbf{f}(\xi) W(\mathbf{x} - \xi, h) \tag{3.18}$$

$$= [\mathbf{f}(\xi) W(\mathbf{x} - \xi, h)]_{\partial\Omega} - \int_\Omega \mathbf{f}(\xi)(-1) \nabla W(\mathbf{x} - \xi, h) d\xi \tag{3.19}$$

$$= \int_\Omega \mathbf{f}(\xi) \nabla W(\mathbf{x} - \xi, h) d\xi \tag{3.20}$$

It is interesting to compare the SPH formulation with the finite element formulation. The SPH method can be stated in the same form as the finite element formulation:

$$\hat{f}(\mathbf{x}) = \sum_i \phi_i(\mathbf{x}) f_i \tag{3.21}$$

where $\phi_i(\mathbf{x}) = W(\mathbf{x} - \mathbf{x}_i) \Delta V_i$. Note that although this notation looks equivalent to the finite element method, there is a distinctive difference between the two methods. Namely the finite

element *shape functions* ϕ_i normally satisfy the Kronecker Delta function property:

$$\phi_i(\mathbf{x}_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3.22}$$

This is not the case for the SPH method. Therefore the discretized kernel function (what SPH essentially is) approximates a function rather than interpolates it and in general we have $f_i \neq \hat{f}(\mathbf{x}_i)$! Designing a kernel which satisfies both the Kronecker Delta function property and the SPH kernel conditions is very difficult.

4

SPH for Fluid Simulations

This chapter gives a detailed presentation of the SPH method applied to the Navier Stokes equations. The SPH model described in this chapter follows closely the presentation in [MCG03] and [Liu03]. Throughout this chapter, the version 2.43, 2.44 and 2.45 of the Navier Stokes equations are used. To recapitulate, this means the density ρ, the thermal conductivity k, and both viscosities η and λ are constant and uniform over the whole fluid volume. As mentioned in a previous chapter, in contrast to the theoretically stated assumption, the density is for practical reasons actually not considered to be constant. In the constitutive relation, the parameter z is set to one, the Navier Stokes equations thus look like:

$$\nabla \cdot \mathbf{v} = 0 \tag{4.1}$$

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \eta \nabla^2 \mathbf{v} + \rho \mathbf{f} \tag{4.2}$$

$$p = A(\frac{\rho}{\rho_0} - 1) \tag{4.3}$$

Note that is is possible to derive the SPH model for a more general version of the Navier Stokes equations, but for the purpose of this thesis this is not necessary and the interested reader is therefore referred for example to the textbook [Liu03].

4.1 SPH Formulation of Navier-Stokes Equations

A SPH fluid simulation spatially discretizes the fluid continuum as a discrete set of particles. The field quantities of the Navier Stokes equations are approximated using the particle approximation 3.13 which requires that a volume is associated with each particle. The volume of a particle i is obviously given by $\Delta V_i = \frac{m_i}{\rho_i}$. Intuitively, the formulation 3.13 smears sampled

values at the particle locations over the simulation domain. The SPH method is a Lagrangian method since the particles float *with* the fluid. Hence the (nonlinear) advection of the density and the velocity are handled naturally by storing a sample of the density and the velocity locally at each particle. This Lagrangian formulation largely simplifies the governing Navier Stokes equations. The dynamics of a particle, i.e. its acceleration due to internal and external forces, is then simply given by:

$$\mathbf{a}_i = \frac{D\mathbf{v}_i}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \frac{\mathbf{f}_i}{\rho_i} \tag{4.4}$$

where $\mathbf{f}_i = -\nabla p + \rho \mathbf{f} + \eta \nabla^2 \mathbf{v}$, $\mathbf{f}$ is the applied *body force*.

The easiest way to implement a SPH fluid simulation is to fix the mass of every fluid particle at a constant value. In contrast, the volume of a particle will change in every simulation step because the density is a physical quantity coupled to the differential equations through the equation of state 4.3. There are basically two different approaches to compute the fluid density of a particle in the SPH method, the *density summation* and the *continuity density* method. Both density computation methods are described in the next subsection. The following subsections are then concerned about how to compute the individual terms in the body force of a particle. Doing so, an inherent problem of the SPH method will become obvious. Namely, the symmetry of forces between pairs of interacting particles is not automatically guaranteed and conservation of momentum is thus generally not fulfilled. Solutions to this problem will be presented in the following subsections as well.

4.1.1 Density

The density summation method can be derived straight forward by inserting the density into the discretized kernel approximation 3.13:

$$\rho_i = \sum_{j \in Neighbors(i)} \rho_j W_{ij} \Delta V_j \tag{4.5}$$

$$= \sum_{j \in Neighbors(i)} \rho_j W_{ij} \frac{m_j}{\rho_j} \tag{4.6}$$

$$= \sum_{j \in Neighbors(i)} m_j W_{ij} \tag{4.7}$$

The unknown value of the particle density ρ_j on the right hand side just cancels out and the density is equal to a weighted sum of the particle masses. The conservation of mass is globally guaranteed throughout the simulation if the density summation method is used for the density computation. The first equation of the Navier Stokes equation is thus fulfilled almost for free.

The continuity density method starts with the continuity equation $\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}$. The right hand side can be rewritten using the product rule $\nabla \cdot (\rho \mathbf{v}) = \rho \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla \rho$

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v} \tag{4.8}$$

$$= -\nabla \cdot (\rho \mathbf{v}) + \mathbf{v} \cdot \nabla \rho \tag{4.9}$$

Inserting this last formula in the discretized kernel approximation leads to the following Lagrangian derivative of the density of particle i:

$$\frac{D\rho_i}{Dt} = \sum_{j \in Neighbors(i)} m_j(\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla W_{ij} \tag{4.10}$$

Because the particles float with the fluid velocity field the above Lagrangian equation can be integrated in time to get the change in the density which then is added to the density of the previous time step. Obviously, the densities must be somehow initialized at the first time step.

The density summation method preserves the mass of the fluid exactly but on the other side this approach leads to problems at the boundary of the fluid because the density is smoothed out too much in these areas. In contrast the continuity density method does not preserve the mass exactly but does not lead to wrong densities near boundaries. Note that it is possible to combine both methods, namely to use the density summation approach in the fluid interior and the continuity density method for particles close to the boundary. In this thesis, only the density summation approach is implemented. We again refer to the textbook [Liu03] for further references and details about how to increase the accuracy of the SPH approximation.

4.1.2 Pressure

The first term in the body force depends on the gradient of the pressure field function. In the kernel approximation, such a gradient simply turns into a weighted sum over the spatial first derivatives of the kernel function:

$$\nabla f(\mathbf{x}) \approx \nabla \hat{f}(\mathbf{x}) = \sum_j m_j \frac{f_j}{\rho_j} \nabla W(\mathbf{x} - \mathbf{x_i}, h) \tag{4.11}$$

Thus in the case for the pressure gradient

$$\mathbf{f}_i^{\text{pressure}} = -\nabla p(\mathbf{x_i}) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{4.12}$$

But this formulation has a severe drawback. The pressure gradient of particle i only depends on the pressure of its neighboring particles p_j. Thus the pressure summands between two neighboring particles i and j are not equal $m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j) \neq m_i \frac{p_i}{\rho_i} \nabla W(\mathbf{x}_j - \mathbf{x}_i)$ and the resulting pressure force is not symmetric and violates the conservation of momentum.

Luckily, there are two ways to fix this. The first is more a heuristic and simply takes the average of the interaction between two neighboring particles:

$$\mathbf{f}_i^{\text{pressure}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h). \tag{4.13}$$

The other method is analytically well founded. The product rule for differentiation gives us the equation $\nabla \left(\frac{p}{\rho}\right) = \frac{\nabla p}{\rho} - p\frac{\nabla \rho}{\rho^2}$ which can be used to reformulate the pressure force term in the

body force:

$$\frac{\nabla p_i}{\rho_i} = \underbrace{\sum_j m_j \frac{p_j}{\rho_j^2} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h)}_{\nabla_i \left(\frac{p_i}{\rho_i}\right)} + \underbrace{\frac{p_i}{\rho_i^2} \sum_j m_j \nabla W(\mathbf{x}_i - \mathbf{x}_j, h)}_{p_i \frac{\nabla_i \rho_i}{\rho_i^2}} \tag{4.14}$$

$$= \sum_i m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2} \right) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{4.15}$$

The resulting acceleration due to pressure force is then: $-\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h)$. Compared to the first method to make the pressure force symmetric, the second method involves more computations and thus the first method was chosen and implemented.

Note that there is another issue in this pressure gradient computation. The kernel function enters these equations through its spatial derivative, so it must be ensured that this spatial derivative is well behaved in the neighborhood of a particle. This point will be discussed in more detail in section 4.2.

The last missing part is how to compute the pressure p_i of particle i. Recalling the equation of state $p = A \left(\left(\frac{\rho}{\rho_0} \right)^z - 1 \right)$, setting $z = 1$ and reformulating this equation, we get:

$$p = \frac{A}{\rho_0} (\rho - \rho_0) \tag{4.16}$$

This equation ensures that a deviation to the rest density ρ_0 gives rise to a pressure gradient proportional to $\frac{A}{\rho_0}$. [Kei06] gave us a hint that in an actual implementation, two different values for A should be used, one when the density difference is negative, the other when it is positive.

4.1.3 Viscosity

The viscosity force requires the definition of a discrete Laplacian. The kernel approximation of the Laplacian follows the same approach as for the gradient. The Laplacian operator applied to a field quantity leads to a weighted sum of the Laplacian applied to the kernel function:

$$\nabla^2 f(\mathbf{x}) \approx \nabla^2 \hat{f}(\mathbf{x}) = \sum_j \frac{f_j}{\rho_j} \nabla^2 W(\mathbf{x} - \mathbf{x_j}, h) \tag{4.17}$$

Direct application of this approach to the viscosity gives $\mathbf{f}_i^{\text{viscosity}} = \eta \sum_j m_j \frac{\mathbf{v}_j}{\rho_j} \nabla^2 W(\mathbf{x}_i - \mathbf{x}_j, h)$. But again, this force is not symmetric. Recalling that viscosity is a measure of how quickly the velocity at a point in space converges to the mean velocity of its immediate neighborhood an obvious fix to get a symmetric force pops up. Instead of using the true velocity of particle j rather use the relative velocity difference between particle i and j:

$$\mathbf{f}_i^{\text{viscosity}} = \eta \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{4.18}$$

[MCG03] gives a very nice and figurative interpretation of this force: "The particle i is accelerated in the direction of the relative speed of its environment."

4.2 Kernel Functions

A major issue in kernel approximations is the choice or the design of an appropriate kernel function. In addition to the conditions to a kernel function stated in the derivation of the SPH kernel approximation, applications targeting real time performance require the kernel to be evaluable efficiently. Because the kernel is evaluated many, many times during a single simulation step this is high priority. The kernels also have a large impact on the accuracy and stability of the SPH approximation. The kernels presented and used in this work all are even and normalized and thus give the SPH approximation second order accuracy. Because the spatial derivative of the kernel is used, too, the gradient of the kernel at the boundary of the support volume should vanish. Note again, there is no requirement to use the same kernel function for the gradient as for the function itself. To avoid some problems for certain terms a specifically adapted kernel function is used.

Normally, the SPH model in this work makes use of the so called *Poly6-Kernel* [MCG03]. The distance between two neighboring particles enters the kernel function squared and thus no expensive square root compuation is required. The Poly6 kernel looks like this:

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

As pointed out in [MCG03], using this kernel for the pressure gradient the particles start to form clusters under high pressure. A closer look at the gradient of the Poly6 kernel reveals that the gradient at the center converges to zero and therefore no repulsion force can push the particles apart. [DG96] designed the *Spiky Kernel*

$$W_{\text{spiky}}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{4.20}$$

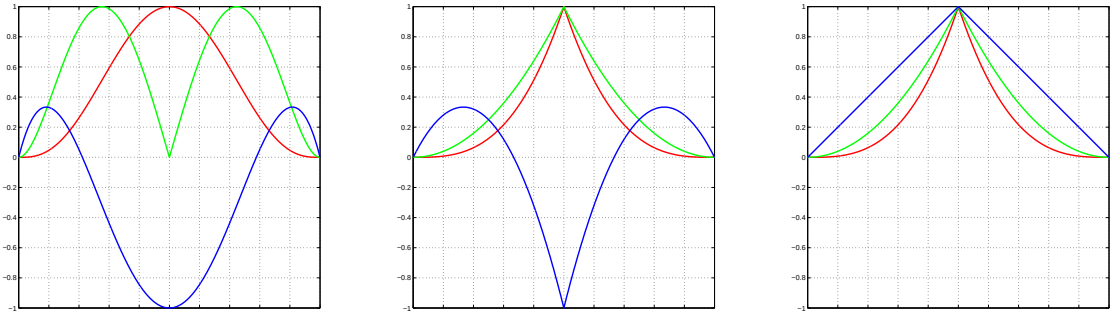
whose gradient does not vanish at the center. The first and second derivative of this kernel evaluate to zero on the boundary.

The viscous term in the body volume gives rise to one more problem. Viscosity results in a force which smooths the velocity field. But the Laplacian (on which the viscous force term depends) of the Poly6 kernel is mostly negative in the support volume. This negative velocity might increase the relative velocity between two particles, as was already pointed out by [MCG03]. According to this paper, this mainly occurs in coarsely sampled domains which is indeed the case for real time applications. We used a corrected version of the kernel presented in [MCG03]

$$W_{\text{viscosity}}(\mathbf{r}, h) = \frac{45}{13\pi h^3} \begin{cases} \frac{hr^2}{2} - \frac{r^3}{6} & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

The Laplacian of this kernel is $\nabla^2 W(\mathbf{r}, h) = \frac{45}{13\pi h^4}(h - r)$ which looks similar to the Laplacian presented in [MCG03]. Again, the gradient and Laplacian of this kernel vanish on the boundary which increases the stability of the simulation. These three kernels are visualized in Figure 4.1.

Finally, some general comments concerning the kernels are advisable. The design and choice of good kernels is almost an art and tuning the parameters is a tedious work. A fluid simulation



Poly6 Kernel Spiky Kernel Viscosity Kernel

Figure 4.1: Kernels used in the SPH method. (Red) Kernel value (Green) Gradient of kernel (Blue) Laplacian of kernel
Note that the kernels and the individual curves are scaled differently in these plots.

requires the definition of quite a lot of different parameters and their combination influences the choice of the smoothing length h in some way. We also remarked that a lot of experimentation was required to get a plausible behavior of the fluid simulation. It is also possible to use an adaptive smoothing length which takes the fluid density into account, i.e. the smoothing length differs from particle to particle (see for example [HK89], [Ada07]). But be aware that such an approach increases the complexity of the implementation a lot and also implies an overhead to update additional data structures. This overhead has to be at least balanced by the decreased number of particles required to simulate the fluid in the same quality as without a multi resolution approach. The surface reconstruction needs special treatment, as well. Otherwise unpleasing popping will appear whenever the resolution near the surface changes. The kernels are normally assumed to be spherically symmetric. [SMVO96] shows that it is possible to replace the smoothing length h by an anisotropic smoothing tensor. This tensor is adapted dynamically to take care of changes in the "local mean interparticle spacing with direction around each fluid element"' ([SMVO96]). Due to the additional degrees of freedoms a practical implementation becomes cumbersome.

5

Collision Detection

A major task of this semester thesis consisted in the development and implementation of a collision handling algorithm between the SPH fluid simulation described in chapter 4 and the already existing framework for deformable bodies [SOG06]. This chapter proceeds by first giving a short introduction to the simulation presented in [SOG06], then a motivation for the chosen collision handling algorithm is given. Finally the collision handling is described in details.

5.1 Deformable Solid Framework

In [SOG06], a method is presented to simulate deformable solids which can be cut and split into separate smaller deformable solids. The PDE governing the physics of deformable solids looks as follows:

$$\rho \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla \cdot \sigma(\mathbf{u}) + \mathbf{f}$$

where $\mathbf{u}$ is the displacement field, σ is the stress tensor and $\mathbf{f}$ are external forces. The simulation domain corresponds to the volume of the deformable solid in undeformed state. Coordinates in this undeformed state are called material coordinates. The displacement field $\mathbf{u}(\mathbf{x})$ is added to the material coordinate $\mathbf{x}$ of a point of a deformable solid to define the final position of this point. The stress tensor at a given point is a function of the displacement at this point.

Most often, this PDE is discretized with the finite element method. To do so, the volume of the deformable solid is explicitly partitioned into finite elements. In the most simple case, the volume is discretized using a tetrahedral mesh. But because the deformable objects can get cut along arbitrary piecewise linear crack surfaces such a tetrahedral mesh must be updated during

a cutting step. It is highly non-trivial to adapt the internal structure of the tetrahedral mesh to the new topology of the deformable object. This is mainly due to the fact that the finite element method suffers from stability problems in case of badly shaped elements. Such badly shaped elements are difficult and computationally expensive to avoid along arbitrary splitting fronts.

The approach presented in [SOG06] is a meshless simulation method and does not make use of the finite element formulation. The displacement field is sampled at several simulation nodes. The displacement $\mathbf{u}$ of a point with material coordinates $\mathbf{x}$ is computed using a moving least squares approach as already presented in [MKN+04]:

$$\mathbf{u} = \frac{1}{\sum_i \omega(\mathbf{x}, \mathbf{x}_i)} \sum_i \omega(\mathbf{x}, \mathbf{x}_i) \left(\mathbf{u}_i + \nabla \mathbf{u}_i^T (\mathbf{x} - \mathbf{x}_i)\right)$$

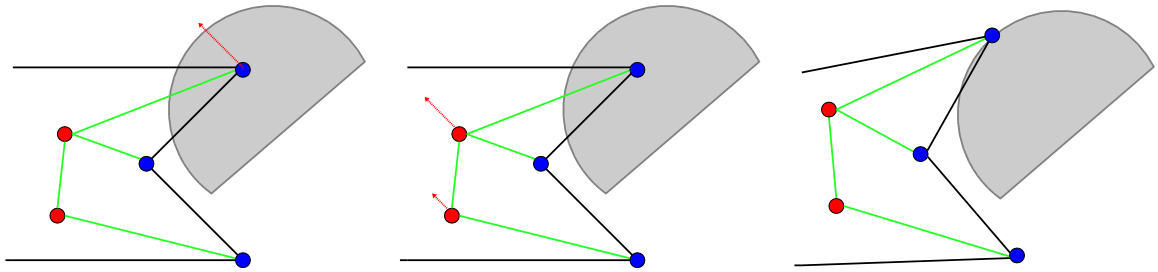
The summation is done over all simulation nodes in the neighborhood of the point with material coordinates $\mathbf{x}$. ω is an appropriate kernel function to smoothly smear the values at the sampling nodes over the whole domain. The evaluation of the kernel functions requires proximity information between simulation nodes. Note that it is not possible to simply use the Euclidean distance because the true *intrinsic* distance is required. Although two points might be close together measured in three dimensional Euclidean distance, if there was a (deep) cut right between them the distance relevant for the deformable solid computation might be very large because this distance corresponds to the length of the shortest path between the points *without* leaving the volume of the object. Obviously if a piece of the deformable object is completely cut away, two separate deformable bodies need to be simulated and a point inside one object should not influence points in the other object at all (ignoring collision effects).

The method in [SOG06] approximates the intrinsic distance with a so called visibility graph. Although this graph stores connectivity information between simulation nodes it does not require an explicit partitioning of the simulation domain. Of course, this data structure needs to be updated whenever a cut occurs.

This simulation method suffers from a similar problem as the SPH fluid simulation method. Namely the discretization of the simulation domain does not allow an easy extraction of a surface. But there is an important difference between these two meshless simulations. A fluid can undergo more complicated topological changes, i.e. it can split and merge multiple times. A deformable body on the other hand is only assumed to split. A crack front never merges again. This motivates the idea of carrying along an explicit triangle surface. Whenever a cut occurs, this triangle surface is updated to match the new circumstances. The displacement of a vertex is approximated based on the internal physical simulation nodes. To do so, the vertices are linked to the visibility graph, too. But note that they are no physical simulation nodes, they are just passive nodes. The interested reader is referred to the paper [SOG06] for further details.

5.2 Collision Handling between Deformable Solids

In order to understand the fluid-deformable solid interaction, it is important to understand how the collision response of a deformable solid in [SOG06] works. For a visual depiction of the collision response, see Figure 5.1. The collision detection between two deformable solids is done via the triangle surfaces. If the two objects intersect a collision response is computed



Step 1 Step 2 Step 3

Figure 5.1: Collision response for deformable solids. Red dots: physical simulation nodes; blue dots: surface vertices; green lines: visibility graph; black line: deformable solid surface; gray: collision partner;
In a first step a collision response (red arrow) is computed at the vertices (Step 1). Then this collision response is propagated backwards to the simulation nodes (Step 2). Finally the collision response is applied to the simulation nodes and new displacements are compute for the vertices (Step 3).

and accumulated at the surface vertices. The collision response is propagated backwards to the physical simulation nodes using the visibility graph. The collision response is then applied at the physical simulation nodes. The collision response changes the internal state of the simulation and therefore new vertex positions of the surface have to be computed by using the updated displacement field defined by the simulation nodes.

There are two very important properties of this collision handling algorithm. First, the surface vertices might still end up penetrating the collision partner because the collision response is not applied at the same place as the collision detection takes place. Second, there is absolutely no direct control over the final vertex positions without an expensive iterative method because the collision response which was computed at the vertices might lead to an undesired displacement field after applying it to the simulation nodes.

5.3 Collision Handling Algorithm

Our first approach to model the interaction between the fluid and deformable solids was a particle based collision handling as described in [MST+04]. Such an algorithm would fit nicely in the meshless simulation methods of the fluid and the deformable solids. The basic idea is to augment the SPH fluid simulation with another kernel. This kernel is designed to mimic a repulsion force when the fluid particle approaches the surface of a solid. This algorithm is quite fast, but it did not work well in our case. On one hand the fluid particles were often repelled way too strong and on the other hand fluid particles sometimes still leaked through a surface. It is of uttermost importance for a pleasing simulation to ensure that no fluid particles are leaking through the surface. Continuous collision detection is very well suited for this kind of task. We therefore decided to abandon the particle based collision handling in favor of a continuous collision detection approach.

A fully fledged collision handling would take care of both effects, the effect of the fluid on the

deformable solid and the effect of the deformable solid on the fluid. But remember that the collision response of the deformable objects does not allow direct control over the final vertex positions of the surface. This makes it almost impossible to design a real-time collision handling algorithm which computes an equilibrium state between the forces due to the two-sided effects between the fluid and the deformable solid and which in addition prevents fluid particles from leaking through the surface. See for example [BFMF06] for further details on the collision handling involving fluids and for the simulation of fluids in general. Considering these facts we decided to only model the effect of the deformable solid on the fluid. The fluid itself will exert no force on the deformable solid.

Our collision handling algorithm consists of four steps, see Figure 5.2 for a graphical depiction. The collision detection is performed between the fluid particles and the triangle surface of a deformable solid. First the fluid particles as well as the deformable solid are integrated in time to get the new positions at time $t + \Delta h$. Then collision partners, i.e. a triangle of the surface and a fluid particle, are detected. This is done with a continuous collision detection test which intersects a ray with a volume swept by a triangle. The ray is defined by the position of the fluid particle at time t and $t+\Delta h$ whereas the swept volume of the triangle is defined by the positions of its vertices at time t and $t+\Delta h$. If there is indeed an intersection, the barycentric coordinates of the intersection point $\mathbf{p}(t_c)$ as well as the intersection time $t_c \in [t, t + \Delta h]$ are stored. Further a fluid particle velocity update is computed to reflect the velocity from the surface. This basically corresponds to an impulse based collision response. Note that if there are multiple intersections for a fluid particle, the first intersection w.r.t. time is stored. In the third step the position of a colliding fluid particle is moved to the point defined by the barycentric coordinates of the intersection point. But note that the barycentric coordinates are now interpreted w.r.t. the triangle at the end of the time step! In addition the velocity of the particle is updated with the velocity which has been computed in the second step. The fluid particle is then integrated in time from the time of intersection to the end of the time interval. Finally the continuous collision detection has to be applied once again, because a fluid particle might get moved through another triangle in the time integration in the fourth step. If this is indeed the case we could iteratively apply all the previous steps again until there is no collision anymore. But we observed that this is not necessary. Thus, if there is again a collision during this final continuous collision detection then we simply set the particle position to the barycentric coordinates of the intersection point w.r.t. the colliding triangle at the end of the time step, i.e. we stop after the third step and do not apply step four again.

The collision handling is sped up by a spatial data structure (spatial hashing as presented in [THM$^+$03]) and with several culling steps.

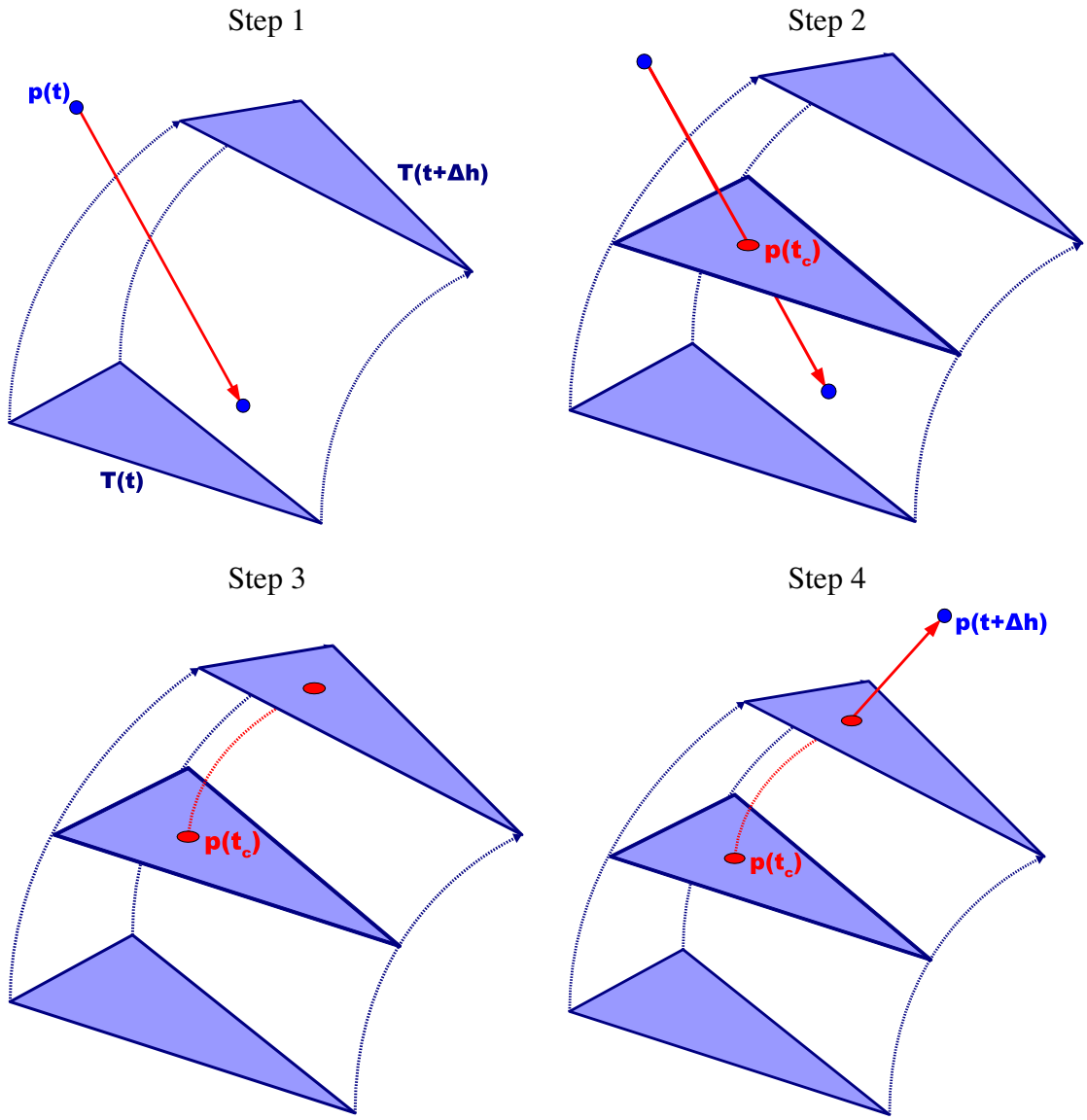


Figure 5.2: Interaction between fluid and deformable solid. First an explicit time step is performed (Step 1). Then a continuous collision detection test computes collision pairs (Step 2). As a first collision response the fluid particle is moved with the triangle to the position of the triangle at the end of the time step (Step 3). At the end the fluid particles is moved forward in time by the time between the time of collision and the end of the time step (Step 4).

6

Visualization

Even though the appearance of a fluid is a volumetric effect, we are mostly only interested in a representation of the interface between the fluid and the surrounding other objects. This representation can then be augmented with effects like refraction, reflection, etc. For meshless methods it is in general a problem how to extract and represent this interface, given a set of physical simulation nodes which sample the simulation domain. Because there is no connectivity information available it is quite difficult to decide whether a given simulation node is on or close to the surface. Further the sampling density of the simulation nodes might be not dense enough for a smooth surface. This chapter addresses this problem for the meshless fluid simulation.

6.1 Motivation for Point-Based Rendering

There exist many different methods to extract a surface from a given set of sampling nodes. Probably the best known method is the Marching Cubes algorithm [LC87]. This is a mesh based algorithm which can produce high quality results given the sampling density is high enough. Many extensions have been proposed, for example to speed up the computation using an octtree data structure. Recently, even a tutorial on how to implement marching cubes using geometry shaders appeared [MCw07]. For Eulerian simulation methods, the grid which discretizes the simulation domain and the marching cubes grid for the visualization are often the same. For particle based fluid simulations, the marching cubes algorithm requires to define a grid data structure wherever there are fluid particles. If the particles are spread over a large volume this grid might become huge. Further the values at the marching cubes grid nodes must be interpolated somehow from the particles.

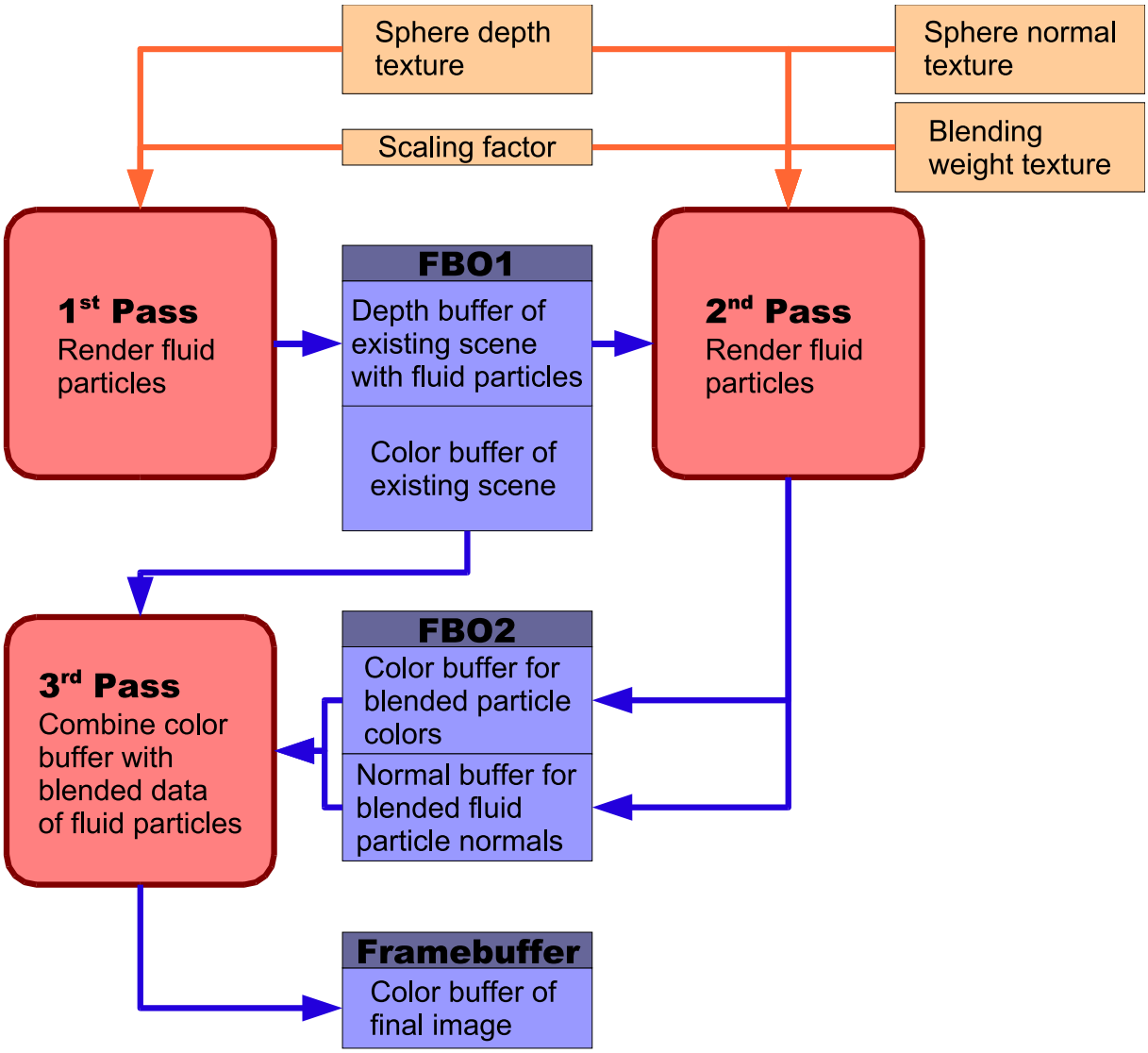


Figure 6.1: Flow diagram of fluid rendering algorithm

In contrast, point-based rendering does not require any grid data structure at all. Point-based rendering is a meshless rendering method and therefore fits nicely into the meshless simulation method of the fluid. We therefore decided to implement a point-based rendering algorithm for the SPH fluid simulation.

6.2 Fluid Rendering Algorithm

Our rendering algorithm is based on [BA07] but it is slightly modified to replace certain computations in the fragment shader by inexpensive texture accesses. The rendering algorithm is implemented in OpenGL and GLSL. The algorithm makes use of frame buffer objects which allow efficient implementations of multipass rendering algorithms. This algorithm uses three passes to render the fluid. The first pass renders the fluid particles to the depth buffer, the second

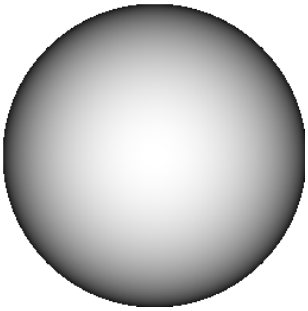


Figure 6.2: The sphere depth texture is simply the scalar z coordinate of the implicit sphere equation with a radius equal to one $z = \sqrt{1 - x^2 - y^2}$.

pass is a blending pass which blends the visible foremost particles and the third rendering pass combines the blended data with the rest of the scene. In the following subsections, each step is described in detail. A flow diagram of the algorithm is shown in Figure 6.1.

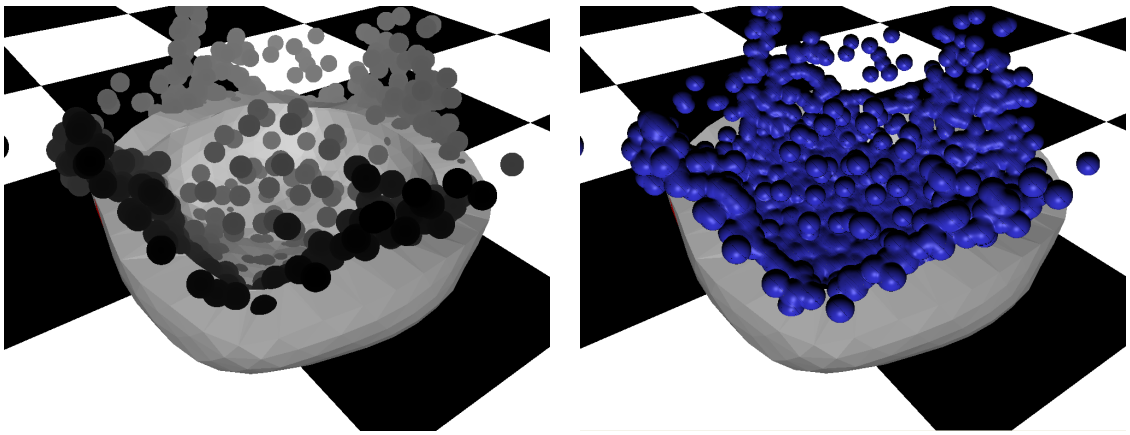
6.2.1 1st Pass

The first pass basically uses the hardware accelerated depth test to extract the visible foremost particles. Note that a fluid particle might belong to the set of foremost particles but is not visible anyway because it gets occluded by another object of the scene. Each fluid particle is rendered as a sphere. Note that for efficiency reasons such a sphere is not rendered using multiple vertices to approximate the sphere. Instead a sphere depth texture 6.2 is used to define the depth offsets between a flat surface parallel to the view plane and a sphere with a mid point in this plane. Each fluid particle is rendered as a flat quad parallel to the view plane and the depth texture is used to lookup the offset at each individual fragment of the quad. The depth texture is computed with a sphere radius equal to one. Thus a depth scaling factor is necessary to transform the unit radius from the depth texture to the proper radius of the fluid particle sphere and to perspectively scale the quad of the fluid particle.

The depth buffer render target is initialized with the depth values of the rest of the scene, such that occluded fluid particles are culled automatically. The fragments undergo the usual depth test of the OpenGL pipeline and are written to the depth buffer if they pass the depth test. The depth buffer thus contains the depth values of the *foremost* fragments of the fluid particles resp. of the already existing scene. After rendering all particles, the depth buffer is rebound as a texture which serves as an input to the second rendering pass.

6.2.2 2nd Pass

As already mentioned, the second rendering pass is a blending pass which blends the visible, foremost particles in a smooth manner. The fluid particles are rendered in exactly the same manner as in the first pass. A sphere normal texture (see Figure 6.5) is used as an additional input to provide each fragment of a fluid particle sphere with an accurate normal. The depth map which has been computed in the first rendering pass defines the z-depth of the fluid surface



Depth Buffer Final image

Figure 6.3: The output of the first rendering pass, i.e. the fragment depth values of the foremost fluid particle spheres overlaid with the color buffer of the already existing scene. The final image is shown on the right.

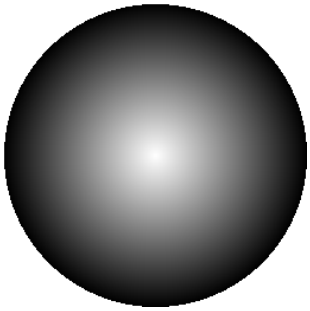


Figure 6.4: The Gaussian fall-off weight function is precomputed and stored as a texture.

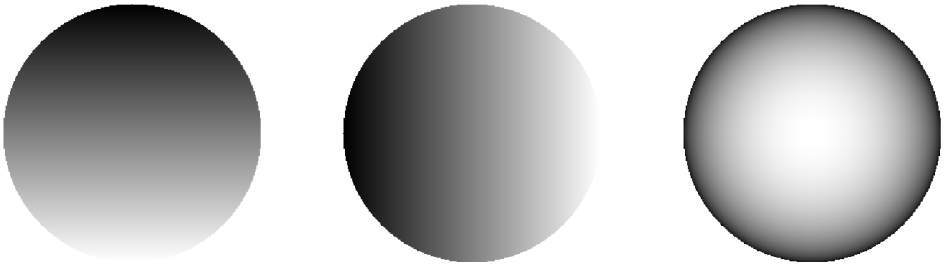


Figure 6.5: The three dimensional sphere normal texture consists of the partial derivatives in x, y and z direction of the implicit sphere equation $x^2 + y^2 + z^2 = 1$.

and is used to cull away particles whose z-depth is larger than the value in the depth map *plus* a certain threshold. This corresponds to a shift of the depth map backwards by this threshold. In this way, all the fragments within a narrow band close to the front surface pass the depth test. Additive blending is used to sum up the color and normal data of all the fragments which pass the depth test. To avoid overflows during this summation, a floating point render target is used rather than the usual render target with one byte per channel. The blending weight of each fragment is the product of two independent weight functions, similar to [BA07]. The first function ensures smooth blending of overlapping fluid particle spheres. This function defines a fall-off emerging from the projected particle position. In contrast to the linear fall-off function used in [BA07], a Gaussian fall-off function is used in the current implementation, see Figure 6.4 for a visualization of this function. Because the fall-off function is defined in the plane parallel to the view plane and the particles are rendered as flat surfaces parallel to the view plane as well, this fall-off function can be precomputed and stored in a blending weight texture to replace its evaluation in the fragment shader as done in [BA07] by an inexpensive texture lookup. As pointed out in [BA07], this weight is not enough to ensure a smooth blending of the fluid particles. For fluid particle spheres which intersect the back shifted depth buffer, the fall-off weight factor is in general non-zero and thus discontinuities appear in these regions. The second weight function has to take care of this fact. The second function fades out fragments close to the back shifted depth buffer such that it evaluates to zero when the fragment depth is equal to the back shifted value in the depth buffer. See [BA07] for the exact definition of this weight function and for further explanations.

The color and normal data of each fragment which passes the depth test is multiplied with the product of these two weight functions and then written to a frame buffer object. Because the

weights of all the fragments at a certain screen position do not sum to one, a division by the sum of all the weights at this location is required. The weight summands are accumulated in the fourth channel of the render target for the normal data and this sum is then used in the final shading pass to normalize the blended data.

Note that by computing the normals of the fluid in this way, the introduction of an additional color field, whose gradient defines the normal direction of the fluid (see [MCG03]), is no longer necessary.

6.2.3 3rd Pass

The final shading pass combines the blended data together with the color data of the already existing scene. The color buffer of the scene, the blended color and normal data of the fluid particles are all stored in separate render targets which are now rebound as input textures. A screen sized quad is sent orthographically to the OpenGL pipeline. Texture coordinates are assigned to the four vertices to map the three textures onto this quad. A sum of weights equal to zero indicates that no fragment of a fluid particle is in front of the fragment of the rest of the scene and thus this latter fragment is copied to the frame buffer. Otherwise if there is a non zero weight sum, the blended data is divided by this weight sum and written to the ordinary frame buffer.

7

Results and Further Ideas

This chapter presents some results for each major task of this thesis. In addition, further ideas are presented throughout this chapter. The reader is also invited to have a look at the movies to watch the simulations in motion.

7.1 SPH Implementation

The SPH implementation works quite well, see Figure 7.1 for several frames of a fluid colliding with a rigid pool. An explicit leap-frog time integration scheme was used to advance the SPH in time. This is a second order accurate method which requires only one force evaluation per time step which is quite important for the SPH simulation because force evaluations are quite expensive since they require summation over all neighbors.

Surely, there is still a lot of potential tuning. For example the SPH method could easily be extended to run in parallel. With the current rise of multi-core processors such an extension would probably decrease the computational time a lot. In [KLRS04] even a method is shown to compute a particle system with collision handling on graphics hardware, although they only handle rigid obstacles. Real-time physical simulation clearly tend to make use of highly parallelized hardware nowadays.

One important advantage of a CPU based implementation is that it can make use of flexible data structures to speed up the computations, especially the neighborhood queries. For example, our implementation uses spatial hashing [THM+03] to speed up the neighborhood search during the force and density computations. We even tried to store a full copy of the particle in the hash table rather than only a pointer to the particle. The idea was to avoid indirect memory accesses. But this did not lead to any noticeable speed up at all and thus this idea has been abandoned.

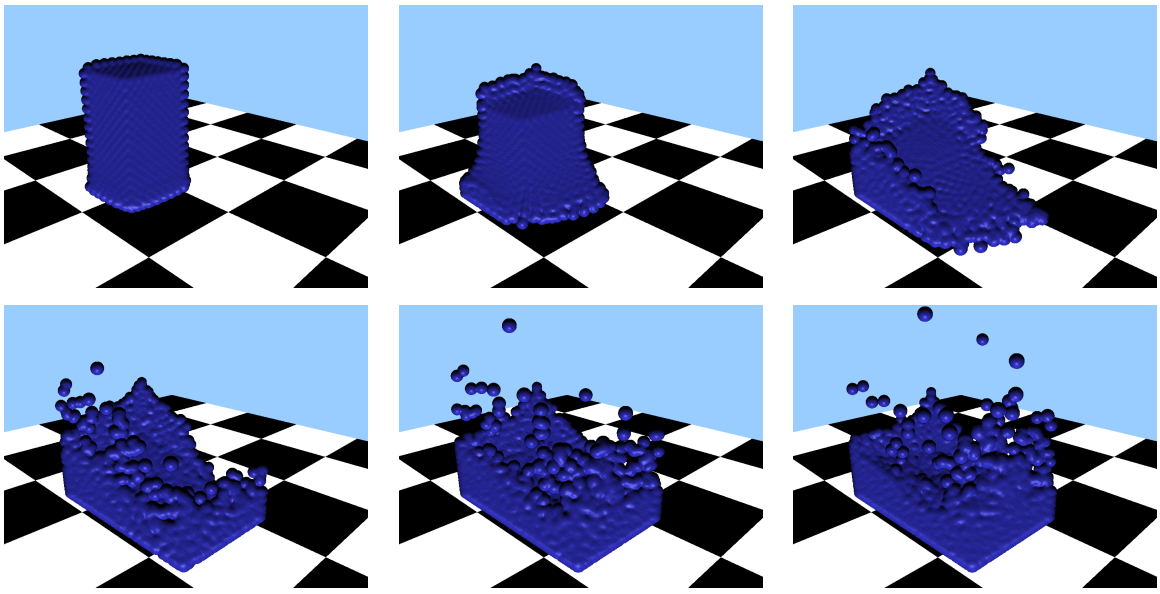


Figure 7.1: Six frames from a fluid colliding with the (invisible) walls of a rigid pool.

A SPH based fluid simulation requires a very large amount of particles for a satisfiable quality. But most of the particles do not move a lot during a simulation. For example particles near the ground of a water pool probably do not move that fast whereas particles near the upper surface are by a large part responsible for the typical appearance of a fluid surface. This observation motivates a multi-resolution approach, as has been presented in [Ada07].

Another interesting idea is to couple the SPH simulation with another fluid simulation method. The SPH method then could handle special effects like splashes or small breaking waves and the other method could simulate the large part of the fluid with a simpler approach. Specifically, the shallow water equations could be used to simulate the fluid as a height field whereas the SPH method takes care of full three dimensional effects.

7.2 Collision Handling

The presented collision handling might appear very straight forward, but a lot of subtle details must be considered. The collision handling is the glue between the two meshless simulation methods and it is not trivial not to corrupt one simulation by changing its state by the collision handling. The current collision handling algorithm allows the simulation of a fluid together with multiple deformable bodies which can collide with each other. As previously explained, only the deformable objects have an influence on the fluid, the fluid itself does not influence the deformable bodies. But interesting effects are possible even with this simplification, see Figure 7.2 for a series of frames.

The performance drops quite substantially with activated collision handling. Because there are several stages of culling, it is difficult to give exact time measurements. But for a simulation with roughly 2500 particles, the frame rate drops from about 10 frames per second to an average of 2 to 3 frames per second when all the fluid particle intersect the bounding volume of a

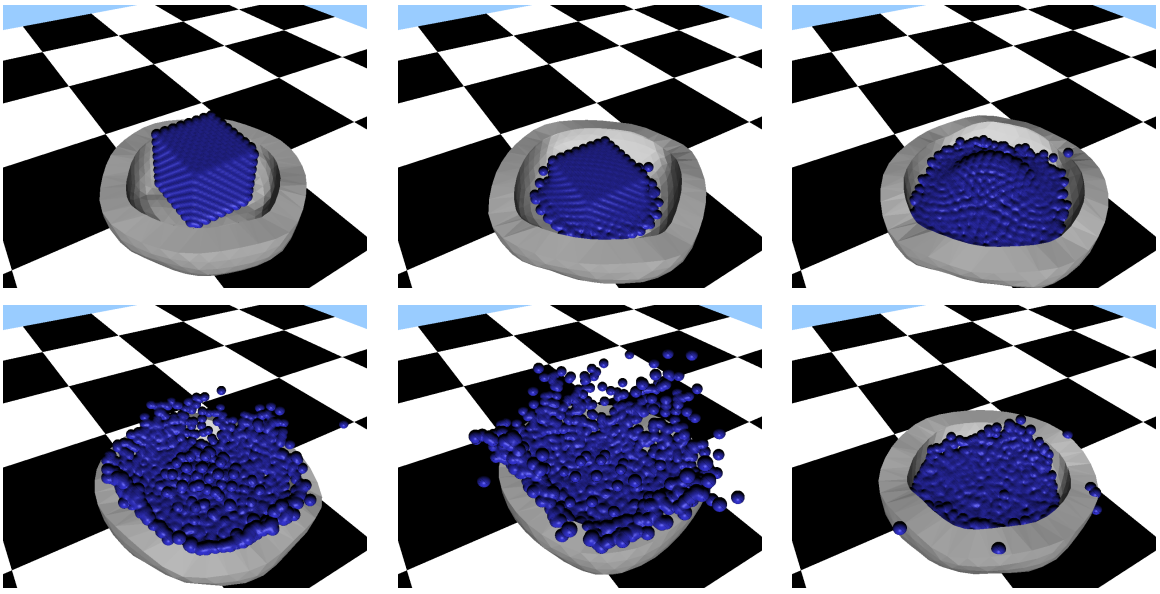


Figure 7.2: Six frames from a fluid colliding with deformable bowl which bumps on the ground plane.

deformable object. Note that these timings include the time for the simulation of the deformable solid, as well.

Unfortunately, the current implementation of the collision handling algorithm does not fully guarantee that no fluid particle is leaking through the surface, although this happened only very seldom. We suppose that a fluid particle can leak through the surface whenever it tries to cross the surface very close to an edge between two triangles. Because the intersection tests are only accurate up to a certain numerical precision, the intersection test might fail to detect an intersection at both adjacent triangles. A fix for this problem would be to slightly blow up the triangles for the intersection tests such that adjacent triangles overlap each other.

A future extension of the collision handling is clearly to implement a true two-way coupling between the fluid and the deformable solids. But the author guesses that this would require a complete redesign of the collision handling between the deformable solids as well because in the current implementation there is no control where the vertices of the deformable solid surface are going to end up after the collision response is applied at the simulation nodes.

7.3 Fluid Rendering

The fluid rendering has probably been the trickiest part to implement. The already existing framework did not yet make use of any GPU shader programs. Thus the framework first had to be extended by this capability. The major problem was caused by bad or buggy graphics drivers. The rendering algorithm made use of some very new OpenGL features and the graphics driver have not yet been up to date to support all of them. Many workarounds had to be used to implement the rendering algorithm. For example, the fluid particles were meant to be rendered as textured point sprites. This is a very efficient way to render a particle system because only

one vertex per particle needs to be sent to the graphics pipeline. The size of the viewing plane aligned points can be controlled in the vertex shader. Further OpenGL 2.0 offers the feature to automatically generate varying texture coordinates for point sprites such that a texture can be mapped onto the point sprite. This would perfectly match the requirements of the fluid rendering algorithm. Unfortunately, the existing framework could not be extended to support this feature. We had to resolve this issue by using screen aligned quad primitives which implies four vertices per fluid particle. But to decrease the rendering time, only one screen aligned template-quad with a given, fixed size is computed each frame. The final size and position of a fluid particle quad is then computed in the vertex shader by perspectively scaling the template-quad. This scale factor depends on the depth of the quad and the near, far, top and bottom clipping planes. Note that the clipping planes can be extracted from the active projection matrix.

The results of the rendering look like expected. The foremost particles are smoothly blended, as an example see Figure 7.3 for several frames. A random color has been assigned to each fluid particle in this simulation. The rendering algorithm smoothly blends these colors. Unfortunately, a closer look reveals that there are artifacts along the diagonal of the quad, specifically there are black pixels along the diagonal. This is probably an aliasing artifact because we had to use quad primitives which get triangulated internally in the graphics pipeline.

The fluid rendering is quite fast as long as the viewer does not zoom too close to the fluid. The closer one looks at the fluid, the bigger the quads and the more fragments need to be processed. Because the depth value of a fragment is changed in the fragment shader no hardware accelerated hierarchical depth culling is possible. Thus the fragment shader becomes a bottleneck and the simulation slows down when the fluid is viewed from close viewpoint. But note that this is only the case for extreme viewpoints, in general we did not observer any slowdown due to the rendering from normal viewpoints.

The fluid rendering offers a couple of improvements, too. For example all the data required to implement reflection is readily available. Note that the surface still appears quite blobby. This is a typical problem whenever the fluid surface is rendered using the particles directly. A marching cubes based rendering algorithm could solve this problem, an interesting idea has been presented recently in [MM07].

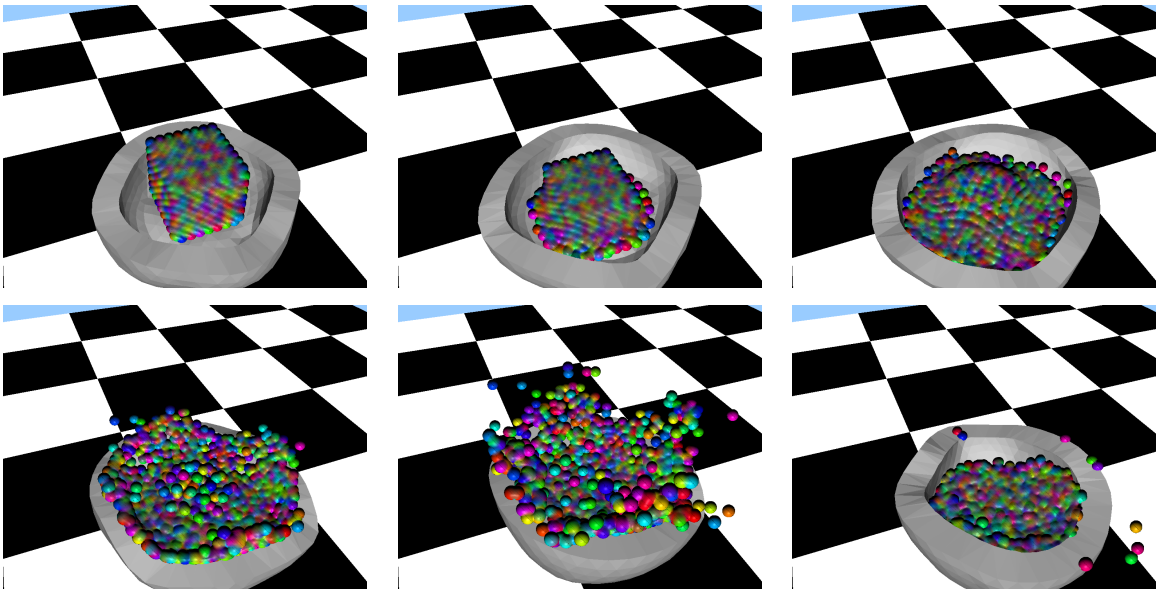


Figure 7.3: Six frames from a fluid colliding which shows the blending of colors.

Bibliography

[Ada07] Adaptively sampled particle fluids. In *ACM Siggraph 2007 (TOG)*, 2007.

[BA07] Philip Dutré Bart Adams, Toon Lenaerts. Particle splatting: Interactive rendering of particle-based simulation data. http://www.cs.kuleuven.ac.be/ graphics/CGRG.PUBLICATIONS/PSIRPBSD/, 2007.

[BFMF06] Robert Bridson, Ronald Fedkiw, and Matthias Muller-Fischer. Fluid simulation: Siggraph 2006 course notes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 1–87, New York, NY, USA, 2006. ACM Press.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 61–76, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[Fer04] Randima Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[GM77] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181:375–389, November 1977.

[HK89] L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *Astrophysical Journal Supplement Series*, 70:419–446, June 1989.

[HS89] E. Kunz H. Schade. *Stroemungslehre*. de Gruyter, 1989.

[Kei06] R. Keiser. Personal communication, 2006.

[KLRS04] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM*

SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pages 123–131, New York, NY, USA, 2004. ACM Press.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[Liu03] Gui-Rong Liu. *Mesh Free Methods, Moving beyond the Finite Element Method*. CRC Press, Boca Raton, London, New York, 2003.

[Luc77] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, December 1977.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MCw07] Opengl geometry shader marching cubes. http://www.icare3d.org/content/view/50/9/, 2007.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[MM07] S. Duthaler M. Müller, S. Schirm. Screen space meshes. In *Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA)*, 2007.

[MST+04] Matthias Müller, Simon Schirm, Matthias Teschner, Bruno Heidelberger, and Markus Gross. Interaction of fluids with deformable solids: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):159–171, 2004.

[oSTT03] McGraw-Hill Dictionary of Scientific and Inc Technical Terms, McGraw-Hill Companies. viscous dissipation function. http://www.answers.com/topic/viscous-dissipation-function, 2003.

[SBH91] Sauro Succi, Roberto Benzi, and Francisco Higuera. The lattice boltzmann equation: a new tool for computational fluid-dynamics. In *Proceedings of the NATO advanced research workshop on Lattice gas methods for PDE's : theory, applications and hardware*, pages 219–230, Amsterdam, The Netherlands, The Netherlands, 1991. North-Holland Publishing Co.

[SMVO96] P. R. Shapiro, H. Martel, J. V. Villumsen, and J. M. Owen. Adaptive Smoothed Particle Hydrodynamics, with Application to Cosmology: Methodology. *Astrophysical Journal Supplement Series*, 103:269–+, April 1996.

[SOG06] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. Fast arbitrary splitting of deforming objects. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 63–72, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[THM+03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV*, pages 47–54, 2003.