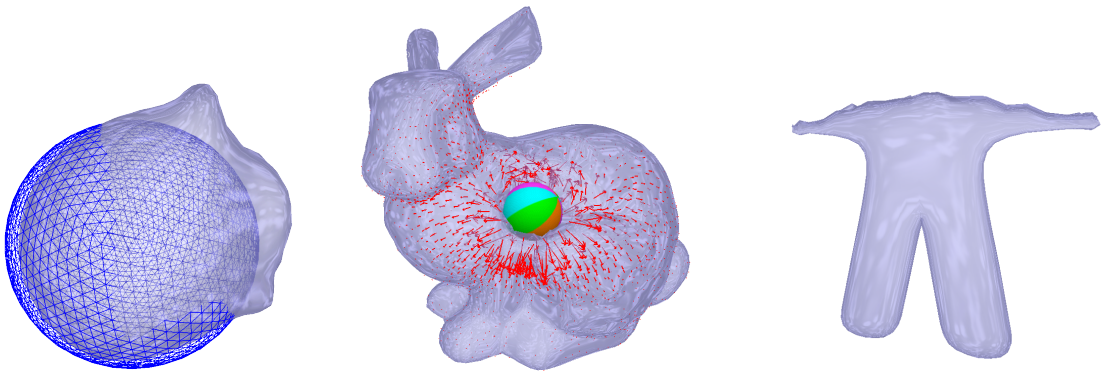
Control Algorithms for Interactively Animated Fluid Characters

Roland Angst



Master's Thesis

2007

Supervised by

Dr. Nils Thuerey
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Graphics Laboratory ETH Zurich

Abstract

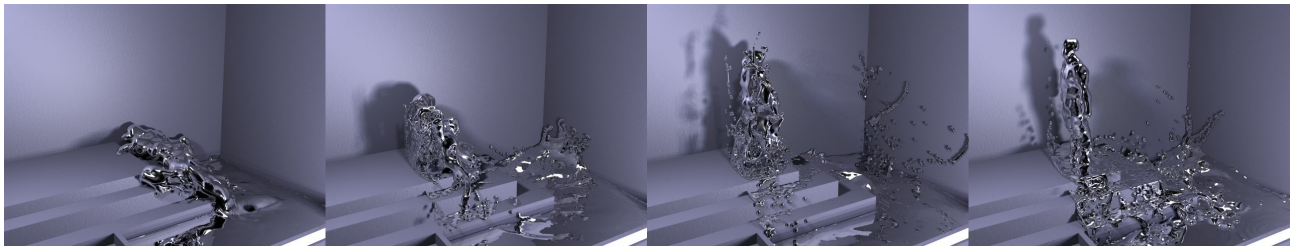
Fluid simulations have recently gained increased importance in virtual environments. Simultaneously, much progress has been achieved in controllable fluid simulations. Due to the complexity of the governing equations, real time controllable fluid simulations still remain a challenging task. In this master's thesis, we present a framework to simulate a volumetric fluid flow on curved surfaces of arbitrary topology in real time. The fluid surface is represented as a heightfield elevation of the curved surface and the fluid dynamics are captured by the shallow water equations. Moreover, we propose a novel method to mimic controllable fluid flows in real time by allow for animations of the curved surface. We exemplify the versatility of our approach with several effects such as interactions between the fluid and rigid bodies, inertia effects due to the animated control shape, and a filling process. Various test cases, such as moving and deforming characters, show that our method is highly efficient and robust.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
Computer Graphics Laboratory ETH Zurich

Master's Thesis

Control Algorithms for Animated Fluid Characters



Topic:

Fluid simulations are becoming more and more important for games and virtual environments. For computer games, animated characters made of fluid are of special interest, but were so far not possible with conventional methods.

Topic:

The goal of this thesis is to develop a method that makes the real-time animation of fluid characters possible. To achieve this, the following algorithms will be developed, evaluated and implemented:

- An SPH-simulation coupled to an animated skeleton.
- The simulation of a mesh attached to the skeleton, to capture the underlying inertia effects and represent the main volume of the fluid.
- A precomputed filling process for the body mesh.

Furthermore, the rendering method will be another focus of this thesis to ensure a closed and smooth surface display of the mesh and the SPH simulation.

Start date: 2007-03-12
End date: 2007-09-12

Remarks
A written report and an oral presentation conclude the work. The thesis is overseen by Prof. Markus Gross and is supervised by Nils Thuerey, Institute of Computational Science.

Contents

List of Figures

List of Tables

1

Introduction

Fluids are such a common day to day phenomenon that we are well accustomed to their dynamics. This raises the expectations to a fluid simulation considerably. The dynamics of fluids can be captured by a set of partial differential equations know as the Navier-Stokes equations. However, these well known equations still pose practical as well as theoretical problems. Although much progress has been achieved in the fields of computational fluid dynamics and computer animation, real-time simulation methods which can handle complex interactions between the fluid and the surrounding world are still rare. Whereas the field of computational fluid dynamics puts more emphasis on accuracy, the fields of computer graphics and computer animation are more interested in highly efficient, plausible, and visually pleasing simulations. Application areas of such simulations are surgical simulations, computer games and previews for more accurate solvers, for example. However, the real-time simulation using the Navier-Stokes equations in three dimensions is currently only possible for simple and regular boundary conditions and simulation domains.

During the last couple of years, a considerable research effort has been invested into target driven fluid simulations where a fluid flow is guided to shape according to a certain control figure while nonetheless exhibiting typical fluid behavior. The special effects industry is highly interested in such approaches because target driven fluid simulations can be very impressive on one side but unfortunately are very time consuming and difficult, if not even impossible, to animate explicitly by hand. Moreover, to the best of our knowledge, all the existing approaches are far from achieving real-time performance.

To reduce the complexity of the three dimensional Navier-Stokes equations, we propose to approximate these equations with a 2.5 dimensional heightfield representation known as the shallow water equations. This simplification allowed us to develop a framework which robustly simulates volumetric fluid flows on arbitrary control shapes in real-time and in a physically sound manner. The fluid surface is represented as an elevation w.r.t. the control shape which

in turn serves as simulation domain. Compared to previous approaches, our framework grants the fluid simulation more flexibility in that the control shape can be animated in time as well. Thanks to our general formulation of the shallow water equations, inertia effects due to the movement of the simulation domain are convincingly transferred to the simulated fluid. Furthermore, our fluid simulation handles arbitrary external forces. We exemplify this feature with a collision handling and a filling process. In a nutshell, the novel contributions of this thesis are:

- The simulation of the truly nonlinear shallow water equations on a two dimensional curved simulation domain with an exact conservation of mass.
- An animated simulation domain allows the fluid to deform and take on virtually any shape with the same topology.
- No restrictions to the discretization of the simulation domain, besides requiring a closed triangle mesh.
- A general derivation of the governing shallow water equations allows several interesting effects and interactions with the fluid, for example colliding solids and a filling process.
- A very efficient formulation of the Semi-Lagrangian advection on triangle meshes.

We hope that this thesis provides promising approaches and stimuli to enhance virtual environments with arbitrarily shaped fluid objects which can change their shape and interact with the environment in real-time. Furthermore, our approach provides animators with two means to influence the fluid flow. Firstly, the simulation domain can be animated which induces inertia effects to the fluid and secondly, on top of such an animation, an arbitrary user definable external force field can drive the flow on this surface.

This thesis is structured as follows. After presenting related work w.r.t. this thesis, we proceed by presenting the general derivation of the shallow water equations. Chapter 4 and Chapter 5 present how the shallow water equations can be discretized in space and time, resp. The next chapter presents in detail how our framework handles external forces. Furthermore, several interesting examples using such external forces are presented in this chapter, as well. This thesis then concludes in Chapter 7 by demonstrating several test cases together with a discussion of the results.

2

Related Work

Fluid simulations have been popularized in the computer graphics community by Foster and Metaxas in [FM96] and in [FM97]. In his seminal paper [Sta99], Stam proposed an unconditionally stable method so solve the Navier-Stokes equations on regular grids. In the following couple of years, his approach has been augmented in various different aspects. For example, [FSJ01] presented vorticity confinement to diminish dissipation artifacts, [FF01] showed results of interactions between a liquid and solids and the surface rendering of water was improved in [EMF02].

All the previously mentioned approaches use uniform regular grids as a simulation domain. However, such grids suffer from difficulties to handle irregularly shaped boundaries and do not take advantage of the actual spatial distribution of the fluid. Recently, many approaches have been presented which allow more flexible and adaptive simulation domains. For example, a multi-resolution approach using an octtree data structure is proposed in [LGF04]. Still, fully irregular and unstructured grids, like tetrahedral meshes, admit even more flexibility to model a simulation domain. [WBOL07] uses a tetrahedral grid, and is related to our work, since ideas are drawn from a finite volume formulation, as well. Feldman proposed to use hybrid meshes between tetrahedral and hexahedral meshes [FOK05]. The recent work in [ETK+07] shows an application of discrete differential forms to solve the Navier-Stokes equations in vorticity form on irregular meshes. Generally, the advancement in discrete differential forms [DKT06] shows promising results for the simulation of differential equations on complex domains while still preserving certain conservation invariants. Particle based simulation methods (see for example [MCG03]) employ a completely different technique to simulate fluids. In general however, large amounts of particles are required to sample a fluid volume sufficiently.

Although the three dimensional Navier-Stokes equations are nowadays even solvable in real-time [CLT07], complex interactions with such a fluid simulation are still not possible, yet. The computational burden to simulate the three dimensional Navier-Stokes equations grows

cubically w.r.t. the spatial resolution. Hence, a reduction from three to two dimension decreases the computational cost largely. Though, the two dimensional Navier-Stokes equations obviously lack a volumetric appearance. This motivates to use 2.5 dimensional heightfield approximations, like the shallow water equations. Kass and Miller introduced these equations to the computer graphics community almost 20 years ago [KM90]. This work was extended in [LvdP02] to also take the nonlinear advection into account. Both of these approaches use a regular, non-curved grid as a simulation domain.

Fluid flows on slightly more general surfaces have been investigated since quite a long time. Specifically, in meteorology, fluid flows on spheres are a well established research area to study atmospheric phenomena. It was again Stam who brought up the question of how flows would look like on surfaces of arbitrary topology [Sta03]. In this work, he presents a method to simulate the two dimensional Navier-Stokes equations in the two dimensional space of an atlas of parameterizations of a Catmull-Clark subdivision surface [Cat74]. The restrictions to the surface being a subdivision surface was removed in [SY04] by avoiding a parameterization and by solving the two dimensional Navier-Stokes equations directly on the mesh. Our approach borrows the method to handle the advection from this work. The already mentioned work [ETK+07] also shows some impressive results of two dimensional fluid flows on arbitrary surfaces. However, these approaches all simulate the two dimensional Navier-Stokes equations and hence the results are basically restricted to flowing textures which lack a volumetric impression. This problem is tackled by [WMT07], which demonstrates an approach similar to the one presented in this thesis, and which handles shallow water equations on triangulated surfaces. A heightfield with respect to the normal direction of the surface gives the fluid a volumetric look while at the same time still enjoying the reduced complexity of a two dimensional fluid simulation. However, besides presenting a novel formulation of the shallow water equations on regular grids, [WMT07] focuses on the simulation of small droplets on curved surfaces. This is in contrast to our work since we are more interested in the simulation of large bodies of water on curved surfaces. Furthermore, our method does not impose any restrictions to the surface mesh quality and does not require a remeshing step.

Another related field to our work are target driven fluid simulations. Controllable fluid simulations were introduced in [TMPS03]. Other examples are [FL04], [SY05a], [SY05b], and [TKPR06]. However, these approaches are all far from being real time algorithms. Our framework can mimic such controllable fluid behavior in real time using an animated simulation domain. Note that the recent work of Klingner [KFCO06] also used changing simulation domains, though they used volumetric, tetrahedral meshes and performed a remeshing step every time the simulation domain changed.

To conclude this section, we would like to point the interested reader to the excellent introductory course notes on fluid and height-field simulations [BMF07] and to the references therein.

3

Shallow Water Equations

This chapter presents the derivation of the shallow water equations (SWE). We augmented these well known equations with an additional term taking a spatially non constant vertical acceleration into consideration. This is useful for the handling of external forces as we will see in Chapter 6. This chapter proceeds by deriving the SWE in weak and strong form and by presenting how to extend the equations to a curved simulation domain.

3.1. Integral Form

The SWE are a specialized and simplified version of the Navier-Stokes equations. The Navier-Stokes equations describe the dynamics of a fluid much like elasticity theory describes the dynamics of deformable solids. The Navier-Stokes equations relate the dynamics of the density field, the temperature field and the velocity field of a fluid to each other. The full Navier-Stokes equations in three dimensions are a partial differential equation system with five equations and unknowns (density, velocity and temperature). The Navier-Stokes equations as well as the SWE are derived (in addition to the material specific constitutive relations) from three balance equations which prescribe three conservation laws, namely the conservation of mass, momentum and energy. A common simplification is to assume that the fluid flow is incompressible which means that the density, the viscosity coefficient, and the thermal conductivity are assumed to be uniform. This assumption has very important implications:

- The density is no longer an unknown of the equation system. Instead, the pressure becomes an unknown.
- The pressure and the velocity field are decoupled from the energy conservation. This means that we can solve for the pressure and the velocity ignoring the temperature field.

If the dynamics of the temperature field is nevertheless of interest, it can be computed later on from the velocity and pressure field.

Most graphic applications are not interested in the evolution of the temperature field and the Navier-Stokes equations are thus reduced to four equations in four unknowns. The SWE indeed assume the fluid to be incompressible. The following sections present the derivation of the SWE, for a detailed introduction to the more general Navier-Stokes equations see for example the excellent reference [Cra].

3.1.1. Balance Equations

The balance equations describe fundamental physical laws and are valid for any system described with continuum mechanics. Hence, they are the same for the Navier-Stokes equations and for the SWE. Since we assume that the fluid is incompressible, the conservation of mass and the conservation of momentum are the only balance equations required for the derivation. The continuity equation or conservation of mass reads like

$$\frac{d}{dt}\iiint_{\Omega^3}\rho dV + \iint_{\partial\Omega^3}\rho\mathbf{v}\cdot\mathbf{n}dA = 0 \tag{3.1}$$

and the conservation of momentum looks like

$$\frac{d}{dt}\iiint_{\Omega^3}\rho\mathbf{v}dV + \iint_{\partial\Omega^3}\rho\mathbf{v}\mathbf{v}^T\cdot\mathbf{n}dA\iint_{\partial\Omega^3}\sigma\cdot\mathbf{n}dA = \iiint_{\Omega^3}\mathbf{f}dV \tag{3.2}$$

where ρ is the fluid density, $\mathbf{v}$ is the fluid velocity and $\mathbf{f}$ captures sources and sinks of the fluid. This last term is also known as body force and is a force density (units of $\frac{N}{m^3}$) and the actual force acting on a given volume V is thus given by the volume integral over this force density. σ denotes the stress tensor which is the sum of the pressure p and the viscous stress tensor $\mathbf{T}$. Note that viscous forces are caused by spatial velocity differences in the fluid. We will assume that there are only small velocity variations and that the viscosity coefficient is small as well such that the viscous stress tensor can be ignored. Such a fluid is called inviscid. This assumption is further justified by observing that water-like fluids have a very small viscosity coefficient anyway and that numerical integration introduces a certain amount of damping which basically has the same effect as viscosity. Actually, in a numerical simulation it is quite difficult to get rid of this artificial viscosity.

3.1.2. Projection

The derivation of the SWE now proceeds by making an important assumption with far reaching consequences, namely the number of unknowns of the Navier-Stokes equations are reduced from four to three. The pressure is assumed to be hydrostatic, i.e.,

$$p = \rho a_n \Delta\eta$$

where a_n is the acceleration in vertical direction and $\Delta\eta$ corresponds to the vertical distance up to the fluid surface. Amongst others, the vertical acceleration takes care of the gravitational

acceleration $g = -9.81\frac{m}{s^2}$. This hydrostatic pressure equation can be derived by assuming that the vertical velocity is much smaller than the horizontal velocity such that the momentum conservation in vertical direction is dominated by the pressure term and hence the advection of the vertical velocity v_n can safely be ignored. This assumption is justified by observing that in areas where the fluid is shallow, the vertical velocity is indeed orders of magnitude smaller than the horizontal velocity components. The implications of the hydrostatic pressure assumption are summarized by the following points:

- The momentum equation for the vertical component is static, i.e., the vertical velocity is temporally constant and equal to zero.
- The velocity is basically a two dimensional tangential vector field which is orthogonal to the vertical (or normal) direction.
- The pressure p acts as the sole vertical variable.
- The fluid surface can be represented by a heightfield.

The tangential velocity can be thought of as averaged over the fluid depth. The momentum conservation equation therefore reads

$$\begin{aligned}\iiint_{\Omega^3} \begin{pmatrix} \mathbf{f}_\tau \\ f_n \end{pmatrix} dV \quad &= \quad \frac{d}{dt} \iiint_{\Omega^3} \rho \begin{pmatrix} \mathbf{v}_\tau \\ 0 \end{pmatrix} dV \\ &\quad + \iint_{\partial\Omega^3} \rho \begin{pmatrix} \mathbf{v}_\tau \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_\tau \\ 0 \end{pmatrix}^T \cdot \mathbf{n} dA + \iint_{\partial\Omega^3} p\mathbf{I} \cdot \mathbf{n} dA \end{aligned} \tag{3.3}$$

where the subscript τ indicates horizontal (or tangential) components and the subscript n denotes the vertical (or normal) component. The following notation is introduced and will be used throughout all the chapters of this thesis, see also the graphical depiction in Figure 3.1:

- H: Height of fluid between zero-level and ground.
- h: Height of fluid above zero-level.
- η: Total fluid height, i.e. $\eta = H + h$.

To derive a heightfield representation of the fluid we must get rid of the vertical dimension in these equations by a projection of the integrals to the two dimensional horizontal domain. Assume the equations are computed over a fluid column with a triangular base area Ω^2_τ, see Figure 3.1. Consider first the surface integral in the continuity equation

$$\iint_{\partial\Omega^3} \rho \mathbf{v} \cdot \mathbf{n} dA = \iint_{\Omega^2_\tau} \rho \mathbf{v} \cdot \mathbf{n} dA + \iint_{\Omega^2_n} \rho \mathbf{v} \cdot \mathbf{n} dA$$

where the surface integral was split up into two parts over the surfaces Ω^2_n vertical to the horizontal projection plane Ω^2_τ and the remaining bottom and top surface. There is no mass flux through the bottom because the bottom surface is impermeable and thus this surface integral vanishes. The slope of the fluid surface is assumed to be small. This implies that $\mathbf{v} \cdot \mathbf{n}$ is small, too and hence, the integral over the top of the fluid column is negligible. The mass flux

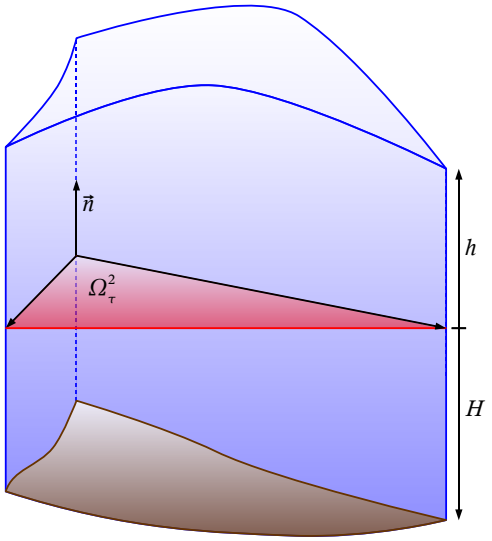


Figure 3.1.: *A fluid volume is represented as a heightfield elevation in normal direction* $\mathbf{n}$*. The forces acting on the triangular-shaped fluid column are projected to a two dimensional simulation domain* Ω^2_τ *which is visualized in red color. The fluid bottom is given at a depth of* H *below the simulation domain and is depicted in brown color. Whereas this depth* H *is temporally constant, the height displacement* h *from the simulation domain is dynamic. The total fluid depth is given by* $\eta = H + h$*.*

through the vertical sides Ω^2_n is reduced to a line integral by projecting the fluid depth onto the boundaries $\partial\Omega^2_\tau$ of the horizontal projection surface

$$\iint_{\Omega^2_n} \rho\mathbf{v}\cdot\mathbf{n}dA = \int_{\partial\Omega^2_\tau}\int_{-H}^{h} \rho\mathbf{v}\cdot\mathbf{n}d\eta ds = \int_{\partial\Omega^2_\tau} \rho(h+H)\mathbf{v}\cdot\mathbf{n}ds = \int_{\partial\Omega^2_\tau} \rho\eta\mathbf{v}\cdot\mathbf{n}ds. \quad (3.4)$$

Note that the normals of the vertical surfaces Ω^2_n of the fluid column are equal to the normals of the boundary curve of the horizontal projection surface. The projection of the momentum advection term is analog to the projection of the mass flux, i.e., the integrals over the non-vertical areas vanish due to the same reasons.

Consider now the pressure force acting on the fluid column

$$\iint_{\partial\Omega^3} p\mathbf{I}\cdot\mathbf{n}dA = \iint_{\Omega^2_\tau} p\mathbf{I}\cdot\mathbf{n}dA + \iint_{\Omega^2_n} p\mathbf{I}\cdot\mathbf{n}dA$$

where the surface integral was again split up into a horizontal and a vertical component. The surface integral over the top of the fluid surface vanishes because the pressure at this interface is negligible compared to the pressure inside the fluid. The surface integral over the bottom leads to a source term of the momentum, provided the bottom is non-flat. This pressure force makes the fluid flow downwards. This force is computed by integrating the hydrostatic pressure $p = \rho g\Delta\eta$ over the bottom area

$$\begin{aligned}\iint_{\Omega^2_\tau} p\mathbf{I}\cdot\mathbf{n}dA &= \iint_{\Omega^2_\tau} \rho a_n(-\eta)\mathbf{n}dA \\ &= \iint_{\Omega^2_\tau} \rho a_n(-H-h)(-\nabla H)dA = \iint_{\Omega^2_\tau} \rho a_n(H+h)\nabla H dA. \end{aligned} \quad (3.5)$$

Note that the normal $\mathbf{n}$ in this equation corresponds to the normal given by the negative gradient of the bottom heightfield $-\nabla H$ rather than the normal of the horizontal projection plane. Furthermore, the normalization of the gradient and the cosine for the area foreshortening cancel each other out.

The projection of the pressure force due to the vertical sides of the fluid column transforms the surface integral over these sides again into a boundary integral over the curve $\partial\Omega_\tau^2$, which bounds the horizontal area. Thus, by integrating over the fluid depth, the fluxes through the vertical sides get concentrated at the boundary curve $\partial\Omega_\tau^2$

$$\begin{aligned}\iint_{\Omega_n^2} p\mathbf{I}\cdot\mathbf{n}dA &= \int_{\partial\Omega_\tau^2}\int_{-H}^{h}\rho a_n(\eta-h)\mathbf{n}d\eta ds \\ &= -\int_{\partial\Omega_\tau^2}\frac{1}{2}\rho a_n(-H-h)^2\mathbf{n}ds = -\int_{\partial\Omega_\tau^2}\frac{1}{2}\rho a_n\eta^2\mathbf{n}ds. \end{aligned} \tag{3.6}$$

Integrating the vertical component in the volume integrals of Equation (3.1) and Equation (3.3) and inserting Equation (3.4), Equation (3.5), and Equation (3.6) the SWE can be restated in the integral form

$$\frac{d}{dt}\iint_{\Omega_\tau^2}\rho\eta dA + \int_{\partial\Omega_\tau^2}\rho\eta\mathbf{v}\cdot\mathbf{n}ds = 0 \tag{3.7}$$

$$\begin{aligned}\frac{d}{dt}\iint_{\Omega_\tau^2}\rho\eta\mathbf{v}dA + \int_{\partial\Omega_\tau^2}\rho\eta\mathbf{v}\mathbf{v}^T\cdot\mathbf{n}ds - \int_{\partial\Omega_\tau^2}\frac{1}{2}\rho a_n\eta^2\mathbf{n}ds \\ = -\iint_{\Omega_\tau^2}\rho a_n\eta\nabla H dA + \iint_{\Omega_\tau^2}\eta\mathbf{f}_\tau dA.\end{aligned} \tag{3.8}$$

From now on, the subscript τ will be omitted in the equations. In the literature, these equations are often given in the so called flux form

$$\frac{d}{dt}\iint_{\Omega^2}\mathbf{q}dA + \int_{\partial\Omega^2}F(\mathbf{q})\cdot\mathbf{n}ds = \iint_{\Omega^2}\psi dA \tag{3.9}$$

where the equations were divided by the constant density and

$$\mathbf{q} = \begin{pmatrix}\eta & \eta u & \eta v\end{pmatrix}^T$$

$$F(\mathbf{q}) = \begin{bmatrix}\mathbf{f}(\mathbf{q}) & \mathbf{g}(\mathbf{q})\end{bmatrix} = \begin{bmatrix}\eta u & \eta v \\ \eta u^2 - \frac{1}{2}a_n\eta^2 & \eta uv \\ \eta uv & \eta v^2 - \frac{1}{2}a_n\eta^2\end{bmatrix}$$

$$\psi = \begin{pmatrix}0 \\ -a_n\eta\nabla H + \frac{\eta}{\rho}\mathbf{f}_\tau\end{pmatrix}.$$

The two components of the velocity are denoted here as $\mathbf{v} = (u, v)^T$. This notation clearly shows the non-linearity and the conservative nature of the equations due to the flux across the boundary.

3.2. Differential Form

We now turn to the derivation of the differential form of the SWE, although the integral form admits more general solutions than the differential form. Solutions to the integral form may exhibit discontinuities (which must still satisfy certain conditions) like for example shocks. Such solutions are also known as weak solutions. The differential form on the other hand is written as a partial differential equation (PDE) which only admits continuous solutions because the derivation requires a continuity assumption. Solutions of the differential form are known as strong (also called pointwise) solutions. But note that the true solution of the SWE with certain boundary conditions might indeed be discontinuous and in such cases, a solution method derived from the differential form fails to converge to the correct solution. However, in computer graphics, accuracy is not the most important factor and thus, a strong solution which is similar to the true weak solution might be adequate enough.

As mentioned before, we will assume in this derivation that the solution to the SWE is continuous and differentiable. The divergence theorem can then be applied to the boundary integral of the integral form 3.7 and 3.8 and the time derivative can be moved inside the integral. We end up with

$$\iint_{\Omega^2} \mathbf{q}_t dA + \iint_{\Omega^2} \nabla \cdot \mathbf{F}(\mathbf{q}) dA = \iint_{\Omega^2} \psi dA.$$

This equation must be valid for any two dimensional domain Ω^2 and therefore the following PDE must hold

$$\mathbf{q}_t + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{q}_t + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}} \mathbf{q}_x + \frac{\partial \mathbf{g}(\mathbf{q})}{\partial \mathbf{q}} \mathbf{q}_y = \psi. \tag{3.10}$$

The chain-rule was used to expand the divergence into a product involving the Jacobian of the flux functions $\mathbf{f}$ and $\mathbf{g}$. Note that such a PDE is called hyperbolic if any linear combination of the two Jacobians $\frac{\partial \mathbf{f}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{g}}{\partial \mathbf{q}}$ has real eigenvalues and can be diagonalized. This is indeed the case for the SWE.

Written out in detail, Equation (3.10) looks like

$$\eta_t = -\nabla \cdot (\eta \mathbf{v}) \tag{3.11}$$

$$\frac{\partial \eta \mathbf{v}}{\partial dt} + \nabla \cdot (\eta \mathbf{v}\mathbf{v}^T - \mathbf{I}\frac{1}{2} a_n \eta^2) = -a_n \eta \nabla H + \frac{\eta}{\rho} \mathbf{f}_\tau. \tag{3.12}$$

This latter equation can be simplified even further. By pushing the differential operator inside the bracketed terms we get

$$\eta \mathbf{v}_t + \mathbf{v}\eta_t + \mathbf{v}\nabla \cdot (\eta \mathbf{v}) + \eta \mathbf{v} \cdot \nabla \mathbf{u} - a_n \eta \nabla \eta - \frac{1}{2}\eta^2 \nabla a_n = -a_n \eta \nabla H + \frac{\eta}{\rho} \mathbf{f}_\tau.$$

The second and third term on the left hand side correspond to the continuity equation times the velocity and thus is equal to zero. Dividing by η and observing that $H - \eta = -h$ finally gives

$$\mathbf{v}_t + \mathbf{v} \cdot \nabla \mathbf{v} - a_n \nabla h - \frac{1}{2}\eta \nabla a_n = \frac{1}{\rho} \mathbf{f}_\tau \tag{3.13}$$

The previous equation clearly highlights the advectional part of the momentum conservation. The differential equation derived from the continuity equation can also be formulated using a material derivative by observing that $\nabla \cdot (\eta \mathbf{v}) = \mathbf{v} \cdot \nabla \eta + \eta \nabla \cdot \mathbf{v}$. The SWE then take the following form

$$\frac{D\eta}{Dt} = -\eta \nabla \cdot \mathbf{v} \tag{3.14}$$

$$\frac{D\mathbf{v}}{Dt} = a_n \nabla h + \frac{1}{2}\eta \nabla a_n + \frac{1}{\rho}\mathbf{f}_\tau. \tag{3.15}$$

Note that when the balance equations were projected from three dimension to two dimensions, the vertical force component has been dropped. This term can now be reinserted as a vertical acceleration component, for example in addition to a constant acceleration due to gravity $g = -9.81\frac{m}{s^2}$:

$$a_n = g + \frac{f_n}{\rho}$$

Thanks to these external force terms $\mathbf{f}_\tau$ and f_n, the SWE can more easily interact with other objects. Note that the SWE are usually derived without these terms.

3.3. SWE on Curved Manifolds

The derivation of the SWE has been purposely presented in a quite general formulation. The only condition to the embedding space of the simulation domain was to be separable into a horizontal and a vertical component such that the projection step of Subsection 3.1.2 is applicable. Usually, the SWE are simulated on a planar simulation domain embedded in a three dimensional Euclidean space, i.e., the horizontal (or tangential) component can be assumed to correspond to the (x,z)-plane and the vertical dimension is given by the direction of the y-axes. But note that any k-dimensional manifold embedded in n-dimensional Euclidean space allows a decomposition into a tangential and vertical component. The neighborhood at any point on the manifold is homeomorphic to an open k-dimensional disc and thus a tangent plane is defined at any point of the manifold. The vertical component at a point is then simply given by the dimensions orthogonal to the tangent plane.

In this thesis, the SWE are applied to two dimensional manifolds embedded in three dimensional Euclidean space, similar to [WMT07]. Therefore, the SWE are simulated on a curved surface rather than on a flat plane. The fluid depth η is defined w.r.t. the normal direction at a point, whereas the velocity $\mathbf{v}$ is defined in the local tangent plane at a point. The constant gravity was chosen to always point in the same direction w.r.t. the local reference system, i.e., the constant gravity always points in negative normal direction. A comparison with [WMT07] is deferred to Chapter 7.

Although the theoretical extension to a curved simulation domain is very simple, a practical implementation of the SWE on curved surfaces involves several changes compared to a SWE simulation on flat simulation domains. The next two chapters present how such an implementation can be achieved.

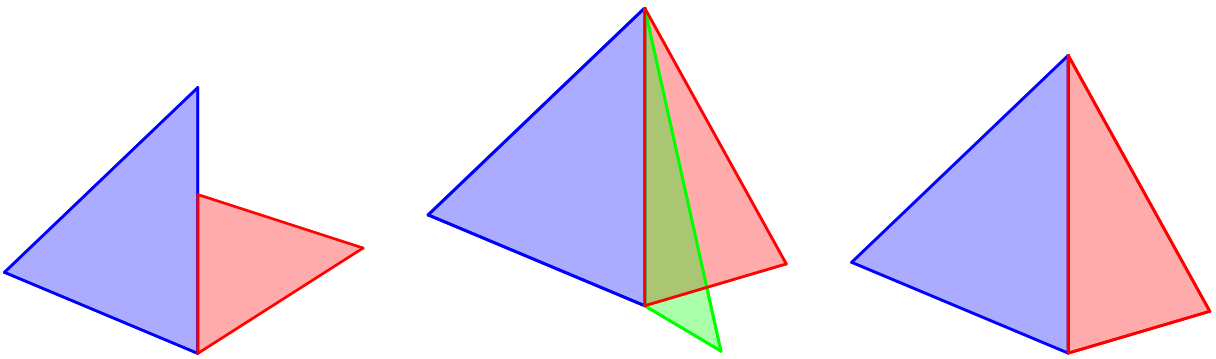
4

Spatial Discretization

There are many different ways to discretize the time evolution of a physical system which is described by a set of equations. A discretization can be broken down into a temporal and a spatial discretization. Although we present the spatial and temporal discretization each in its own chapter, such a clear separation is not advisable in practice because the stability and accuracy of a scheme depends on both discretizations, spatial and temporal. This chapter puts more emphasis in where and how to store the system variables η and $\mathbf{v}$ of the SWE and how to represent the discrete counterparts of continuous differential operators. In the next chapter, the temporal dimension is added to come up with a complete discretization of the SWE.

A vast literature exists about the discretization of PDEs. In the case of the SWE, especially a lot of research has been done and is still done in ocean and climate modeling. Ocean and climate modeling both put more emphasis on accurate solutions rather than efficiency of the simulation method. Nevertheless, these fields offer interesting approaches. They can be adapted to trade accuracy for efficiency which is more important for physical simulations in computer graphics. The following list is given to recall the most prominent methods for the simulation of fluids, for a more comprehensive review of current numerical methods for fluid simulations we refer to [LMW02] which presents spatial and temporal discretization methods for the Navier-Stokes equations.

- **Finite Difference Method (FDM)**: Only applicable on simple geometries but allow very efficient code. Straightforward and very easy to implement.
- **Finite Element Method (FEM)**: Usable only in low velocity conditions but arbitrary complex geometries are possible. FEMs are based on the differential form.
- **Finite Volume Method (FVM)**: Most industrial computational fluid solvers are based on a finite volume approach. They allow arbitrary complex geometries, as well. FVMs are more general than FEMs because they are based on the integral form.



(a) *no simplicial complex and no discrete manifold*

(b) *simplicial complex but no discrete manifold*

(c) *simplicial complex and discrete manifold*

Figure 4.1.: *Discretization of a 2-manifold.*

The complex simulation domain of our SWE simulation rules out finite difference methods. Hence, finite element and finite volume methods are the only possibilities. We implemented and investigated a finite volume as well as a finite element based approach. Both discretizations are presented in this chapter and the final decision which of them is more suitable for our intentions is motivated and explained.

This chapter proceeds by first presenting the discretization of the simulation domain. After that, some important points about the discretization of physical equations are mentioned. Afterwards, the finite volume method is described in more details. Finally, we present the spatial discretization of the SWE, which produced the most pleasing results.

4.1. Spatial Discretization of Simulation Domain

The goal of this thesis is the simulation of the SWE on a two dimensional manifold which is embedded in three dimensional Euclidean space. The discretization of such a manifold requires the notion of a simplicial complex which in turn requires the notion of a simplex. A *k-simplex* is defined as the non-degenerate convex hull of $k+1$ geometrically distinct points in n-dimensional Euclidean space, where $n \geq k$. By convention, these points are called vertices. A *simplicial complex* K in R^n is a collection of simplices in $\mathbb{R}^n$ such that every face of a simplex of K is in K and the intersection of any two simplices of K is a face of each of them [Mun93]. Using these definitions, a discretization of a k-dimensional manifold is a k-dimensional simplicial complex which fulfills the following condition. For every simplex in the simplicial complex, the union of all the incident k-simplices forms a k-dimensional ball (or half a ball if the simplex is a boundary simplex). This implies that every non-boundary $(n-1)$-simplex has *exactly* two adjacent n-simplices. Intuitively, this means that a two dimensional manifold in 3D is represented as a proper triangle mesh, i.e., no triangle soup. See also Figure 4.1 for a graphical explanation. To facilitate the implementation, the mesh is required to be closed which means that there are no boundary simplices. But note that our simulation could be extended to handle boundaries as well.

This discretization of the simulation domain is very general, for example unstructured and badly

shaped triangulations are allowed. This is in contrast to most existing simulation methods for the SWE which require that there are no badly shaped triangles in the discretization of the simulation domain. Because we intend to animate the simulation domain itself as well, such obtuse triangles might emerge during the animation sequence of the simulation domain. Hence, it is important to design a spatial and temporal discretization which can handle any quality of triangulation.

4.2. Discretization of Governing Equations

After the discretization of the simulation domain, the equations which govern the physical system must be discretized. This involves the discretization of the field variables of the equations as well as the involved operators. A proper spatial discretization is crucial for the stability of the method and for the visual quality of the simulation. This section first presents some general thoughts concerning the discretization of a physical system. Afterwards, the idea of discrete exterior calculus (DEC) is presented. DEC provides a consistent framework for the discretization of differential and integral equations. Although we were not able to exclusively use concepts from DEC, it nonetheless generalizes and relates certain concepts from FDM, FEM and FVM which justifies to give a short introduction into DEC.

4.2.1. General Considerations

A numerical discretization scheme should fulfill the following conditions:

- **Consistency**: When applied to any smooth function, the numerical discretization scheme of an operator should approach the true continuous operator if the resolution of the discretization grows to infinity.
- **Convergence**: The numerical solution should be equal to the true solution for any initial conditions when the resolution of the discretization grows to infinity.
- **Stability**: No amplification of numerical errors which arise due to truncation errors of the discretized equations.
- **Conservation**: No artificial sources or sinks ensure the conservative nature of a physical system, for example the conservation of mass or momentum.
- **Boundedness**: No meaningless values for physical quantities, for example no negative densities.

It is a very difficult task to come up with a proper spatial discretization, especially for unstructured, non planar simulation grids. Specifically, boundedness of the water height η is quite difficult to ensure for the SWE because they do not prescribe the positivity of the water height. The stability of a numerical method is strongly coupled with the time integration method. A discussion of the stability of our discretization is therefore deferred to the next chapter.

The major problem of finding an appropriate spatial discretization stems from the fact that certain physical quantities should be conserved exactly, as described by the balance equations.

Classical methods (like the FDM and the FEM to a certain extent) discretize the differential operators locally but most often do not respect global invariants, for example the conservation of mass is often not met exactly. Even the often used staggered grid discretization on flat regular grids introduces spurious effects which depend on the orientation of the grid. This is due to the fact that the numerical diffusion is much lower for flows in axis-aligned direction. Fortunately, computer graphics only requires plausible results and is thus quite forgiving when the true solution is not exactly met. Hence, accuracy can be traded for stability. For example, we observed that the conservation of momentum is not as important for a pleasing result as the conservation of mass. A violation of the conservation of mass is more easily spotted by an observer than a violation of the momentum conservation. This might be explained by the fact that momentum can be consumed by non-conservative forces like friction. Mass on the other hand is always conserved. Due to this observation, we put more attention to the conservation of mass than to the conservation of momentum.

Another important requirement to a spatial discretization scheme is that the discrete operators should act like their continuous counterparts. This means that the discretization should be consistent, for example the discrete divergence of a discrete gradient should correspond to the discrete Laplacian. We will proof later on that our discretization indeed fulfills this property. If this requirement is not met, spurious effects might emerge.

4.2.2. Discrete Exterior Calculus

Discrete Exterior Calculus (DEC) is a new emerging field which provides a consistent approach for the spatial discretization of physical quantities on one side and operators acting on them on the other side. DEC is the discrete extension to the exterior calculus on differential forms which was introduced by Cartan [Car45]. Exterior calculus allows to formulate differential operators like gradient, divergence and curl and the theorems of Green, Gauss and Stokes in a consistent way using differential forms and the exterior derivative which is defined as an operator acting on these differential forms. Therefore, differential and integral equations on smooth and curved spaces can be expressed in a manner which clearly respects the geometrical properties of the space. This geometrical intuition is very important for a proper spatial discretization.

Remark that in nature, a physical quantity is most often not measurable as a pointwise value but rather as an average over a volume, may it be line segment, an area element or even a space-time volume. This observation motivates to associate a physical quantity with an appropriate volume element of the simulation domain such that no global or local coordinate frame is necessary (unlike the FVM method which defines a local coordinate system at every cell interface). This intrinsic representation of the physical quantities should get manipulated by geometrically motivated operators which are provided by the exterior calculus.

DEC maps these concepts to the discrete setting in which the simulation domain is represented by a simplicial complex. A simplicial complex implicitly defines the graph of a dual mesh. The simplicial complex is also known as the primal mesh in this context. Each primal k-simplex is associated with a dual $(n-k)$-cell. The vertices of the dual mesh are thus the centers of the cells of the primal simplicial complex. The connectivity of the dual mesh is inferred from the adjacency information of the primal mesh. See Figure 4.2 for a graphical depiction. Note that in general, the dual mesh is not a simplicial complex. A physical quantity is associated with a

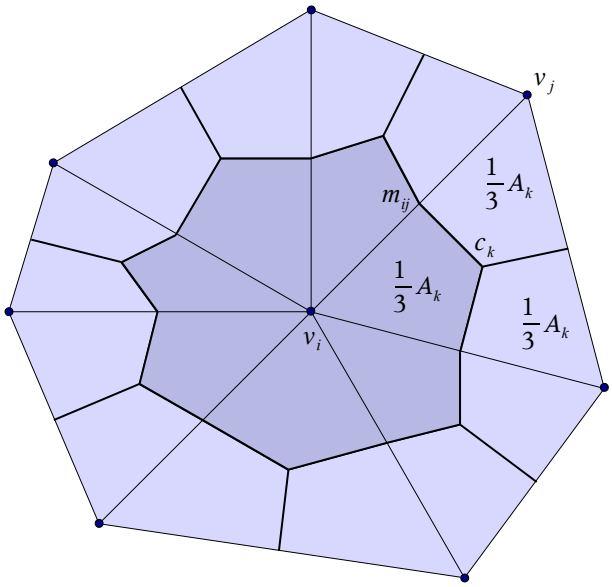


Figure 4.2.: *1-ring neighborhood of a primal vertex v_i and its corresponding dual cell (visualized in blue color). Vertices of the dual mesh are given by the triangle circumcenters or the barycenters c_k and the edge midpoints m_{ij} of the primal mesh. This diagram shows the barycentric scheme. Each primal vertex is associated to a dual cell. Hence, there are as many dual cells as there are primal vertices.*

specific element from either the primal or the dual mesh. Values at vertices, edges, faces and tetrahedra correspond to a discrete version of a pointwise function, line integral, surface integral or volume integral, respectively. Such an association is not always obvious because, rather than the original physical variables of the system, sometimes quantities, which are derived from the original variables by applying an operator, are associated and stored at a mesh element. Thanks to these associations of physical quantities with mesh elements, physical conservation laws can more easily be ensured in the discrete setting. Hence, the concepts of DEC are very well suited for unstructured, non-planar simulation domains. See [DKT06], [GDP+06] and [DFM07] for a deeper introduction into the concepts of DEC or [ETK+07] for an example of how DEC can be applied to simulate the Navier-Stokes equations in vorticity form.

There are basically two options to define the location of a dual vertex. Either the circumcenter or the barycenter of the primal cell can be used. The circumcentric representation has some advantages over the barycentric. Consider the flux of the velocity in Equation (3.14). DEC tells to associate the flux with its corresponding geometrical element, which obviously is the interface between a cell, i.e., an edge in the case of a discretized 2-dimensional manifold. Rather than storing the full velocity, only the normal component of the velocity w.r.t. the edge is stored. Hence, directional derivatives must be used in Equation (3.15) rather than gradients. Given that the water depth η is stored cell based at the circumcenters, a simple finite difference scheme can be used to compute the directional derivative of the water depth at the edge interface between triangle i and j

$$\nabla\eta \cdot \mathbf{n}_{ij} = \frac{\partial\eta}{\partial\mathbf{n}_{ij}} \approx \frac{\eta_j - \eta_i}{\|\mathbf{c}_j - \mathbf{c}_i\|}.$$

It is very interesting to point out that this method boils down to the staggered C-grid [AL77] representation on regular planar meshes. Therefore, the flux representation in DEC generalizes

the staggered grid representation to unstructured grids. Unfortunately, the circumcenter is not guaranteed to lie in the triangle for arbitrarily shaped triangles. Thus, triangulations with obtuse triangles rule out a circumcentric based scheme. In contrast, the barycenter is always guaranteed to lie in the triangle. But a barycentric centered scheme requires a more complex and time consuming reconstruction of the directional derivative because the edge normal $\mathbf{n}_{ij}$ is not parallel to the line connecting the dual vertices $\mathbf{c}_j - \mathbf{c}_i$. See [HKSP07] how this can be done in the context of the SWE and for a comparison between a circumcentric and a barycentric centered scheme for the SWE. Note that, although quite uncommon, there also exist methods which associate a full velocity vector at each edge of the simplicial complex to solve this problem [WBOL07].

As a drawback, DEC does not provide a solution to advection problems. A stable method to solve the advection is the semi-Lagrangian advection which is presented in Section 5.2. But this method requires that the velocity vector field is available. Hence, the velocity field has to be reconstructed from the available fluxes stored at the edges. For a divergence free velocity field, a unique constant velocity vector is determined by the three edge fluxes of a triangle because the absence of divergence makes the edge fluxes dependent on each other. But the tangential velocity field in the SWE is not divergence free and thus the velocity vector would have to be recovered in the least squares sense, for example. This observation, in addition to the requirement of a barycentric centered scheme, convinced us to depart from the DEC concept for the velocity representation. We define the velocity field as a piecewise constant vector field over the triangles of the primal mesh. A more detailed presentation is deferred to section Section 4.4.

To finish this digression, we would like to point the interested reader to [MW01]. This work presents some current developments on the field of discrete differential forms coupled with variational integrators. This is very interesting because the combination of the concepts of DEC with variational integrators allows to perfectly preserve physical quantities even through time.

4.3. Finite Volume

The finite volume method is applicable to equations in the flux form (3.9). As stated previously, this allows discontinuities in the solution and thus the finite volume method is more general than a scheme based on the differential form of the SWE. This section first presents the basic idea of the finite volume method. Afterwards, Godunov's method is described which is a widely used finite volume method. Finally, the implemented finite volume algorithm is presented. Note that this finite volume approach is based on Godunov's method.

4.3.1. Derivation

The basic idea of the finite volume method is to store averaged quantities over a volume rather than storing point-wise sampled values at the simulation nodes. Thus, the variable $\mathbf{q}_k^n(\mathbf{x})$ of the flux form (3.9) of the SWE at time t^n for a point $\mathbf{x}$ in triangle k is averaged over the triangle volume

$$\mathbf{q}_k^n = \frac{1}{A_k} \iint_{A_k} \mathbf{q}_k^n(\mathbf{x}) dA.$$

The volume of the triangle obviously corresponds to its area A_k. The flux form (3.9) is now integrated in time over the interval $[t^n, t^{n+1}]$ which gives

$$\mathbf{q}_k^{n+1} - \mathbf{q}_k^n + \int_{t^n}^{t^{n+1}} \frac{1}{A_k} \int_{\partial A_k} \mathbf{F}(\mathbf{q}) \cdot \mathbf{n} ds dt = 0 \tag{4.1}$$

Note that for simplicity the source term ψ on the right hand side was dropped. But the derivation can also be done including this term. It is remarkable that this equation still holds exactly, i.e., there is no numerical error introduced, yet. An actual discretization is defined by the choice of a numerical flux function which approximates the time-averaged fluxes $\mathbf{F}_k^{n+1}$ over the boundary of a volume

$$\mathbf{F}_k^{n+1} \approx \int_{t^n}^{t^{n+1}} \frac{1}{A_k} \int_{\partial A_k} \mathbf{F}(\mathbf{q}) \cdot \mathbf{n} ds dt. \tag{4.2}$$

This formulation clearly shows that the finite volume method is characterized by a numerical scheme to compute the flux over the boundary of the control volume. If Equation (4.2) holds with equality then Equation (4.1) is exactly met by the true solution at some point in the time interval. A clear advantage of the finite volume formulation is that it is inherently a conservative numerical method, i.e., the physical quantity $\mathbf{q}$ is conserved.

Let us introduce the notation $\mathbf{F}(\mathbf{q}_i^n, \mathbf{q}_j^n, \mathbf{n}_{ij})$ for the normal component of the flux $\mathbf{F}(\mathbf{q}) \cdot \mathbf{n}_{ij}$ across the edge $\mathbf{e}_{ij}$ between two adjacent triangles i and j at time t^n. The normal $\mathbf{n}_{ij}$ denotes the edge normal pointing from triangle i to triangle j (hence: $\mathbf{n}_{ij} = -\mathbf{n}_{ji}$). Then the boundary integral in Equation (4.2) can be restated as

$$\int_{\partial A_k} \mathbf{F}(\mathbf{q}) \cdot \mathbf{n} ds = \sum_{j \in \mathcal{N}(k)} \|\mathbf{e}_{kj}\| \mathbf{F}(\mathbf{q}_k^n, \mathbf{q}_j^n, \mathbf{n}_{kj}).$$

The time integration in Equation (4.2) can be approximated by any method of choice, for example an explicit Euler step. Note that for the SWE, the flux is a nonlinear function of the water depth and water velocity.

4.3.2. Godunov's Method

Godunov introduced his method in 1959 and many fluid simulation methods are based on his ideas. Recall that the differential form allowed a formulation (3.10) which can be interpreted as an advection of the physical quantity $\mathbf{q}$. The eigenvalues of the Jacobian correspond to the characteristic speeds of the advection. A traditional method to solve an advection equation is the method of characteristics which traces characteristic curves backward in time. But the non-linearity of the flux function of the SWE imposes severe problems to the method of characteristics, for example non-unique solutions due to over crossings of characteristic curves. Rather than using the method of characteristic to trace the physical quantities backwards in time, Godunov proposed to compute the numerical flux in the previously derived finite volume formulation by solving a differential equation at every cell interface exactly. The solution is assumed to be a piecewise constant function over the finite volume cells. Therefore, the flux across the interface edge is completely described by a Riemann problem. For simplicity, the

method is presented for the one dimensional case. An extension to multiple dimension is possible and involves to express the physical variables $\mathbf{q}$ in the eigenvector basis of the Jacobian. But this becomes much more complex, for example see [EKH02] for a summary on Riemann solvers.

A Riemann problem is a conservation equation with piecewise constant initial data. Assume that two finite volume cells are given with a boundary interface at x_0 between them. Then the Riemann problem is given by the partial differential equation

$$q_t + vq_x = 0$$

with initial conditions

$$q(x, t^n) = \begin{cases} q_l & x < x_0 \\ q_r & x > x_0 \end{cases}$$

where q_l and q_r denote the values at the left resp. the right cell. Such a Riemann problem corresponds to a discontinuity which travels with a characteristic speed of v across the interface. The solution to this problem is

$$q(x, t) = \begin{cases} q_l & x < x_0 + v(t - t^n) \\ q_r & x > x_0 + v(t - t^n) \end{cases}$$

as can easily be verified. The numerical flux $\mathbf{F}(\mathbf{q}_i^n, \mathbf{q}_j^n, \mathbf{n}_{ij})$ is then computed based on this solution. Such a Riemann problem has to be solved at each time step for each cell interface. For the SWE, the velocity v is not constant and depends on the unknown solution of the Riemann problem. The Riemann problem can then only be solved approximately.

A disadvantage of Godunov's method is the restriction on the allowable time step for the time integration. The Riemann problems are only solvable exactly over a short time interval because in general at some point in time, the characteristics between two neighboring Riemann problems intersect each other. At this point, the solution is no longer constant in the cell. By observing that the wave speeds are bounded by the eigenvalues $\lambda_i(\mathbf{q})$ of the Jacobian, an upper bound for the allowable time step is given by

$$\max_{i,k} |\lambda_i(\mathbf{q_k})| \leq \frac{\Delta x_k}{\Delta t}$$

where $\lambda_i(\mathbf{q_k})$ denote the eigenvalues of the Jacobian at a cell interface k and Δx_k denotes the spacing between the two adjacent cells of the cell interface. This quantity is also known as the *Courant number*. A single, very small finite volume cell might thus impose severe restrictions on the time integration step.

4.3.3. Kinetic Formulation

From the many available finite volume methods, we chose to implement the approach of [AB05]. The method [AB05] presents a finite volume method for irregular but planar triangulations. But the extension to curved simplicial complexes is straight forward. The only difference is that the

normal $\mathbf{n}_{ij}$ of an interface between triangle i and j has to be understood rotated to the respective plane which is defined by the adjacent triangle cell. Usually, the solution of a two dimensional Riemann problem involves the analysis of the local Jacobian due to the non-linearity of the flux function of the SWE. This analysis is computationally rather costly, we refer to [AC97] for an example.

In contrast, [AB05] introduces a kinetic transformation which transforms the nonlinear advection equations to a linear transport problem of a non-linear quantity M called the Gibbs equilibrium

$$M(t, \mathbf{x}, \xi) = \frac{\eta(\mathbf{x}, t)}{\tilde{c}} \chi(\frac{\xi - \mathbf{u}(t, \mathbf{x})}{\tilde{c}})$$

where ξ is a kinetic velocity, $\tilde{c}$ is defined as $\tilde{c}^2 = \frac{1}{2} g\eta$ and χ is a function $\mathbb{R}^2 \rightarrow \mathbb{R}$ fulfilling following conditions:

- χ is even: $\forall \mathbf{w} \in \mathbb{R}^2 : \chi(\mathbf{w}) = \chi(-\mathbf{w})$
- χ is non-negative: $\forall \mathbf{w} \in \mathbb{R}^2 : \chi(\mathbf{w}) \geq 0$
- χ is compactly supported: $\exists w_m \in \mathbb{R} : \|\mathbf{w}\| \geq w_m \rightarrow \chi(\mathbf{w}) = 0$
- χ is a partition of unity and fulfills the Kronecker property:

$$\int_{\mathbb{R}^2} \begin{pmatrix} 1 \\ w_i w_j \end{pmatrix} \chi(\mathbf{w}) d\mathbf{w} = \begin{pmatrix} 1 \\ \delta_{ij} \end{pmatrix}$$

Based on the definition of the Gibbs equilibrium M and the assumptions on χ, it is easily profable that the kinetic quantity M can be transformed to the original physical variable $\mathbf{q}$ and the lower part of the flux function $\mathbf{F}(\mathbf{q})$:

$$\begin{pmatrix} \mathbf{q}(\mathbf{x}, t) \\ \eta \mathbf{v}\mathbf{v}^T - \frac{1}{2} a_n \eta^2 \mathbf{I} \end{pmatrix} = \int_{\mathbb{R}^2} \begin{pmatrix} 1 \\ \xi \\ \xi\xi^T \end{pmatrix} M(t, \mathbf{x}, \xi) d\xi \tag{4.3}$$

The linear transport equation, which is also known as *kinetic equation*, is given by

$$M_t + \xi \cdot \nabla M = 0. \tag{4.4}$$

The original flux formulation is revealed by an integration of Equation (4.4) against $(1, \xi)^T$ and using Equation (4.3). The advantage of the kinetic formulation is that the linear transport Equation (4.4) is easier to integrate in time than the non-linear SWE. An integration against $(1, \xi)^T$ then returns the original variables.

For a detailed description of the numerical implementation and the specific choice of the function χ we refer to the paper [AB05]. But it is worth to point out that the resulting formula for the numerical fluxes $\mathbf{F}(\mathbf{q}_i^n, \mathbf{q}_j^n, \mathbf{n}_{ij})$ is no longer dependent on the kinetic velocity ξ because there is an analytic solution to the integration against $(1, \xi)^T$ such that the kinetic velocity is integrated

out. In contrast to a more classical analysis of the local Jacobian matrix to derive the numerical fluxes, the resulting formula of the kinetic formulation is computationally very efficient.

Although a first and second order accurate finite volume method based on the kinetic formulation is presented in [AB05], only the first order accurate method was implemented. The authors of [AB05] also present an upper bound to the admissible time step for their method. We observed that in practice, the time step must meet this requirement rather strictly, i.e., the simulation diverged whenever the time step was slightly larger than this upper bound. Unfortunately, this upper bound was too restrictive due to badly shaped triangles which can often occur in an animated mesh. The upper bound forced the simulation to run with a time step which was too small to be practically useful in a real-time simulation. In addition, the results, which we achieved with this method, were less pleasing compared to the mixed FEM-FVM method which will be described in Section 4.4. Hence, the classical finite volume method was abandoned in favor of a FEM-FVM approach.

4.4. Mixed Finite Elements and Finite Volumes

We experimented a lot with different discretizations. This section presents the discretization which gave the best results and which we finally chose. Our discretization is based on a finite element formulation of scalar and vector fields which, for example, has already been used in [PP02] and [TLHD03], in a different context, though. It is worth to point out that previous approaches to simulate the SWE in computer graphics do not use such a formulation since they reformulate the SWE into a second order partial differential equation w.r.t. the water height and no longer carry along a tangential velocity field.

The spatial discretization involves two main tasks. The physical quantities which determine the state of the system must be discretized first. Then, consistent discrete operators acting on this discretization must be defined.

4.4.1. Discretization of State Variables

The state of a shallow water system is prescribed by a velocity vector field and a scalar field of the water depth. We found that the most efficient, accurate and stable discretization defines vector fields as a piecewise constant vector field over the triangle faces and scalar fields as a piecewise linear scalar field on the vertices. Given a sampling f_i of a scalar function $f : \mathbb{R}^3 \to \mathbb{R}$ at the vertices of the simulation mesh. Then the discrete scalar field is the piecewise linear function

$$\hat{f}(\mathbf{x}) = \sum_{v_i \in \mathcal{V}} \phi_i(\mathbf{x}) f_i.$$

The piecewise linear basis functions are denoted ϕ and they evaluate to 1 at the vertex v_i and to 0 at all the other vertices, see also Figure 4.3. This discretization basically corresponds to a FEM approach with piecewise linear basis functions. This approach also bears resemblance to the staggered grid approach on regular grids in that the physical quantities are stored at

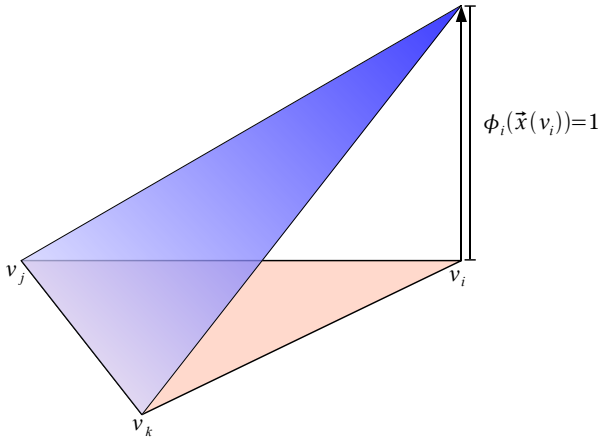


Figure 4.3.: *Linear basis function visualized as an elevation of the triangle: A linear basis function is assigned to each vertex. This function evaluates to* 1 *at its vertex and to* 0 *at all other vertices.*

different places. This discretization does not fully conform to the concepts of DEC since the velocity is stored as a vector per triangle. Nevertheless, the discretization allows a consistent and efficient definition of differential operators acting on the discretized vector and scalar fields. Note that these operators are again defined in the spirit of DEC. Another advantage of the chosen discretization is that the gradient of a scalar field is a piecewise constant vector valued function on triangles, which is consistent with the discretization of vector fields.

4.4.2. Discretization of Differential Operators

Two differential operators are applied to the field variables in the SWE, namely the gradient of the water depth and of the acceleration in vertical direction and the divergence of the velocity or of the product between the water depth and the velocity, depending whether the water depth advection is handled separately or not. The implicit solvers in Section 5.3.1 also require the definition of a scalar counterpart of the Laplacian of a scalar field. In the following, each of these discrete differential operators is presented shortly.

Gradient

We employ a classical finite element approach for the gradients. Given the piecewise linear basis functions ϕ_i centered at the vertices. The gradient of a basis function turns into a piecewise constant vector field on the adjacent triangles. We denote the gradient of vertex i in triangle k as $\nabla\phi_{ik}$. This gradient is the vector perpendicular to the edge $\mathbf{e}_{ik}$ opposite to vertex i in triangle k with a magnitude given by the inverse of the triangle height h_{ik} w.r.t. the edge $\mathbf{e}_{ik}$ and the gradient is oriented toward vertex i. See Figure 4.4 for a graphical depiction. A simple computation reveals that the gradient $\nabla\phi_{ik}$ corresponds to the edge $\mathbf{e}_{ik}$ rotated by 90 degrees in the plane of the triangle scaled by the inverse of twice the triangle area A_k:

$$\nabla\phi_{ik} = \frac{1}{h_{ik}}\mathbf{R}_{\frac{\pi}{2}}\frac{\mathbf{e}_{ik}}{\|\mathbf{e}_{ik}\|} = \frac{\|\mathbf{e}_{ik}\|}{2A_k}\mathbf{R}_{\frac{\pi}{2}}\frac{\mathbf{e}_{ik}}{\|\mathbf{e}_{ik}\|} = \frac{1}{2A_k}\mathbf{R}_{\frac{\pi}{2}}\mathbf{e}_{ik}$$

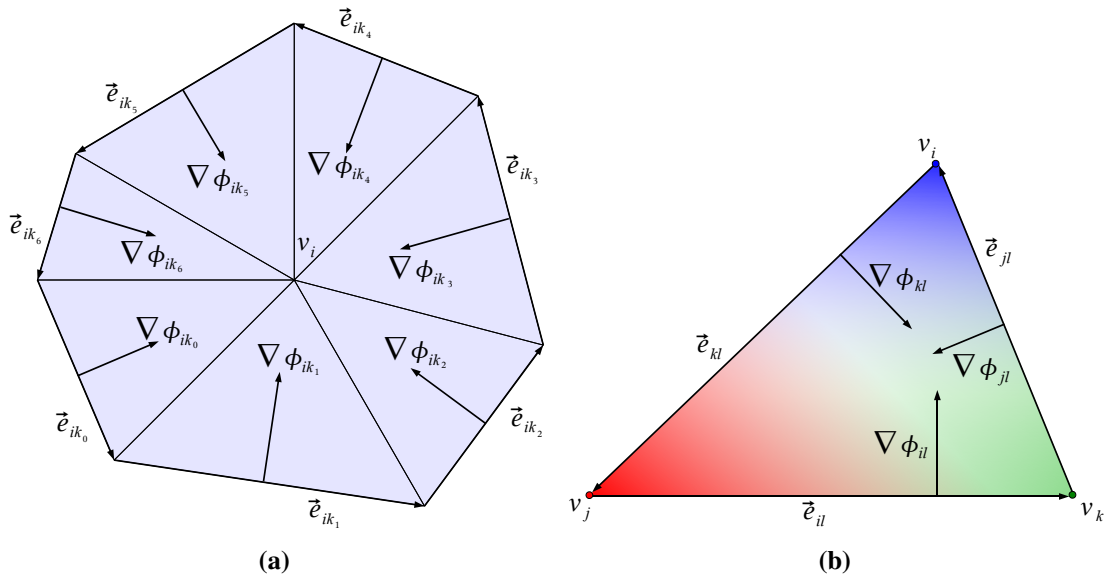


Figure 4.4.: *Gradients of a Linear Basis Function.*
(a) *The gradient of a linear basis function at vertex v_i induces a piecewise constant vector field $\nabla\phi_{ik_j}$ per adjacent triangle k_j.*
(b) *In a triangle l, only three gradients are non-vanishing. The colors indicate the scalar linear basis function at the vertices.*

A short glimpse at the Figure 4.5 reveals that the gradient $\nabla\phi_{ik}$ can be represented as a linear combination of the edges adjacent to vertex i in triangle k:

$$\begin{aligned}\cot\alpha\mathbf{a} + \cot\beta\mathbf{b} &= \cot\alpha(\mathbf{h}+\mathbf{c}_2) + \cot\beta(\mathbf{h}-\mathbf{c}_1)\\ &= -\mathbf{R}_{\frac{\pi}{2}}\mathbf{c}_1 - \mathbf{R}_{\frac{\pi}{2}}\mathbf{c}_2 + \cot\alpha\mathbf{c}_2 - \cot\beta\mathbf{c}_1\\ &= -\mathbf{R}_{\frac{\pi}{2}}\mathbf{c} + \frac{\mathbf{c}}{\|\mathbf{c}\|}(\|\mathbf{c}_2\|\frac{\|\mathbf{c}_1\|}{\|\mathbf{h}\|} - \|\mathbf{c}_1\|\frac{\|\mathbf{c}_2\|}{\|\mathbf{h}\|})\\ &= -\mathbf{R}_{\frac{\pi}{2}}\mathbf{c}\end{aligned}$$

where the identities $\cot\alpha\mathbf{h} = -\mathbf{R}_{\frac{\pi}{2}}\mathbf{c}_1$, $\cot\beta\mathbf{h} = -\mathbf{R}_{\frac{\pi}{2}}\mathbf{c}_2$, $\cot\alpha = \frac{\|\mathbf{c}_1\|}{\|\mathbf{h}\|}$ and $\cot\beta = \frac{\|\mathbf{c}_2\|}{\|\mathbf{h}\|}$ were used. For a more detailed derivation and additional properties, see [PP93].

Obviously there are exactly three non-vanishing gradients of the basis functions in a triangle. The gradient at a point $\mathbf{x}$ in triangle k of a discrete scalar field thus is given by

$$\nabla\hat{f}(\mathbf{x}) = \nabla\sum_{v_i\in\mathcal{V}}\phi_i(\mathbf{x})f_i = \sum_{\mathbf{e}_{ik}\in T_k}\nabla\phi_{ik}f_i. \tag{4.5}$$

The magnitude of the gradient of a basis function is inverse proportional to the height of a triangle. This causes stability problems for badly shaped triangles. An upper and lower bound for the angles in the triangles are therefore introduced. If an angle exceeds or falls below the upper resp. lower bound, this angle is set back to the corresponding bound. Note that although only one angle of a triangle might violate such a criterion, all the other angles inside the same triangle must be changed accordingly such that all the angles sum up to 180 degrees. Otherwise spurious artifacts occur since, for example, a constant scalar field would not result

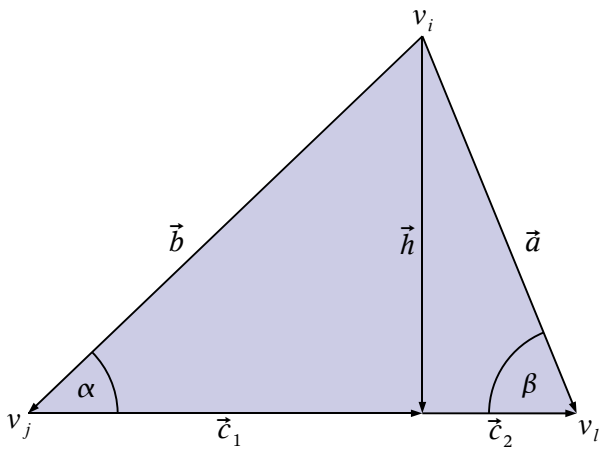


Figure 4.5.: *The gradient of the linear basis function ϕ_i of vertex v_i in triangle k can be formulated as a weighted sum of the outgoing edges using the cotangent of the angle opposite to the edge as the weight factor.*

in a vanishing gradient anymore. Fortunately, the cotangent formulation of the basis function gradients facilitates such a rescaling of the angles. In all our examples, we used a lower bound of 10 degrees which implicitly implies an upper bound of 160 degrees.

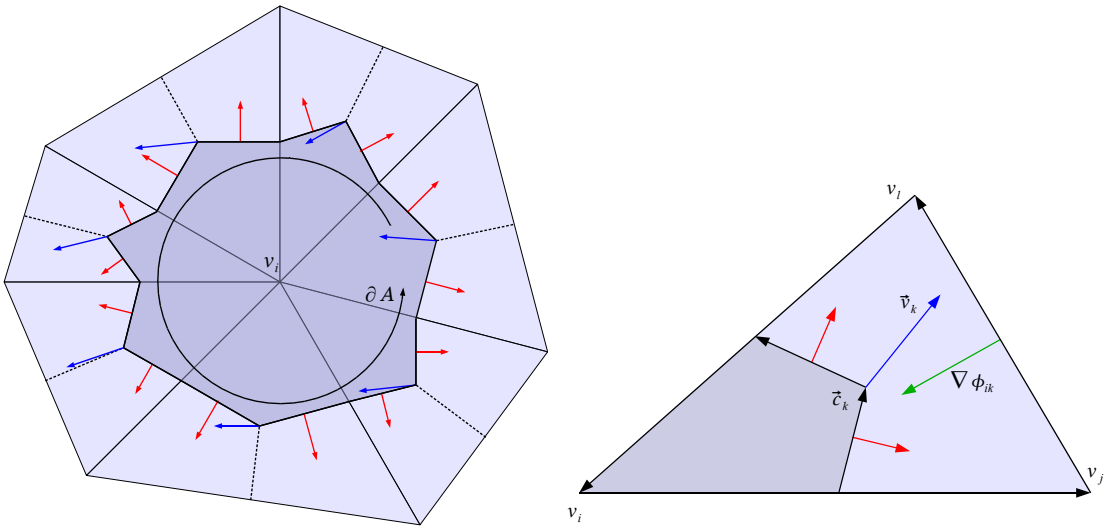
Divergence

A proper discretization of the divergence is crucial for the simulation because the conservation of mass involves the divergence operator and as pointed out earlier, the conservation of mass is very important for a plausible simulation. The chosen discretization follows the concepts of DEC and is basically a finite volume approach. Rather than trying to approximate the spatial derivatives of a vector field $\mathbf{v}$, the discrete divergence of the vector field is averaged over an area by computing the flux across the boundary of this area

$$\frac{1}{A}\iint_A \nabla \cdot \mathbf{v} dA = \frac{1}{A}\int_{\partial A} \mathbf{v} \cdot \mathbf{n} ds.$$

The cell can be given by either a triangle of the primal mesh or by a dual cell of the dual mesh. Remember that since barycenters of the triangles are used as dual vertices and since the dual cells should not overlap but should nonetheless partition the whole simulation domain, the boundary of a dual cell is given by the piecewise linear curve defined by the triangle barycenters and the midpoints of the primal edges (see Figure 4.6).

In the case of a primal cell, the above line integral is simply discretized by a summation over the edges adjacent to the triangle. Since a discrete vector field is defined as a piecewise constant vector field at the triangles, an appropriate interpolation scheme must be defined to interpolate the discrete vector field at the edges. The flux across the boundary of a dual cell is given by a summation of the fluxes across the piecewise linear boundary of the dual cell. Note that in contrast to the flux in a primal cell, no interpolation scheme is required for the flux across the boundary of a dual cell because the piecewise constant discrete vector field can be used directly. The division by the cell area deserves special attention. The most obvious idea is to divide the



(a) **(b)**

Figure 4.6.: *Computation of the flux across a dual cell boundary.*
(a) *The flux of a vector field (blue) in a dual cell A (dark gray) at a vertex v_i is the sum of the fluxes across the piecewise linear boundary of the dual cell ∂A (thick, solid line). The normal (red) of a dual edge is defined to point outwards.*
(b) *The flux across the dual boundary in a primal triangle k is equal to the scalar product between the gradient of the linear basis function (green) and the vector field (blue).*

flux across the boundary by the area of the cell. But doing so, the area-scaled flux from two adjacent cells across a cell interface does not sum to zero and thus the flux is not conserved and artificial sources and sinks are introduced. A better idea is to associate each segment of the cell boundary with its own averaging volume. For a segment of the dual cell boundary, a natural choice for the averaging area is given by one third of the corresponding triangle area. This choice leads to a compact formulation of the discrete divergence. Consider the boundary of a dual cell restricted to a single triangle, see Figure 4.6. The area averaged flux due to the vector field over the boundary of the dual cell i in triangle k is given by

$$\begin{aligned}
&\frac{3}{A_k}\mathbf{v_k} \cdot (\mathbf{R}_{-\frac{\pi}{2}}(\mathbf{c} - \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j)) + \mathbf{R}_{\frac{\pi}{2}}(\mathbf{c} - \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_l))) \\
=&\frac{3}{A_k}\mathbf{v_k} \cdot \mathbf{R}_{\frac{\pi}{2}}(-\frac{1}{6}(2\mathbf{x}_j - \mathbf{x}_i - \mathbf{x}_l) + \frac{1}{6}(2\mathbf{x}_l - \mathbf{x}_i - \mathbf{x}_j)) \\
=&-\frac{1}{2A_k}\mathbf{v} \cdot \mathbf{R}_{\frac{\pi}{2}}(\mathbf{x}_l - \mathbf{x}_j) \\
=&-\mathbf{v}_k \cdot \nabla\phi_{ik}.
\end{aligned}$$

Thus, the divergence over the dual cell i reduces to a summation of the scalar products between the vector field and the gradient of the basis function of vertex v_i

$$(\nabla \cdot \mathbf{v})_i = -\sum_{v_k \in \mathcal{N}(v_i)} \mathbf{v}_k \cdot \nabla\phi_{ik} \tag{4.6}$$

This approach perfectly conserves the flux across dual cell boundaries.

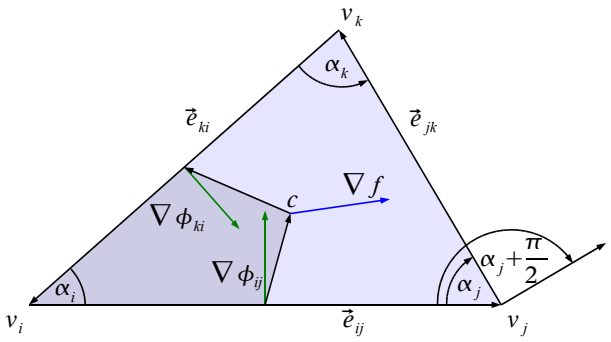


Figure 4.7.: *Diagram of the notation used in the derivation of the discrete Laplacian.*

Laplacian

To complete the derivation of the discrete differential operators, the discrete Laplacian is derived here. The derivation presented here is slightly different from the famous derivations presented in [MDSB02], but we will end up with the same discrete Laplacian nonetheless. Our derivation clearly shows the consistency of the discrete operators (gradient, divergence and Laplacian), as we start the derivation using the discrete versions of the gradient and divergence. This shows that the differential operators, which are involved in the SWE, are consistently discretized. Similar to [MDSB02] we will also use a finite volume approach to derive the discrete Laplacian.

The discrete Laplacian should obey the same definition as its continuous counterpart, namely the continuous Laplacian is defined as the divergence of the gradient $\nabla^2 f = \nabla \cdot \nabla f$ of a scalar function f. Using the discrete divergence of Equation (4.6) and using the concepts of the finite volume approach, the discrete Laplacian at vertex v_i is given by

$$\begin{aligned}\frac{1}{A_i}\iint_{A_i} \nabla^2 f dA = \frac{1}{A_i}\iint_{A_i} \nabla \cdot \nabla f dA = -\frac{1}{A_i}\sum_{v_k \in \mathcal{N}(v_i)} \nabla f \cdot \nabla \phi_{jk} A_{ijk} \\ = \frac{1}{A_i}\sum_{v_k \in \mathcal{N}(v_i)} \nabla f \cdot (\frac{1}{2}\mathbf{R}_{-\frac{\pi}{2}}\mathbf{e}_{jk}) = \frac{1}{A_i}\sum_{v_k \in \mathcal{N}(v_i)} \nabla f \cdot \mathbf{n}_{jk}\end{aligned}$$

where A_{ijk} denotes the triangle area of the triangle given by vertices v_i, v_j and v_k, $\nabla \phi_{jk}$ is the gradient of the basis function of vertex v_i in this triangle and $\mathbf{n}_{jk} = -\nabla \phi_{jk} A_{ijk} = \frac{1}{2}\mathbf{R}_{-\frac{\pi}{2}}\mathbf{e}_{jk}$ denotes the (scaled) normal of the edge opposite to vertex i in this triangle. We restrict the derivation first to a single triangle. Consider the triangle and notation depicted in Figure 4.7. Throughout this derivation $\mathbf{e}_{ij}$ denotes the edge from vertex i to vertex j, $\nabla \phi_{ij}$ is the gradient associated with this edge and α_k is the angle opposite to this edge. The discrete gradient of the

scalar field in a triangle is given by Equation (4.5) which can be slightly reformulated

$$\begin{aligned}
\nabla f &= f_i \nabla \phi_{jk} + f_j \nabla \phi_{ki} + f_k \nabla \phi_{ij} \\
&= \frac{1}{2A} \mathbf{R}_{\frac{\pi}{2}} \left(f_i(\mathbf{x}_k - \mathbf{x}_j) + f_j(\mathbf{x}_i - \mathbf{x}_k) + f_k(\mathbf{x}_j - \mathbf{x}_i) \right) \\
&= \frac{1}{2A} \mathbf{R}_{\frac{\pi}{2}} \left((f_j - f_i)(\mathbf{x}_i - \mathbf{x}_k) + (f_k - f_i)(\mathbf{x}_j - \mathbf{x}_i) \right) \\
&= \nabla \phi_{ki}(f_j - f_i) + \nabla \phi_{ij}(f_k - f_i) \\
&= \frac{1}{2A} \left((-\mathbf{e}_{jk} \cot \alpha_i - \mathbf{e}_{ji} \cot \alpha_k)(f_j - f_i) + (-\mathbf{e}_{ki} \cot \alpha_j - \mathbf{e}_{kj} \cot \alpha_i)(f_k - f_i) \right).
\end{aligned}$$

Due to orthogonality, the scalar product between the gradient ∇f and the edge normal $\mathbf{n}_{jk}$ cancels out the terms involving the edge $\mathbf{e}_{jk}$

$$\begin{aligned}
\mathbf{n}_{jk} \cdot \nabla f = &\left(\frac{1}{2} \mathbf{R}_{-\frac{\pi}{2}} \mathbf{e}_{jk} \right) \cdot \Big(\frac{1}{2A} \big((-\mathbf{e}_{jk} \cot \alpha_i - \mathbf{e}_{ji} \cot \alpha_k)(f_j - f_i) \\
&+ (-\mathbf{e}_{ki} \cot \alpha_j - \mathbf{e}_{kj} \cot \alpha_i)(f_k - f_i) \big) \Big) \\
= &\left(\frac{1}{2} \mathbf{R}_{-\frac{\pi}{2}} \mathbf{e}_{jk} \right) \cdot \left(\frac{1}{2A} \big((-\mathbf{e}_{ji} \cot \alpha_k)(f_j - f_i) + (-\mathbf{e}_{ki} \cot \alpha_j)(f_k - f_i) \big) \right).
\end{aligned}$$

We now remark that $(\mathbf{R}_{-\frac{\pi}{2}} \mathbf{e}_{jk}) \cdot \mathbf{e}_{ji}$ is equal in magnitude to twice the triangle area

$$\begin{aligned}
(\mathbf{R}_{-\frac{\pi}{2}} \mathbf{e}_{jk}) \cdot \mathbf{e}_{ji} &= -\|\mathbf{e}_{jk}| \, \|\mathbf{e}_{ji}\| \cos(\frac{\pi}{2} + \alpha_j) \\
&= -\|\mathbf{e}_{jk}| \, \|\mathbf{e}_{ji}\| \sin(\alpha_j) \\
&= -2A.
\end{aligned}$$

The same holds for $(\mathbf{R}_{-\frac{\pi}{2}} \mathbf{e}_{jk}) \cdot \mathbf{e}_{ki}$ and therefore we end up with

$$\mathbf{n}_{jk} \cdot \nabla f = \frac{1}{2} (\cot \alpha_k (f_j - f_i) + \cot \alpha_j (f_k - f_i)).$$

The final discretized Laplacian operator is given by summing the above terms over the one-ring neighborhood and scaling by the area of the dual cell. The terms can be re-sorted to finally end up with the very same formula as derived in [MDSB02], see also Figure 4.8.

$$\nabla^2 f = \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}(v_i)} (\cot \alpha_j + \cot \beta_j)(f_j - f_i). \tag{4.7}$$

This discretization results in a linear operator which can be written as the product $\mathbf{D}\hat{\nabla}^2$ between a diagonal matrix $\mathbf{D}$ and a highly sparse and symmetric matrix $\hat{\nabla}^2$. The diagonal matrix takes care of the division by the finite-volume area whereas the sparse, symmetric matrix consists of the cotangent weights $\hat{\nabla}^2_{ij} = \hat{\nabla}^2_{ji} = \cot \alpha_j + \cot \beta_j$ in off-diagonal elements and of $\hat{\nabla}^2_{ii} = -\sum_{v_j \in \mathcal{N}(v_i)} \cot \alpha_j + \cot \beta_j$ in diagonal elements.

As pointed out in [MDSB02] the discretization is valid, as long as the finite-volume region of the averaging goes through the edge midpoints. Remember that we use barycentric finite-volume areas which is advantageous compared to Voronoi cells in case of obtuse triangles.

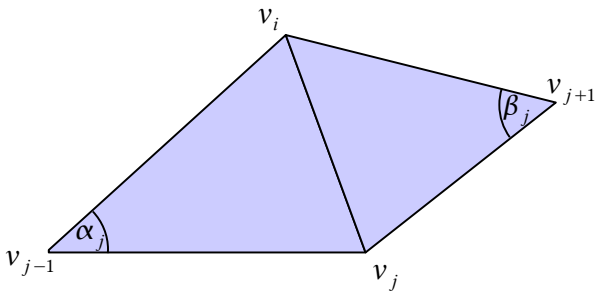


Figure 4.8.: *The discrete Laplacian of a scalar function f at a vertex v_i can be written as a weighted sum of the differences $f_j - f_i$ over the adjacent vertices v_j. The weight for the vertex v_j is given by the sum of the two cotangents of the angles α_j and β_j.*

But even this choice of discretization is not without problems. Firstly, the cotangent weights $w_i = \cot \alpha_i + \cot \beta_i$ are not guaranteed to be positive. Indeed, if $\alpha_i + \beta_i > \pi$, then the cotangent weight becomes negative. These negative weights can lead to flipped triangles. In our implementation, negative cotangent weights are simply set back to zero to guard against flipped triangles. Secondly, this discretized Laplacian is not completely intrinsic. It might evaluate to slightly different values for different triangulations of the very same surface.

The singularities of the cotangent at $k\pi$ give rise to another problem. A triangle obvisously can not have an internal angle of $\theta = k\pi$. Nonetheless, an angle $\theta \in [\pi - \epsilon, \pi]$, where ϵ is a small positive number, is possible in badly shaped triangles. Such an angle would lead to a huge cotangent weight which is not conducive to the condition of the linear system. A simple solution is to limit the angles, or equivalently limit the cosines of the angles to lie in $[-1+\epsilon, 1-\epsilon]$ where again ϵ is a small positive number. Despite these drawbacks, the discretization (4.7) is nonetheless the most widely used discretization of the Laplacian nowadays.

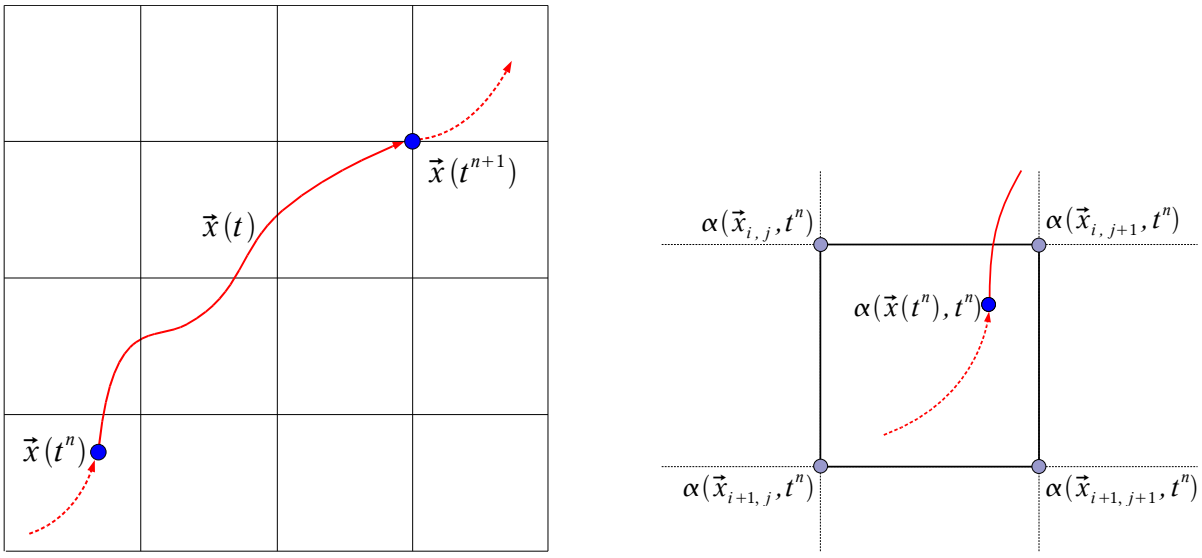
5

Temporal Discretization

With a suitable spatial discretization at hand, the temporal discretization will be addressed in this chapter. Remember that based on the experiments with several different spatial discretization schemes, we chose to store the water height η as a piecewise linear scalar field with sampling nodes at vertices and the velocity $\mathbf{v}$ as a piecewise constant vector field with sampling nodes at triangle barycenters.

5.1. Fractional Step Method

The SWE can be formulated with a material derivative as in Equation (3.14) and Equation (3.15). The material derivatives clearly show that the SWE basically consist amongst others of an advectional part. The advection imposes difficulties to ensure stability because the CFL condition is mostly dominated by the advection, as has been observed in the finite volume formulation in Section 4.3. In addition, the height and velocity advections are both non-linear in the system variables of the SWE. To circumvent these problems, we chose to use the fractional step method. This method was introduced to the computational fluid dynamic community in [Cho69] and has been used in the seminal paper [Sta99] as well. The idea is very simple and intuitive. Rather than solving the time evolution of the whole PDE to the next discretized point in time in one step, the system is solved in several separate sequential steps. Each step takes care of one specific operator involved in the PDE. The input state to one step is the output of the previous step. For each operator involved in the PDE, a specific scheme tailored to the properties of the operator is used to advance the system to the next intermediate state. For example, the advection can be solved with an unconditionally stable method called Semi-Lagrangian advection.



(a) **(b)**

Figure 5.1.: *The backtracing and interpolation step of the Semi-Lagrangian advection on a regular grid.*
(a) *Characteristic line (red) going through a grid vertex at time* t^{n+1}.
(b) *The end point of the tracing is in general not a mesh vertex and an interpolation method for the advected quantity* $\alpha(\mathbf{x}(t), t)$ *based on the known values at the mesh vertices at time* t^n *has to be used.*

In our solution scheme we indeed solve the advection with the Semi-Lagrangian method. The remaining part of the PDE can then be solved with any method of choice. Note that we investigated both formulations of the continuity equation, i.e., with advection as in Equation (3.14) and without advection as in Equation (3.11).

The next section presents the Semi-Lagrangian method on a simplicial complex and afterwards several methods to integrate the remaining parts of the SWE are presented. Note that in the case of the SWE, the Semi-Lagrangian approach is only used to advect the velocity and maybe the water depth, depending which formulation of the continuity equation is used. However, the Semi-Lagrangian advection on a 2-manifold is presented in a quite general formulation since this scheme is applicable to any quantity which should get advected.

5.2. Semi-Langrangian Advection on 2-Manifold

Semi-Lagrangian advection was introduced in the computer graphics community by Stam in 1999 [Sta99]. This scheme provides an unconditionally stable solution to the advection equation, given a proper interpolation scheme is used. Instead of approximating the differential operators in the advection equation

$$\frac{D\alpha(\mathbf{x},t)}{Dt} = \frac{\partial\alpha(\mathbf{x},t)}{\partial t} + \frac{\partial\alpha(\mathbf{x},t)}{\partial\mathbf{x}} \cdot \frac{\partial\mathbf{x}}{\partial t} = \alpha_t + \mathbf{v} \cdot \nabla\alpha = 0$$

with an upwind scheme, the Semi-Lagrangian method follows the lines of characteristic backward in time to deduce which value is going to get advected to the starting point of the tracing

at the next discretized point in time. The method consists of three sequential steps, see also Figure 5.1:

- Back tracing: Given a starting point $\mathbf{x}(t^{n+1})$ at time t^{n+1}, the back tracing follows the line of characteristic at this point backwards in time to find the point $\mathbf{x}(t^n)$ which will get advected to the starting point.
- Interpolation: Because the point $\mathbf{x}(t^n)$ is in general not a sampling node, the value $\alpha(\mathbf{x}(t^n), t^n)$ which gets advected must be interpolated at this point.
- Forward transportation: Finally, the interpolated value is transported forward and stored at the starting point of the tracing. Note that this last step is only required if the advected quantity is vector valued and if the embedding space of the characteristic lines has non-vanishing curvature. To see this, observe that the local frame at a point traveling along a characteristic curve on the manifold surface is expected to rotate with the Frenet Frame of this curve. Therefore, the rotation of the Frenet Frame has to be applied to the interpolated vector as well. However, if the curvature of the manifold surface is vanishing the interpolated value can directly be stored at the starting point.

Straight forward methods for each step can easily be implemented for a regular grid in Euclidean space (2D or 3D). The back tracing can even be made second order accurate ([KLLR07], [SFK+07]) while the interpolation can be improved by using a more accurate interpolation scheme than bi- or trilinear interpolation. As a drawback, the Semi-Lagrangian advection introduces a considerable amount of numerical damping due to the interpolation step and unfortunately, there is no explicit control over the amount of this artificial numerical damping. However, in a real time application numerical stability is more important than less numerical damping.

The curvature of a two dimensional manifold embedded in three dimensional Euclidean space complicates the Semi-Lagrangian advection considerably. The next subsections describe each of the aforementioned steps of the Semi-Lagrangian advection on a discrete manifold in detail and give guidelines how to implement them on a simplicial complex in an efficient manner.

5.2.1. Back Tracing

The tracing starts at a sampling node $\mathbf{x}(t^{n+1})$ of the field $\alpha(\mathbf{x}, t)$ which should get advected and integrates the position of this sampling node along the characteristic curve going through this node backwards in time. The most simple discretization of the backward time integration assumes the velocity at the starting point $\mathbf{v}(\mathbf{x}(t^{n+1}), t)$ to be constant in the time interval $[t^n, t^{n+1}]$ and performs one explicit Euler step backwards in time $\mathbf{x}(t^n) = \mathbf{x}(t^{n+1}) - (t^{n+1} - t^n)\mathbf{v}(\mathbf{x}(t^{n+1}), t^{n+1})$. If the velocity field is not sampled at the same nodes as the advected field $\alpha(\mathbf{x}, t)$, the velocity has to be first interpolated to the sampling node $\mathbf{x}(t^{n+1})$, see Section 5.2.2 how this can be done on a discrete manifold. Note that if the advected quantity is the velocity itself (self-advection) the velocity of the previous time step $\mathbf{v}(\mathbf{x}(t^{n+1}), t^n)$ is used for backward Euler step.

Consider a vector field on a two dimensional manifold embedded in three dimensional Euclidean space, for example the velocity field of the SWE. The characteristic lines due to such a

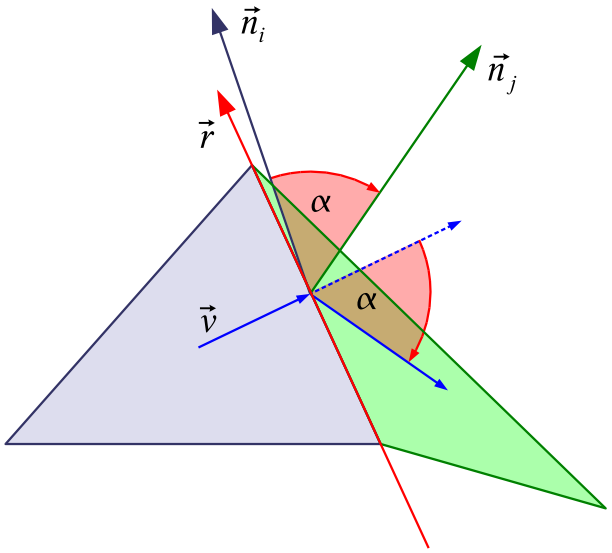


Figure 5.2.: *When the backtracing reaches a triangle edge the velocity vector* $\mathbf{v}$ *has to be rotated onto the plane defined by the neighboring triangle* T_j*, otherwise the tracing might leave the simplicial complex.*

vector field are also on the manifold and the curvature of the manifold is thus reflected in these characteristic lines. In the discrete setting this means that whenever following a velocity vector for a certain amount of time one is going to end up leaving the simplicial complex, provided the curvature is non vanishing. But because a two dimensional simplicial complex consists of a set of connected triangles and a triangle has zero curvature, the basic idea of following a point along the velocity vector backwards in time is still applicable, only for short time intervals however. Whenever the tracing reaches a triangle edge care has to be taken not to leave the simplicial complex. For example in [SY04], this problem is solved by rotating the velocity vector of the original triangle onto the plane given by the neighboring triangle, see Figure 5.2. The angle of rotation α is obviously given by the angle between the normals $\mathbf{n}_i$ and $\mathbf{n}_j$ of the two triangles and the axis of rotation $\mathbf{r}$ corresponds to the common edge of the two triangles. However, for efficiency reasons we used a slightly different approach which is described next.

Three ray-line intersection tests are required to compute which triangle edge will be reached when following a vector inside the triangle. These intersection tests are quite costly in computational time. Fortunately, the computations can be done more efficient using barycentric coordinates for the velocity and the starting point. Let a given triangle be defined by its three vertices v_0, v_1, v_2 and assume the starting point $\mathbf{x}$ and the velocity $\mathbf{v}$ in that triangle are given in barycentric coordinates $(a, b, c)^T$ resp. $(e, f, g)^T$ w.r.t. the triangle. Then the ray emanating from $\mathbf{x}$ in direction $\mathbf{v}$ can be written as

$$\mathbf{x} + t\mathbf{v} = \begin{bmatrix} \mathbf{x}(v_0) & \mathbf{x}(v_1) & \mathbf{x}(v_2) \end{bmatrix} \left(\begin{pmatrix} a \\ b \\ c \end{pmatrix} + t \begin{pmatrix} e \\ f \\ g \end{pmatrix} \right).$$

This representation reduces the computation of the intersection time t^* between the ray and the

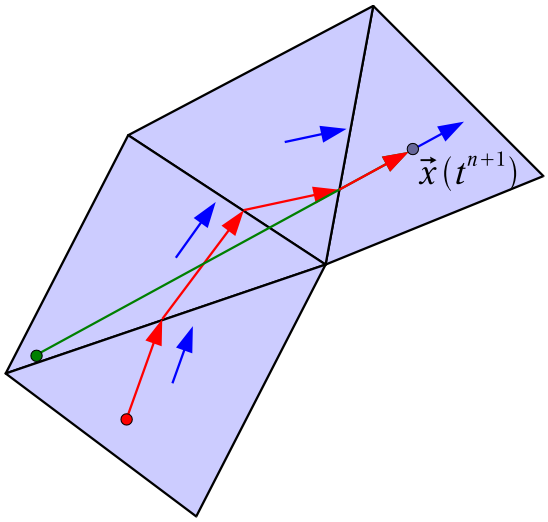


Figure 5.3.: *Semi-Lagrangian back tracing on irregular grids. If the velocity at the starting point* $\mathbf{x}(t^{n+1})$ *of the back tracing is held constant throughout the back tracing step the green line results. If the velocity of the current triangle is used the piecewise linear red curve results. Note that for simplicity, a flat domain with vanishing curvature is depicted.*

edge (v_1, v_2) to one division

$$\{\mathbf{x} + t\mathbf{v}\} \cap \{(1-s)\mathbf{x}(v_1) + s\mathbf{x}(v_2)\} \quad \rightarrow \quad a + t^* e = 0 \quad \rightarrow \quad t^* = -\frac{a}{e}.$$

The barycentric coordinates of the intersection point $\mathbf{x} + t^*\mathbf{v}$ are efficiently and easily computable. An important fact is that there is no need to recompute the barycentric coordinates of this intersection point w.r.t. the neighboring triangle because the barycentric coordinates can be permuted depending on which vertices of the two adjacent triangles make up the common edge. Therefore a computation of the Euclidean coordinates of the edge intersection point is not required at all.

The computation of the new barycentric representation of the velocity w.r.t. the neighboring triangle can be avoided by using the velocity of the neighboring triangle to continue the tracing instead. See Figure 5.3 for a graphical depiction. Hence, only one pass over all triangles is required to compute the barycentric representations of the velocities. As we will see shortly, the barycentric representation of the velocity vectors is required for the velocity interpolation anyway. Note that if the tracing is started at the triangle barycenters, the barycentric coordinates of the starting point are known a priori.

5.2.2. Vector Interpolation

Because the back-tracing step can end up at any point on the simplicial complex, the Semi-Lagrangian advection step needs an interpolation scheme for the advected quantity at any point on the simplicial complex. For a scalar valued advected quantity, this interpolation is straight forward because scalar fields are sampled at the vertices of the simplicial complex and hence barycentric interpolation can be used directly, for example. However, if the advected quantity

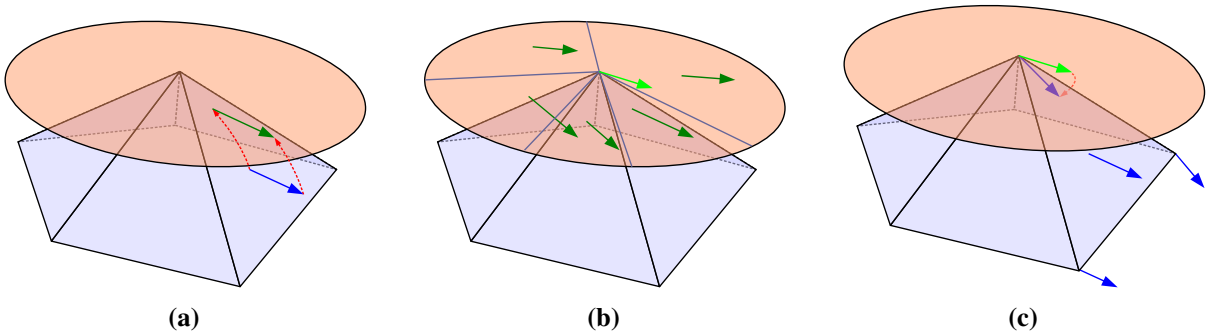


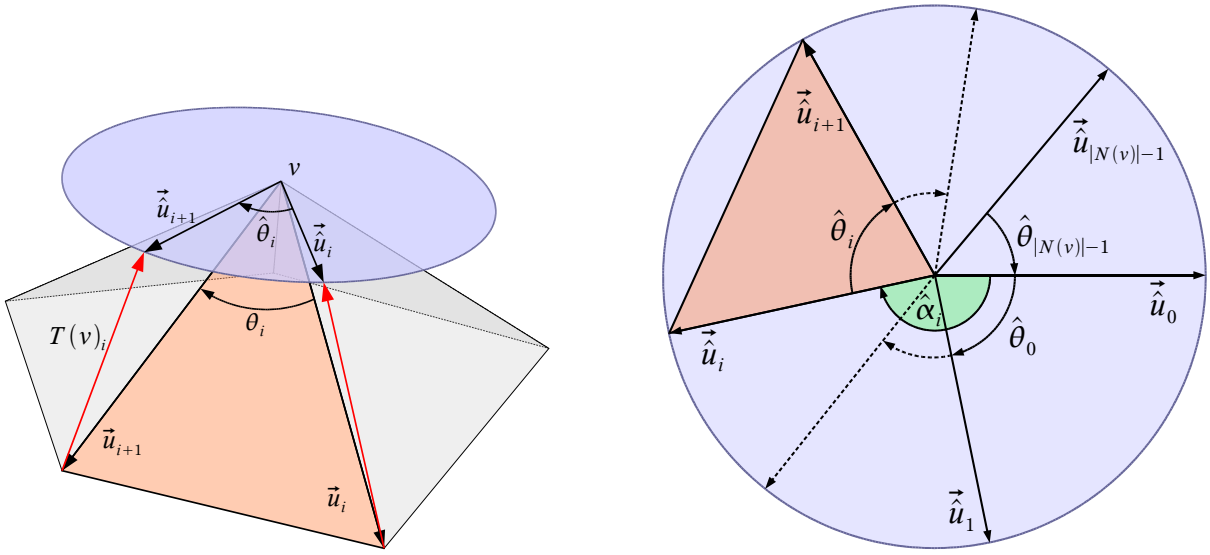
Figure 5.4.: *The steps involved in the interpolation of a vector at a vertex of a simplicial complex.*
(a) *Find a mapping (red) of the vector in the triangle (blue) to a parameterization plane (visualized as the tangent plane) at the vertex.*
(b) *Interpolation of the vector at the vertex (green) in the parameterization plane using all the vectors in the parameterization plane which have been mapped from the adjacent triangles (darker green).*
(c) *Map the interpolated vector at the vertex from the parameterization plane back to the plane of the triangle and perform the final interpolation using the other interpolated vectors at the other two vertices and the original vector at the triangle barycenter. Note that all these vectors lie in the plane of the triangle.*

is vector valued, then the interpolation becomes more complicated. It is not possible to simply use a weighted sum of the Euclidean 3D vectors of the surrounding triangles because the vector field is defined *in* the two dimensional manifold surface rather than in the embedding three dimensional Euclidean space and in general such a linear combination of vectors is no longer in the manifold. Hence, a proper *intrinsic* representation of the vector field must be defined.

Stam showed in [Sta03] how to apply his stable fluids solver [Sta99] to curved Catmull-Clark surfaces using an atlas of parameterizations. The Semi-Lagrangian advection is performed in the parameter domain which obviously is a two dimensional Euclidean space with zero curvature. The intrinsic representation is thus given by this parameterization. But because a 2-manifold is only locally homeomorphic to an Euclidean disc, many individual patches might make up the atlas of parameterizations. This is a problem because special care must be taken to enforce continuity across patch boundaries. In contrast, the method presented in [SY04] avoids the need for an atlas of parameterizations by working directly on the simplicial complex. Furthermore, the mesh has no restrictions besides being a proper triangle mesh (no 'triangle soup') and thus is applicable to a wide range of meshes. For these reasons, our method to find a proper intrinsic representation follows closely the ideas proposed in [SY04].

Our spatial discretization samples a vector field at the faces. Hence, a method has to be defined to interpolate a vector field, given the sampled vectors at the faces. As an intermediate step, the vectors at the triangles are first interpolated to the vertices of the simplicial complex. Then, given the sampled vectors at both the face barycenters and the vertices, the vector at any point on the simplicial complex is interpolated using barycentric interpolation.

We know that a d-dimensional manifold is locally homeomorphic to an open Euclidean d-disc. In our specific case, this means we can define a tangential disc at each vertex. The vectors of the adjacent faces are then mapped to this tangential disc and the interpolation of the vector field at the vertex takes place in this local disc. We follow exactly the same approach as described in



(a) *Flattening transformation in 3D space* **(b)** *Parameterization plane*

Figure 5.5.: *A piecewise constant vector field at the triangle faces is interpolated to a vertex v by defining for each adjacent triangle i a linear, angle ratio preserving mapping $T(v)_i$ (red arrows).*
(a) *The linear mapping maps the triangle i to a local parameterization plane at the vertex v which is visualized here as the tangent plane at the vertex.*
(b) *The linear, angle ratio preserving mapping maps the triangle fan at vertex v to a planar disc which serves as the local parameterization plane.*

[SY04] to find a mapping from each adjacent face to the tangential disc at a given vertex. For the final barycentric interpolation of the vector field at any point inside a triangle, the interpolated vector at a vertex in the local disc is mapped back to the triangle using the inverse of the mapping. Figure 5.4 gives a graphical overview of this process.

This subsection proceeds as follows: first, we describe how to find the mapping for a given vertex and an adjacent triangle. After that we describe how to interpolate a vector field at the vertex inside the tangential disc, given the vectors at the face barycenters of the adjacent triangles and the mappings computed in the previous step. Then it is shown how to map the interpolated vector from the disc back to an adjacent triangle. Finally, several improvements are derived to speed up the vector interpolation process.

Angle Ratio Preserving Linear Mapping

The same method for the derivation of a mapping is used as in [SY04]. Given a vertex v a *linear* transformation $\mathbf{T}(v)_i$ is derived for each adjacent triangle i. These linear transformations are defined such that the ratios of angles of the triangle fan at vertex v are preserved. This is illustrated intuitively as locally unfolding the simplicial complex at the vertex v and flatten its 1-ring neighborhood $\mathcal{N}(v)$ by stretching or squeezing the angles such that they sum up to 2π (the curvature then is zero and thus the transformations map the local neighborhood of the vertex to a planar disc).

The following derivation uses the notations depicted in Figure 5.5. Given the outgoing edges $\mathbf{u}_i(v)$ of a vertex v where i enumerates the outgoing edges of the 1-ring neighborhood $\mathcal{N}(v)$ in clockwise order. Define θ_i as the angle between edges $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ and define α_i as the angle between edge $\mathbf{u}_i$ and $\mathbf{u}_0$. Further let $\hat{\theta}_i$ denote the angle of the angle ratio preserving mapping

$$\hat{\theta}_i = \frac{2\pi\theta_i}{\sum_{j\in\mathcal{N}(v)}\theta_j}.$$

Then $\hat{\alpha}_i$ is defined as the angle between the flattened edges $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_0$

$$\hat{\alpha}_i = \sum_{j\in\mathcal{N}(v), j<i} \hat{\theta}_j.$$

Without loss of generality, we set $\hat{\mathbf{u}}_0$ to $(1,0,0)^T$. With this definitions at hand, a linear and angle ratio preserving transformation can be defined for each triangle i adjacent to a vertex v

$$\begin{bmatrix}\hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i & \hat{\mathbf{n}}\end{bmatrix} = \mathbf{T}(v)_i \begin{bmatrix}\mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i\end{bmatrix} \begin{bmatrix}\frac{1}{\|\mathbf{u}_{i+1}\|} & 0 & 0\\ 0 & \frac{1}{\|\mathbf{u}_i\|} & 0\\ 0 & 0 & 1\end{bmatrix}.$$

where $\mathbf{n}_i$ is the unit normal of triangle i, $\hat{\mathbf{u}}_{i+1}$, $\hat{\mathbf{u}}_i$ are the desired vectors after the mapping of the respective edges and $\hat{\mathbf{n}}_i$ is the mapped normal:

$$\hat{\mathbf{u}}_{i+1} = \begin{pmatrix}\cos\hat{\alpha}_{i+1}\\ \sin\hat{\alpha}_{i+1}\\ 0\end{pmatrix} \qquad \hat{\mathbf{u}}_i = \begin{pmatrix}\cos\hat{\alpha}_i\\ \sin\hat{\alpha}_i\\ 0\end{pmatrix} \qquad \hat{\mathbf{n}}_i = \begin{pmatrix}0\\ 0\\ 1\end{pmatrix} \tag{5.1}$$

Solving for the transformation $\mathbf{T}(v)_i$ is straight forward:

$$\mathbf{T}(v)_i = \begin{bmatrix}\hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i & \hat{\mathbf{n}}_i\end{bmatrix} \begin{bmatrix}\|\mathbf{u}_{i+1}\| & 0 & 0\\ 0 & \|\mathbf{u}_i\| & 0\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i\end{bmatrix}^{-1}$$

Vector Interpolation at Vertices

Given the vectors $\mathbf{v}_i \in \mathbb{R}^3$ at the face barycenters of the adjacent triangles w.r.t. a vertex v. Each vector $\mathbf{v}_i$ is mapped to the flattened plane using the transformation $\mathbf{T}(v)_i$ at the triangle i w.r.t. the vertex v

$$\hat{\mathbf{v}}_i = \mathbf{T}(v)_i \mathbf{v}_i.$$

The interpolated vector $\hat{\mathbf{v}}(v)$ at the vertex v in the flattened plane is computed by a weighted average of the mapped face vectors $\hat{\mathbf{v}}_i$. There are several different ways to define the weight of a face vector. For example the area of the triangle could be used as the weight. As in [SY04],

we also chose to use the angle $\hat{\theta}_i$ of the wedge given by the two outgoing edges spanning the triangle plane as the weight of the face vector. Because the angles of all the adjacent wedges in the parameterization plane partition a circle, a division by 2π is required as a normalization

$$\begin{aligned}\hat{\mathbf{v}}(v) &= \frac{1}{2\pi}\sum_{i\in\mathcal{N}(v)}\hat{\theta}_i\hat{\mathbf{v}}(v)_i = \frac{1}{2\pi}\sum_{i\in\mathcal{N}(v)}\hat{\theta}_i\mathbf{T}(v)_i\mathbf{v}_i \\ &= \frac{1}{2\pi}\sum_{i\in\mathcal{N}(v)}\hat{\theta}_i\begin{bmatrix}\hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i & \hat{\mathbf{n}}_i\end{bmatrix}\begin{bmatrix}\|\mathbf{u}_{i+1}\| & 0 & 0\\ 0 & \|\mathbf{u}_i\| & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i\end{bmatrix}^{-1}\mathbf{v}_i.\end{aligned}$$

As pointed out earlier, the Semi-Lagrangian advection step requires an interpolation function for the advected quantity at any point $\mathbf{x}$ in a triangle k. In contrast to [SY04], where an interpolation that is continuous across the triangle edges is used, we decided to use a simpler interpolation scheme, namely barycentric interpolation. For each vertex v adjacent to the triangle k, the interpolated vector $\hat{\mathbf{v}}(v)$ in the flattened plane needs to get mapped back to the plane defined by triangle k. This is done by simply applying the inverse of the linear mapping from the triangle k to the flattened plane at vertex v, i.e., $\mathbf{v}(v)_k = \mathbf{T}(v)_k^{-1}\hat{\mathbf{v}}(v)$.

Optimizing the Flattening

The procedure described above allows the interpolation of a vector field, which is sampled at the triangle barycenters, at arbitrary points on the simplicial complex. The method was first implemented exactly as described in the previous paragraphs. But the procedure induced a considerable amount of computations: a vector interpolation in the parameterization plane requires a weighted sum of matrix-vector products and a 3-by-3 matrix inversion for each adjacent triangle. In addition, to map an interpolated vector in the flattened plane back to a triangle requires one more matrix inversion. Luckily, these costly oprations can be avoided. We derived a method to simplify and speed up the vector interpolation at a vertex by using barycentric coordinates rather than three dimensional Cartesian coordinates. In the following, our derivation is presented in detail.

Because the two dimensional manifold is embedded in $\mathbb{R}^3$ a vector field is most easily represented as a three dimensional vector field. However, the vector field is restricted to lie *in* the two dimensional manifold and therefore intrinsically seen, this vector field is actually a two dimensional vector field. This observation motivates to represent the vector at a triangle of the simplicial complex in barycentric coordinates with respect to this triangle. Let $\mathbf{u}_i(v), i \in \mathcal{N}(v)$ denote again the enumeration of the outgoing edges at vertex v in clockwise order. Then the vector $\mathbf{v}_i$ of a triangle i can be represented in the local basis of the wedge $(\mathbf{u}_{i+1}, \mathbf{u}_i)$:

$$\mathbf{v_i} = \begin{bmatrix}\mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i\end{bmatrix}\begin{pmatrix}a_i\\ b_i\\ 0\end{pmatrix}$$

Note that the barycentric representation was augmented with the normal vector of the triangle. Because the vector field is defined in the simplicial complex, this additional coordinate obviously is zero. If we use this representation and apply the linear mapping to map the vector to the flattened plane, the matrix inversion cancels out

$$
\begin{aligned}
\hat{\mathbf{v}}_i = \mathbf{T}(v)_i \mathbf{v}_i &= \mathbf{T}(v)_i \begin{bmatrix} \mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i \end{bmatrix} \begin{pmatrix} a_i \\ b_i \\ 0 \end{pmatrix} \\
&= \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i & \hat{\mathbf{n}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 & 0 \\ 0 & \|\mathbf{u}_i\| & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{u}_{i+1} & \mathbf{u}_i & \mathbf{n}_i \end{bmatrix} \begin{pmatrix} a_i \\ b_i \\ 0 \end{pmatrix} \\
&= \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i & \hat{\mathbf{n}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 & 0 \\ 0 & \|\mathbf{u}_i\| & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} a_i \\ b_i \\ 0 \end{pmatrix} \\
&= \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix}.
\end{aligned}
$$

This clearly shows that the mapping actually is a two dimensional rather than a three dimensional mapping. There is a very intuitive interpretation. The information provided by the normals $\mathbf{n}$ and $\hat{\mathbf{n}}$ gives rise to a rotation matrix which rotates the plane of the triangle to the flattened plane. But from the viewpoint of a vector on the simplicial complex this is an extrinsic rotation of the embedding space and thus is of no importance for the intrinsic flattening mapping.

If this approach is used for the vector interpolation, the following formula for the interpolated vector $\mathbf{v}(v)_k$ at vertex v mapped back to triangle k results:

$$
\begin{aligned}
\mathbf{v}(v)_k = \mathbf{T}(v)_k^{-1} \hat{\mathbf{v}}(v) &= \mathbf{T}(v)_k^{-1} \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \hat{\mathbf{v}}_i \\
&= \mathbf{T}(v)_k^{-1} \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \\
&= \begin{bmatrix} \mathbf{u}_{k+1} & \mathbf{u}_k & \mathbf{n}_k \end{bmatrix} \begin{bmatrix} \frac{1}{\|\mathbf{u}_{k+1}\|} & 0 & 0 \\ 0 & \frac{1}{\|\mathbf{u}_k\|} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}_{k+1} & \hat{\mathbf{u}}_k & \hat{\mathbf{n}} \end{bmatrix}^{-1} \\
&\quad \cdot \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix}
\end{aligned}
$$

$$= \begin{bmatrix} \mathbf{u}_{k+1} & \mathbf{u}_k & \mathbf{n}_k \end{bmatrix} \begin{bmatrix} \frac{1}{\|\mathbf{u}_{k+1}\|} & 0 & 0 \\ 0 & \frac{1}{\|\mathbf{u}_k\|} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\hat{\alpha}_{k+1} & \cos\hat{\alpha}_k & 0 \\ \sin\hat{\alpha}_{k+1} & \sin\hat{\alpha}_k & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}$$
$$\cdot \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \begin{bmatrix} \cos\hat{\alpha}_{i+1} & \cos\hat{\alpha}_i \\ \sin\hat{\alpha}_{i+1} & \sin\hat{\alpha}_i \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \tag{5.2}$$

The inverse matrix appearing in the above formula obviously has a special structure:

$$\begin{bmatrix} \cos\hat{\alpha}_{k+1} & \cos\hat{\alpha}_k & 0 \\ \sin\hat{\alpha}_{k+1} & \sin\hat{\alpha}_k & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} =$$
$$\frac{1}{\cos\hat{\alpha}_{k+1}\sin\hat{\alpha}_k - \cos\hat{\alpha}_k\sin\hat{\alpha}_{k+1}} \begin{bmatrix} \sin\hat{\alpha}_k & -\cos\hat{\alpha}_k & 0 \\ -\sin\hat{\alpha}_{k+1} & \cos\hat{\alpha}_{k+1} & 0 \\ 0 & 0 & \cos\hat{\alpha}_{k+1}\sin\hat{\alpha}_k - \cos\hat{\alpha}_k\sin\hat{\alpha}_{k+1} \end{bmatrix}$$

By inserting this inverse into Equation (5.2) it becomes evident that we can completely get rid of the dimension associated with the normal direction:

$$\mathbf{v}(v)_k = \begin{bmatrix} \mathbf{u}_{k+1} & \mathbf{u}_k \end{bmatrix} \begin{bmatrix} \frac{1}{\|\mathbf{u}_{k+1}\|} & 0 \\ 0 & \frac{1}{\|\mathbf{u}_k\|} \end{bmatrix} \frac{1}{\cos\hat{\alpha}_{k+1}\sin\hat{\alpha}_k - \cos\hat{\alpha}_k\sin\hat{\alpha}_{k+1}}$$
$$\cdot \begin{bmatrix} \sin\hat{\alpha}_k & -\cos\hat{\alpha}_k \\ -\sin\hat{\alpha}_{k+1} & \cos\hat{\alpha}_{k+1} \end{bmatrix} \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \begin{bmatrix} \cos\hat{\alpha}_{i+1} & \cos\hat{\alpha}_i \\ \sin\hat{\alpha}_{i+1} & \sin\hat{\alpha}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix}$$

Note that the summation goes around all the outgoing edges in clockwise order and that the indices i should be understood modulo the valence $|\mathcal{N}(v)|$ of the vertex v. Thus, the interpolation in the flattening plane can be reformulated in the following way:

$$\begin{aligned} \hat{\mathbf{v}}(v) &= \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \begin{bmatrix} \hat{\mathbf{u}}_{i+1} & \hat{\mathbf{u}}_i \end{bmatrix} \begin{bmatrix} \|\mathbf{u}_{i+1}\| & 0 \\ 0 & \|\mathbf{u}_i\| \end{bmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \\ &= \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \hat{\theta}_i \left(\hat{\mathbf{u}}_{i+1}\|\mathbf{u}_{i+1}\|a_i + \hat{\mathbf{u}}_i\|\mathbf{u}_i\|b_i \right) \\ &= \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \left(\hat{\theta}_{i-1}a_{i-1} + \hat{\theta}_i b_i \right) \|\mathbf{u}_i\|\hat{\mathbf{u}}_i \end{aligned} \tag{5.3}$$

Hence, the summation can be formulated as a weighted average of the vectors $\|\mathbf{u_i}\|\hat{\mathbf{u}}_\mathbf{i}$ where the weight w_i is equal to $\hat{\theta}_{i-1}a_{i-1} + \hat{\theta}_i b_i$. Define

$$\tilde{\mathbf{u}}_\mathbf{i} = \|\mathbf{u_i}\|\hat{\mathbf{u}}_\mathbf{i} \tag{5.4}$$

then the final interpolation formula for the interpolated vector at a vertex v in triangle k is

$$\mathbf{v}(v)_k = \begin{bmatrix}\mathbf{u}_{k+1} & \mathbf{u}_k\end{bmatrix} \begin{bmatrix}\tilde{\mathbf{u}}_{k+1} & \tilde{\mathbf{u}}_k\end{bmatrix}^{-1} \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \left(\hat{\theta}_{i-1} a_{i-1} + \hat{\theta}_i b_i\right) \tilde{\mathbf{u}}_\mathbf{i}.$$

Deferring the multiplication with $[\mathbf{u}_{k+1}, \mathbf{u}_k]$, we end up with the interpolated vector at vertex v in triangle k represented in barycentric coordinates $(a(v)_k, b(v)_k)^T$ with respect to the wedge $(\mathbf{u}_{k+1}(v), \mathbf{u}_k(v))$:

$$\begin{pmatrix} a(v)_k \\ b(v)_k \end{pmatrix} = \begin{bmatrix}\tilde{\mathbf{u}}_{k+1}(v) & \tilde{\mathbf{u}}_k(v)\end{bmatrix}^{-1} \frac{1}{2\pi} \sum_{i \in \mathcal{N}(v)} \left(\hat{\theta}_{i-1} a_{i-1} + \hat{\theta}_i b_i\right) \tilde{\mathbf{u}}_\mathbf{i}(v) \tag{5.5}$$

This is useful to speed up the final barycentric interpolation. Let v_0, v_1 and v_2 denote the three vertices of the triangle k and let $\mathbf{x}(v_0)$, $\mathbf{x}(v_1)$ and $\mathbf{x}(v_2)$ denote their respective positions in three dimensional Euclidean space. Further let $\mathbf{x}$ denote the point in the triangle at which the vector should get interpolated and $(a, b, c)^T$ denotes its barycentric coordinates with respect to the barycentric basis $\{\mathbf{x}(v_0), \mathbf{x}(v_1), \mathbf{x}(v_2)\}$. The interpolated vector $\mathbf{v}(\mathbf{x})$ is then given by

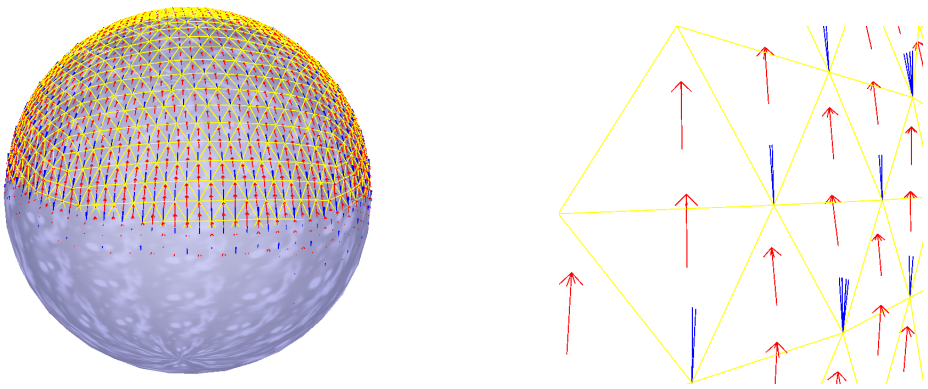
$$\begin{aligned}
\mathbf{v}(\mathbf{x}) &= \begin{bmatrix}\mathbf{v}(v_0)_k & \mathbf{v}(v_1)_k & \mathbf{v}(v_2)_k\end{bmatrix} \begin{pmatrix} a & b & c \end{pmatrix}^T \\
&= \begin{bmatrix}\mathbf{x}(v_0) & \mathbf{x}(v_1) & \mathbf{x}(v_2)\end{bmatrix} \\
&\quad \cdot \begin{bmatrix} -a(v_0)_k - b(v_0)_k & b(v_1)_k & a(v_2)_k \\ a(v_0)_k & -a(v_1)_k - b(v_1)_k & b(v_2)_k \\ b(v_0)_k & a(v_1)_k & -a(v_2)_k - b(v_2)_k \end{bmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}
\end{aligned}$$

which requires only two matrix-vector products to evaluate. See Figure 5.6 for an example of the vertex vector interpolation.

Another performance gain can be achieved by observing that a separate thread can compute the flattening of the triangle fans of the simplicial complex as soon as the next frame of the animation of the simulation domain is available. For a statically known animation, it would even be possible to precompute and store the flattening for every single frame. But nowadays, animations usually consist of multiple individual smaller animations which get blended together to dynamically create new animations at runtime. In such a case, the flattening can not be precomputed, but the next frame of the animation might still be known soon enough to start the flattening computation in a separate thread.

Artifact-free Barycentric Interpolation

There is one problem with the approach described above. We observed that if the interpolated velocities $\mathbf{v}(v_0)_k$, $\mathbf{v}(v_1)_k$ and $\mathbf{v}(v_2)_k$ at the three vertices of a triangle k are used for the barycentric velocity interpolation then stable states with non-vanishing heightfield gradients can occur. Note that the velocity at a vertex might interpolate to zero although the velocities of the surrounding faces are non-zero but are all pointing inwards or outwards w.r.t. the vertex. If



(a) **(b)**

Figure 5.6.: *An example of the local flattening operation.*
(a) *A spherical simulation domain with the velocity vector field at faces in red and the interpolated velocity field at vertices in blue. The simulation domain is visualized as a yellow mesh.*
(b) *Close-up view of the velocity interpolation at a vertex. The interpolated velocity in the parameterization plane at a vertex is mapped to each adjacent triangle at the vertex using the inverse of the linear angle ratio preserving transformation of the adjacent triangle. Due to the curvature of the sphere, these vectors do not lie in a common plane and hence get visualized as a cone of line segments.*

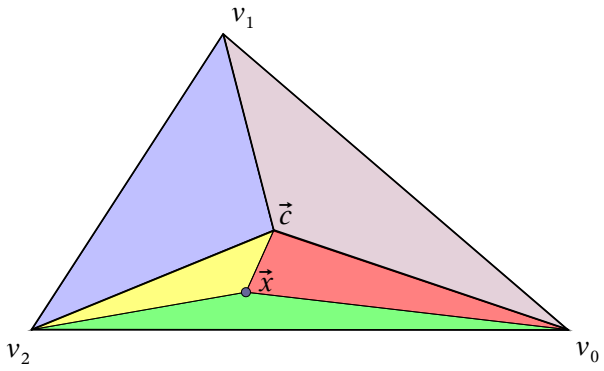


Figure 5.7.: *The smallest barycentric coordinate of a point* $\mathbf{x}$ *determines in which subtriangle the point is situated. In this example, the smallest coordinate is associated with the vertex* v_1*. The barycentric coordinates of the point w.r.t. the subtriangle can then be computed very efficiently using the barycentric coordinates w.r.t. the original triangle.*

this case applies to all the vertices of a triangle then the velocity in this triangle is wrongly assumed to be zero. Therefore the velocity at the face barycenter has to be taken into account, too. This makes more sense anyway, because using the non-interpolated values at the physical simulation nodes is conducive to avoid numerical damping which occur whenever interpolated values are used. Therefore, each triangle is subdivided into three smaller triangles made up by two vertices and the barycenter. The interpolated vector at a point inside such a sub triangle is computed using again barycentric interpolation with the two interpolated vertex vectors and the vector at the triangle barycenter. After the tracing, the interpolation point is given in barycentric coordinates w.r.t. the original triangle and thus these coordinates need to be converted to the appropriate sub triangle. The barycentric coordinates of this point facilitate to determine the correct subtriangle. The conversion of the barycentric coordinates $(a, b, c)^T$ to barycentric coordinates $(e, f, g)^T$ w.r.t. the subtriangle is straight forward, too. Assuming the point $\mathbf{x}$ is inside the subtriangle spanned by the barycenter $\mathbf{c}$ and the vertices v_0, v_2 (see Figure 5.7) and observing that the barycenter has barycentric coordinates $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ w.r.t. the original triangle, the conversion is derived using linear algebra:

$$\begin{aligned}\mathbf{x} &= \begin{bmatrix}\mathbf{x}(v_0) & \mathbf{x}(v_1) & \mathbf{x}(v_2)\end{bmatrix}\begin{pmatrix}a\\b\\c\end{pmatrix} = \begin{bmatrix}\mathbf{x}(v_0) & \mathbf{c} & \mathbf{x}(v_2)\end{bmatrix}\begin{pmatrix}e\\f\\g\end{pmatrix}\\ &= \begin{bmatrix}\mathbf{x}(v_0) & \mathbf{x}(v_1) & \mathbf{x}(v_2)\end{bmatrix}\begin{pmatrix}e+\frac{f}{3}\\ \frac{f}{3}\\ g+\frac{f}{3}\end{pmatrix}\end{aligned}$$

Hence the unknown barycentric coordinates are given by $f = 3b$, $e = a - b$ and $g = c - b$.

5.2.3. Forward Transportation

In Euclidean space, the interpolated vector at the position of the back traced point can simply be copied to the starting point of the tracing. But on a curved manifold, this is not possible because the local frame at a point traveling along a characteristic curve on the surface is expected to rotate with the Frenet Frame of this curve. The back tracing step discretizes this rotation by following at every triangle along the locally defined velocity vector at this triangle, see Figure 5.3. Note that this is in contrast to [SY04] where the velocity at the starting point is used throughout the whole tracing step. In [SY04], the velocity vector is rotated every time the tracing crosses an edge. At the end, the interpolated vector is transformed to the starting point by applying a linear transformation $\mathbf{T}$ to this interpolated vector. This linear transformation is derived using the velocity $\mathbf{v}_{start}$ and $\mathbf{v}_{end}$ at the start resp. end point of the tracing represented in three dimensional Cartesian coordinates, the normals $\mathbf{n}_{start}$ and $\mathbf{n}_{end}$ at the start resp. end triangle and the binormals $\mathbf{b}_{start} = \mathbf{v}_{start} \times \mathbf{n}_{start}$ and $\mathbf{b}_{end} = \mathbf{v}_{end} \times \mathbf{n}_{end}$. Note that $\mathbf{v}_{end}$ is the velocity used for the tracing in the end triangle and does not correspond to the interpolated vector. The linear transformation is then given by solving the system

$$\begin{bmatrix}\mathbf{v}_{start} & \mathbf{n}_{start} & \mathbf{b}_{start}\end{bmatrix} = \mathbf{T}\begin{bmatrix}\mathbf{v}_{end} & \mathbf{n}_{end} & \mathbf{b}_{end}\end{bmatrix}.$$

For a domain with zero curvature the linear transformation $\mathbf{T}$ results in the identity transformation because the local frames defined by the velocity, the normal and the binormal are identical. But note that in our approach, the start and end velocity are not the same even in a domain with zero curvature and hence, the above described approach is not applicable. Rather than rotating the velocity vector during the tracing and deducing a linear transformation at the end, a quaternion is carried along which sums up and stores all the rotations encountered while crossing edges. At the end, the interpolated vector is transformed to the start triangle by applying this quaternion.

5.3. Time Integration

After the Semi-Lagrangian advection step we end up with intermediate values for the water height and the velocity. According to the fractional step approach, these values are then used to integrate the remaining parts of the SWE. Several different explicit and implicit time integration methods were investigated to find a suitable integration method for these remaining parts of the SWE. The results are summarized in the following sections.

5.3.1. Implicit Time Integration

Unfortunately, the remaining part of the SWE is still nonlinear. There are two methods how to solve these nonlinear parts of the SWE in an implicit time integration scheme. Either an iterative method is used to solve the true non-linear system of equations or the system is linearized once over the whole time integration interval. If the non-linear system is solved by an iterative method, for example with the Newton-Raphson method, the system of equations must be linearized in each iteration by using the Jacobian of the system. But the computation and inversion of this Jacobian is too expensive for a real time application.

We implemented two methods which linearize the system over the whole time integration interval only once and thus only one matrix inversion per time step is required to solve the system. Both methods, namely an implicit Euler and a Newmark scheme, are presented in the next two subsections.

Implicit Euler

An implicit time integration scheme for the SWE has already been presented in [LvdP02], [KM90] and [WMT07]. While [KM90] and [LvdP02] both use a regular planar grid as a simulation domain, [WMT07] combines and extends the approaches of these antecedent schemes to irregular and curved triangle meshes. Our derivation is very similar to this latter one but deviates from it as we try to linearize as few terms as possible to stay closer to the original nonlinear SWE.

The derivation of our system matrix for the SWE assumes that the advection is handled by the Semi-Lagrangian method which provides intermediate values $\tilde{\eta}^{n-1}$ and $\tilde{\mathbf{v}}^{n-1}$. Similar to the aforementioned approaches, the derivation starts by taking the temporal derivative of the

advection-less continuity equation and substituting the emerging temporal first derivatives with the analytical advection-less terms given by the SWE:

$$\begin{aligned}\eta_{tt} &= \frac{d\eta_t}{dt} = -\frac{d(\eta\nabla\cdot\mathbf{v})}{dt} = -\eta_t\nabla\cdot\mathbf{v} - \eta\nabla\cdot\mathbf{v}_t \\ &= \eta(\nabla\cdot\mathbf{v})^2 - \eta\nabla\cdot\left(a_n\nabla h + \frac{1}{2}\eta\nabla a_n\right) \\ &= \eta(\nabla\cdot\mathbf{v})^2 - \eta\nabla a_n\cdot\nabla h - \eta a_n\nabla^2 h - \frac{1}{2}\eta\nabla a_n\cdot\nabla\eta - \frac{1}{2}\eta^2\nabla^2 a_n \\ &= \eta(\nabla\cdot\mathbf{v})^2 - \frac{3}{2}\eta\nabla a_n\cdot\nabla h - \frac{1}{2}\eta^2\nabla^2 a_n - \eta a_n\nabla^2 h - \frac{1}{2}\eta\nabla a_n\cdot\nabla H \end{aligned} \tag{5.6}$$

Note that the vertical acceleration a_n is due to external forces and is thus independent of the system variables. This acceleration can safely be assumed to be constant in the time interval $[t^{n-1}, t^n]$. The same holds for tangential forces $\mathbf{f}_\tau$ which can be handled by an explicit integration scheme and hence are not included in the following derivation.

Discretizing the temporal derivatives of the water height in the same way as in [KM90] gives

$$\begin{aligned}\eta_t^n &= \frac{\eta^n - \eta^{n-1}}{\Delta t} \\ \eta_{tt}^n &= \frac{\eta_t^n - \eta_t^{n-1}}{\Delta t} = \frac{\eta^n - 2\eta^{n-1} + \eta^{n-2}}{\Delta t^2}. \end{aligned} \tag{5.7}$$

Finally, an implicit equation for the new water height η^n is derived by substituting the second temporal derivative of the water height in Equation (5.7) with the expression given in Equation (5.6). This results in a non-linear implicit equation

$$\begin{aligned}2\eta^{n-1} - \eta^{n-2} &= \eta^n - \Delta t^2\eta_{tt}^n \\ &= \eta^n - \Delta t^2\Big(\eta^n(\nabla\cdot\mathbf{v}^n)^2 - \frac{3}{2}\eta^n\nabla a_n\cdot\nabla h^n - \frac{1}{2}\eta^n\eta^n\nabla^2 a_n - \eta^n a_n\nabla^2 h \\ &\quad -\frac{1}{2}\eta\nabla a_n\cdot\nabla H\Big).\end{aligned}$$

The vertical acceleration a_n is caused by external forces which normally vary slowly in space. Thus the spatial derivatives of the vertical acceleration are very small and can be ignored. The remaining non-linearities are eliminated by substituting variables where necessary by the intermediate values from the Semi-Lagrangian advection step $\tilde{\eta}^{n-1}$ and $\tilde{\mathbf{v}}^{n-1}$. The differential operators are discretized using the formulas presented in Subsection 4.4.2. We thus end up with the following linear, implicit equation

$$\left[\mathbf{A} + \Delta t^2\tilde{\eta}^{n-1}a_n\mathbf{D}\hat{\nabla}^2\right]\eta^n = 2\eta^{n-1} - \eta^{n-2} + \Delta t^2\eta^{n-1}a_n\mathbf{D}\hat{\nabla}^2 H$$

where $\mathbf{A} = \mathbf{I} - \Delta t^2(\nabla\cdot\tilde{\mathbf{v}}^{n-1})^2$ summarizes the diagonal matrices. As pointed out earlier, the matrix built from the cotangent weights $\hat{\nabla}^2$ is symmetric, but in the current system matrix, this matrix gets multiplied with a diagonal matrix and this product is no longer symmetric. Fortunately, by multiplying $[|\Delta t^2\tilde{\eta}^{n-1}a_n\mathbf{D}|]^{-1}$ from the left, we end up with a sparse and symmetric system

$$\begin{aligned}&\left[[|\Delta t^2\tilde{\eta}^{n-1}a_n\mathbf{D}|]^{-1}\mathbf{A} + \operatorname{sgn}(\Delta t^2\tilde{\eta}^{n-1}a_n\mathbf{D})\hat{\nabla}^2\right]\eta^n \\ &= \left[|\Delta t^2\tilde{\eta}^{n-1}a_n\mathbf{D}|\right]^{-1}\left(2\eta^{n-1} - \eta^{n-2} + \Delta t^2\eta^{n-1}a_n\mathbf{D}\nabla^2 H\right).\end{aligned} \tag{5.8}$$

The term $2\eta^{n-1} - \eta^{n-2}$ on the right hand side corresponds to an extrapolation of the vertical motion of the water height. As pointed out in [KM90], an artificial damping is given by introducing a damping parameter $s \in [0, 1]$ and reformulating this term as $\eta^{n-1} + (1 - s)(\eta^{n-1} - \eta^{n-2})$.

We observed that the term involving the divergence of the velocity renders the linearized system very fragile. This term caused the system matrix often to loose its positive definiteness. This prevents the usage of specialized and very efficient solvers. Furthermore, the water heights sometimes grew endlessly due to this term. Interestingly, an implicit system for the SWE is derived in [LvdP02] in a similar way, but the system contained this term in a slightly different form $\Delta t \eta \nabla \cdot \mathbf{v}$ on the right hand side of the implicit equation rather than in the diagonal of the system matrix. We also tried this approach, but without success. The resulting system was only stable for small values of the water depth and even then, the damping had to be fixed to the maximum ($s = 1$). Hence, similar to [KM90] and [WMT07], we were forced to ignore this term.

Ignoring the mentioned divergence term due to the velocity, the system (5.8) is positive definite and specialized solvers can thus be used. We investigated a sparse Cholesky factorization with back substitution from the Taucs library [1] and a conjugate gradient solver. The values at the last time step were used as an initial guess for the conjugate gradient algorithm. The Cholesky factorization was in average only slightly slower than the conjugate gradient algorithm but the latter algorithm sometimes required many iterations and sometimes did not even converge at all in an acceptable time. We thus decided to use the Cholesky factorization to solve our implicit system.

Newmark Integration

Implicit integration schemes gain increased stability because they introduce numerical damping. But for the implicit Euler scheme, this numerical damping impaired the quality of the simulation way too much (see Table 5.1). Since the implicit system (5.8) is only first order accurate, a second order accurate integration method might be very suitable to decrease damping artifacts. We investigated such an implicit time integration method, namely the Newmark integration scheme [Nem59]. It is known that integration schemes based on the Newmark method are very well suited for linear dynamic equations of motions of the form $\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{F}(t)$ where $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{K}$ denote the mass, the damping and the elastic stiffness matrix respectively. The SWE are not given in this form and thus a Newmark-type scheme is not directly applicable. But the SWE can be reformulated to match this form and the Newmark integration scheme for the SWE then looks like

$$\eta^{n+1} = \eta^n + \Delta t \eta_t^n + \frac{1}{2}\Delta t^2 \left((1 - 2\beta)\eta_{tt}^n + 2\beta\eta_{tt}^{n+1}\right)$$
$$\eta_t^{n+1} = \eta_t^n + \Delta t \left((1 - \gamma)\eta_{tt}^n + \gamma\eta_{tt}^{n+1}\right).$$

In addition to the water height η and the tangential velocities $\mathbf{v}$, the vertical velocity of the water height η_t and the vertical acceleration of the water height η_{tt} must be carried through time as well. The Newmark integration scheme prescribes a numerical scheme to advance the vertical velocity η_t in time. However, the SWE already provides an analytic solution for this

[1] www.tau.ac.il/ stoledo/taucs/

term $\eta_t = -\eta\nabla \cdot \mathbf{v}$ resp. $\eta_t = -\nabla \cdot (\eta\mathbf{v})$ if the water height is not advected. The analytic solution is advantageous since it allows to incorporate external forces via the tangential velocity $\mathbf{v}$ more easily as will be presented in Chapter 6. We hence decided to replace the numerical integration of the vertical velocity with the analytic solution given by the SWE. Note that the tangential velocities are easily computed when the new water heights are known because the temporal derivative of the tangential velocity depends only on the water height. An implicit Euler integration $\mathbf{v}^n = \mathbf{v}^{n-1} + \Delta t a_n \nabla h^n + \frac{1}{2}\eta^n \nabla a_n$ is used for the tangential velocity time evolution. Surprisingly, a more accurate implicit midpoint integration scheme $\mathbf{v}^n = \mathbf{v}^{n-1} + \frac{1}{2}(\Delta t a_n \nabla h^{n-1} + \frac{1}{2}\eta^{n-1}\nabla a_n + \Delta t a_n \nabla h^n + \frac{1}{2}\eta^n \nabla a_n)$ lead to an instable integration scheme.

A stable method results by choosing $\beta = \frac{1}{4}$ and $\gamma = \frac{1}{2}$. With this choice of parameter values, the Newmark integration scheme is also known as average acceleration method. Rearranging the terms, we get the following implicit equation

$$\eta^{n+1} - \frac{1}{4}\Delta t^2 \eta_{tt}^{n+1} = \eta^n + \Delta t \eta_t^n + \frac{1}{4}\Delta t^2 \eta_{tt}^n \,. \tag{5.9}$$

The vertical accelerations η_{tt}^{n+1} and η_{tt}^n are again replaced by the linearized and discretized version of the analytical derivative (5.6).

5.3.2. Explicit Time Integration

As stated in Equation (3.11) and Equation (3.14), the conservation of mass admits two formulations, one involving the divergence only:

$$\frac{\partial \eta}{\partial t} = -\nabla \cdot (\eta\mathbf{v}) \,, \tag{5.10}$$

and one which separates the advectional part with a material derivative:

$$\frac{D\eta}{Dt} = \frac{\partial \eta}{\partial t} + \mathbf{v} \cdot \nabla\eta = -\eta\nabla \cdot \mathbf{v} \tag{5.11}$$

Because the Semi-Lagrangian advection involves an interpolation step, the conservation of mass is not guaranteed when the water height η is advected. In contrast, solving the formulation involving only the divergence and no advection ensures the conservation of mass perfectly. Experiments showed that this advection-free divergence formulation of the continuity equation does not harm the stability. Thus, the first equation is solved directly without reformulating it with a material derivative to separate the advectional part. The Semi-Lagrangian advection is applied only to the velocity which gives an intermediate velocity field $\tilde{\mathbf{v}}^{n-1}$. This also saves some computational time because no characteristic curves starting from the vertices must be traced back.

The discretization of the divergence at a vertex requires the computation of the fluxes across dual cell boundaries. The flux of the water height across the dual edge emerging from the midpoint of the two vertices i and j and heading to the triangle barycenter is captured by $\frac{3}{A_k}(\frac{1}{2}(\eta_i + \eta_j)\mathbf{v}_k) \cdot \mathbf{R}_{-\frac{\pi}{2}}(\mathbf{c} - \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j))$, i.e., the average water depth of the vertices adjacent to the primal edge is used. This scheme ensures that the flux across the dual edge viewed from vertex i equals the negative flux across the very same edge viewed from vertex j. Hence, the flux is

locally and globally conserved. There is one more important issue about the discretization of Equation (3.11), though. Although the assumptions of the SWE turned the continuity equation into a differential equation involving the flux of the water height, the conserved quantity actually corresponds to the mass. But if the flux of the water height is used, then the conserved quantity corresponds to the water height rather than the mass. It is the flux of the mass across a dual edge that has to be considered. Given the dual edge between the vertices i and j in a triangle k then the mass flux across this edge is given by

$$\frac{3}{A_k}(m_{k,ij}\mathbf{v}_k) \cdot \mathbf{R}_{-\frac{\pi}{2}}(\mathbf{c} - \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j))$$

where $m_{k,ij} = \frac{1}{2}(\eta_i + \eta_j)\frac{A_k}{3}\rho$. As usually, ρ denotes the density and A_k denotes the triangle area. The total divergence $\nabla \cdot (m\mathbf{v})$ at a vertex is then given by a summation of the mass fluxes across all the dual cell boundaries, analog to Equation (4.6). This scheme perfectly ensures the conservation of mass since the mass fluxes sum up to zero. This fact is very important for a plausible fluid simulation.

A simple integration scheme is the explicit Euler method which, in this case, is

$$\eta^n = \eta^{n-1} - \frac{\Delta t}{A\rho}\nabla \cdot (m^{n-1}\tilde{\mathbf{v}}^{n-1}) \tag{5.12}$$

$$\mathbf{v}^n = \tilde{\mathbf{v}}^{n-1} + \Delta t \left(a_n^n \nabla h^n + \frac{1}{2}\eta^n \nabla a_n^n + \frac{1}{\rho}\mathbf{f}_\tau \right), \tag{5.13}$$

where $\nabla \cdot (m^{n-1}\tilde{\mathbf{v}}^{n-1})$ corresponds to the aforementioned divergence of the product between the mass and the velocity, A corresponds to the dual cell area and ρ is the fluid density. The division renders the mass fluxes into the proper height difference such that the mass is conserved. Note that since the temporal derivative of the velocity does not depend on the velocity itself, this explicit Euler method basically corresponds to a leapfrog scheme.

As badly shaped triangles might appear due to the animated simulation domain, the explicit Euler integration is not unconditionally stable. Therefore, the time step has to be chosen according to the current size and shape of the triangles of the simulation domain. The stability of the explicit Euler scheme is improved in the following way. After flattening the simulation domain and advecting the velocity with the Semi-Lagrangian method once, multiple sequential explicit Euler time integrations are performed with a time step equal to $\frac{\Delta t}{k}$ where k is the number of sequential substeps (see also Algorithm 1 in Appendix B). Three such sequential substeps proved to be optimal for all the meshes used in this thesis. Nevertheless, in certain frames of the simulation domain animation, a triangle might be so badly shaped that the divergence and the gradients had to be limited by an upper bound, as has been described in Subsection 4.4.2. According to our experiments, such a limitation does not seem to harm the visual plausibility of the simulation. Note that even implicit methods have problems with badly shaped triangles since the condition number of the system matrix depends on the shapes of the triangles.

5.3.3. Comparison of Integration Schemes

The implicit Euler integration scheme (5.8) is unconditionally stable. Hence, arbitrary large time steps and water depths can be chosen with the implicit Euler integration and the system

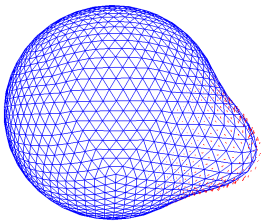
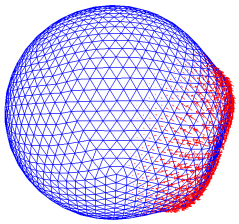
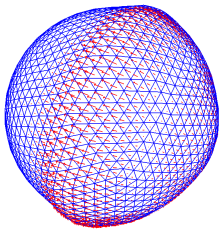
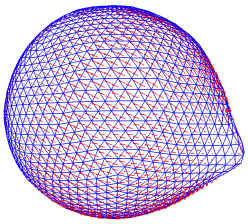
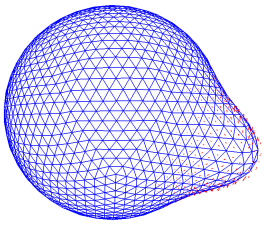
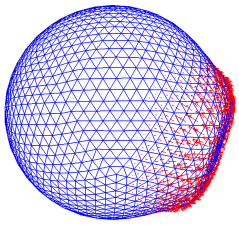
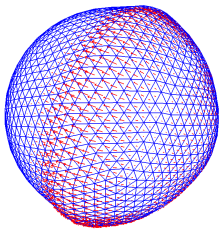
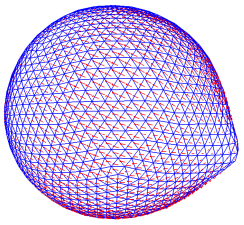
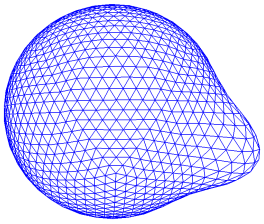
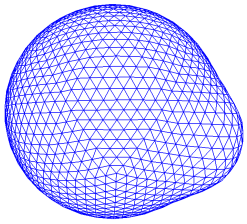
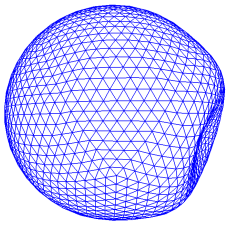
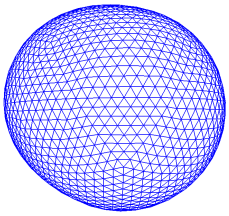
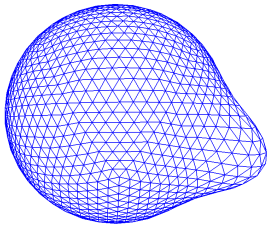
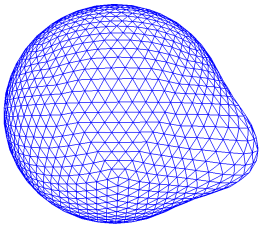
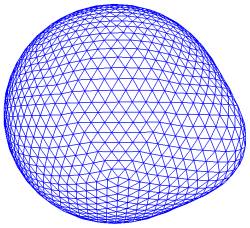
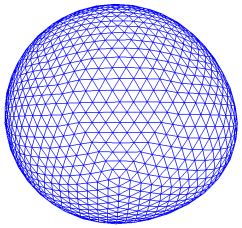
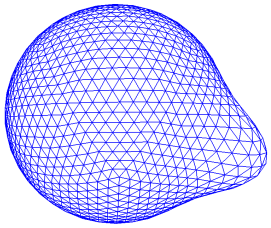
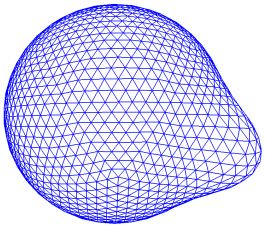
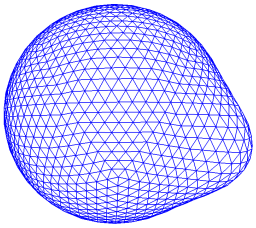
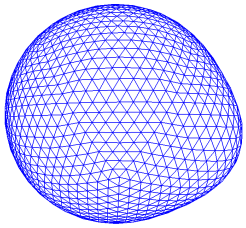
2nd Frame	8th Frame	27th Frame	141th Frame
Explicit Euler			
			
Implicit Newmark			
			
Implicit Euler ($s = 0$)			
			
Implicit Euler ($s = 0.5$)			
			
Implicit Euler ($s = 1$)			
			

Table 5.1.: *Comparison of the various temporal integration methods. The water surface is visualized as a blue grid to not obscure the geometry due to shading effects and the tangential velocity is visualized with red arrows when available. As an initial condition, a displacement is added to the spherical simulation domain. The time step was set to* $\Delta t = \frac{1}{60}$.

remains stable. However, a short glance at Table 5.1 reveals that the explicit Euler and the Newmark integration scheme both yield very pleasing and similar results. In contrast, the implicit Euler integration suffers from too much numerical damping, although this damping is slightly alleviated when the damping parameter s is set to a small value. But the implicit Euler method is still unsuitable to integrate the SWE in time.

In general, the time step size in a real time application is quite small to guarantee a high frame rate and hence, the explicit Euler scheme is very stable. Considering in addition the fact that the explicit integration is about five times faster (including the three sequential substeps) than an implicit integration, the explicit Euler integration is clearly the method of choice to advance the remaining parts of the SWE in time. Furthermore, linearized implicit schemes can not ensure the conservation of mass since this is described by a nonlinear equation.

6

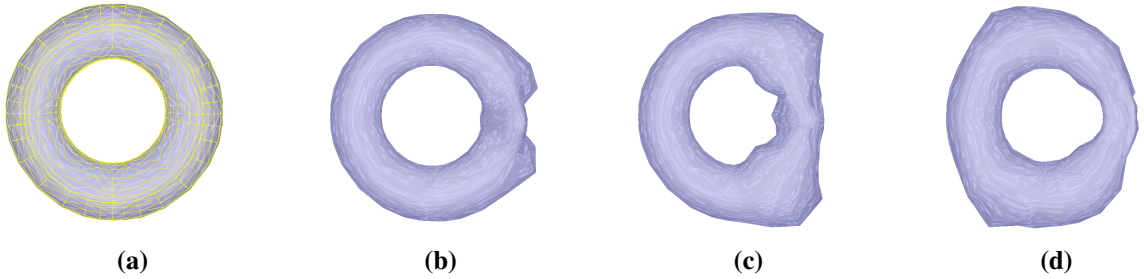
Coupling with Embedding Space

The presented simulation discretizes the SWE entirely on the discrete manifold, i.e., the tangent plane and its orthogonal direction define a local coordinate system at every point and hence there is no notion of a global three dimensional Cartesian coordinate system required. Nevertheless, effects like collision handling or a filling process require a global coordinate system to define globally consistent representations of vectors. This raises the question of how to relate the three dimensional embedding space with the two dimensional manifold. Fortunately, the general formulation of the SWE in Equation (3.15) allows the coordinate system of the embedding space to be coupled to the two dimensional manifold using an external force or acceleration field. A detailed description how this coupling can be done is given in Section 6.1. This chapter then proceeds by presenting several interesting effects which can be achieved with the current simulation framework. Namely collision handling, a filling effect, and the animation of the simulation mesh itself are presented.

6.1. External Forces

It is important to recall that according to the derivation based on the balance equations (3.1) and (3.2) the external force acting on the fluid is defined to be a force density. To get a better grasp of this definition, consider for the moment a force field $\mathbf{f}$ not defined as a force density. The acceleration at a triangle i would then be given by $\mathbf{a}_i = \frac{\mathbf{f}_i}{\rho \eta_i A_i}$. Hence, a larger acceleration would be exerted at small triangles with small water depths due to the division by the triangle area and water depth. This would lead to unexpected and wrong results because the resulting acceleration would depend on the triangulation of the simulation domain.

Although the projection during the SWE derivation in Equation (3.8) transforms the volume



(a) (b) (c) (d)

Figure 6.1.: *An external force is exerted from the right side. Image (a) shows the triangulation of the mesh. Images (b), (c) and (d) show several frames of the animation sequence.*

force density $\mathbf{f}$ into an area force density $\eta\mathbf{f}$, this projection is undone in Equation (3.13) by dividing through the water depth η. Therefore, even though the simulation domain is only two dimensional, a volume force density has to be used. The external acceleration of triangle i due to this force density is then given by $\mathbf{a}_i = \frac{\mathbf{f}}{\rho}$.

Accelerations due to an external force density can not simply be integrated and added to the velocity field of the fluid because the velocity field is restricted to a two dimensional manifold whereas an external force field is a true three dimensional vector field. Doing so would correctly handle the tangential part of the acceleration, but would ignore the component of the acceleration which is perpendicular to the fluid simulation domain. The general derivation of the SWE in Equation (3.14) and Equation (3.15) allows a proper handling of this perpendicular acceleration component by splitting the acceleration $\mathbf{a}$ into a normal component $a_n = \mathbf{a} \cdot \mathbf{n}$ and a tangential component $\mathbf{a}_\tau = \mathbf{a} - a_n\mathbf{n}$. This normal component is then added to the vertical acceleration of Equation (3.15) which consists amongst others of the global gravitational acceleration g. Figure 6.1 shows an example of an external force density applied to a torus shaped simulation domain.

Note that in [WMT07], similar ideas to handle external forces are presented. Namely, this paper also proposes to split the external forces into a normal and tangential component. However, the derivation is slightly different and leads to the acceleration term $\frac{\nabla p}{\rho} = a_n\nabla\eta + \eta\nabla a_n$ which lacks an additional factor of $\frac{1}{2}$ in the second term compared to our term $a_n\nabla\eta + \frac{1}{2}\eta\nabla a_n$. Note that this latter term is derived from the divergence of the diagonal term in Equation (3.12), i.e., $a_n\nabla\eta + \frac{1}{2}\eta\nabla a_n = \frac{1}{\eta}\nabla \cdot (\frac{1}{2}\mathbf{I}a_n\eta^2)$.

6.2. Collision Handling

Our shallow water simulation also handles collisions with other rigid shapes. This collision handling is presented in two parts. The first subsection is concerned with the detection of collisions whereas the second subsection then describes the collision response.

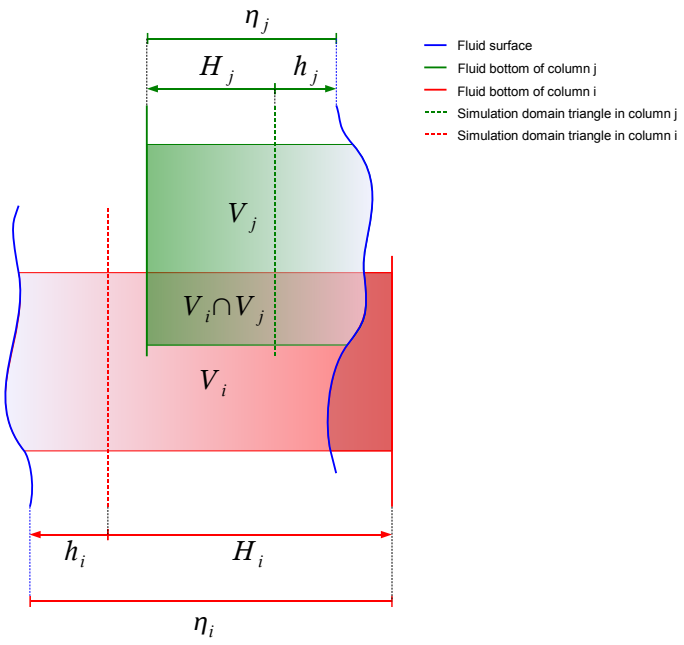


Figure 6.2.: *Fluid columns V_i (red) and V_j (green) of two opposite triangles i and j might intersect each other. The fluid bottom H_i of a column could even extend farther than the fluid surface η_j of the opposite fluid column. Such a fluid column thus virtually occupies empty space. The triangle mesh of the simulation domain, which also serves as the zero-level of the fluid (see Figure 3.1), is visualized as dashed lines.*

6.2.1. Collision Detection

The most obvious idea to detect a collision is to test whether the volume of a collision candidate intersects the water volume represented by the SWE heightfield. But this approach will not work due to the fact that the SWE are simulated on a closed two dimensional manifold. Consider Figure 6.2. The water surface in equilibrium is defined to correspond exactly to the triangle mesh which also serves as simulation domain. Hence, the water volume virtually extends into the interior of the mesh. Because the simulation domain is a closed mesh, the virtual water volumes from two opposite triangle faces i and j might intersect each other, indicated in Figure 6.2 as the volume $V_i \cap V_j$. Such a water column could even extend through the fluid surface of the other side, as indicated with the slightly darker color of the volume V_i. Simply applying the above mentioned collision detection would mean that a collision might be detected when a collision candidate enters such a virtual volume although there is no water visible at all at this point. This obviously would look very odd. Furthermore, the buoyancy causes another problem in the volume of the mutual intersection $V_i \cap V_j$. Normally, buoyancy gives rise to a force pointing upwards to the water surface but there is no unique definition of this direction in the intersection volume.

Considering these facts we chose to use the following heuristic. There is a valid collision if the potential collision partner indeed intersects the water volume but is *not* completely below the water surface. An exact volume intersection test between the fluid and the solid is quite expensive. Hence the following approximative intersection test is used, see Figure 6.3 for a graphical

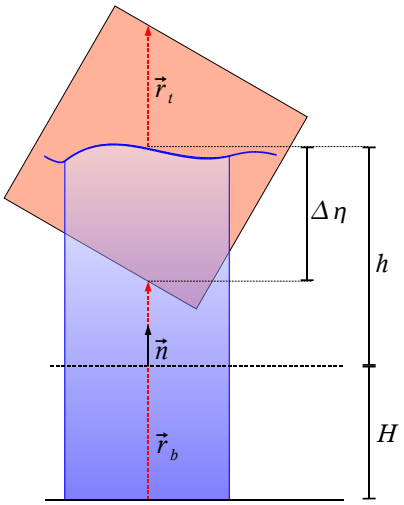


Figure 6.3.: *An intersection is defined to be only valid if the solid is not completely below the fluid surface. This is easily verified by intersecting the solid with a ray $\mathbf{r}_b$ emanating from the fluid bottom in normal direction and with a ray $\mathbf{r}_t$ emanating from the fluid surface in normal direction. The normal $\mathbf{n}$ is given by the triangle normal of the simulation domain.*

sketch. First, the volume of the potential collision partner is tested against an intersection with a water column by intersecting the solid volume with a ray $\mathbf{r}_b$ emerging from the virtual fluid bottom in normal direction of the simulation domain. If there is an intersection point within the fluid column a second ray $\mathbf{r}_t$ is sent in normal direction starting from the water surface. This second ray intersection test detects whether or not the volume of the solid is completely below the water surface. The ray-solid intersection tests are performed using the PhysX library of Ageia [1]. We perform some coarse culling steps to speed up the collision detection. Of course, there is room for improvement in using more sophisticated culling techniques.

Note that this heuristic does not provide a bulletproof solution to the aforementioned problems. Nevertheless, the heuristic resolves the otherwise distracting cases and is a good trade off between efficiency and accuracy. For the effects we plan to achieve, i.e., throwing obstacles at and through the fluid mesh, the heuristic is sufficient.

6.2.2. Collision Response

The penalty force acting on a submerged solid due to the fluid basically consists of two individual forces, a buoyancy and a drag force. The drag force of a solid is caused due to friction and pressure forces. The drag force is zero when the velocities of both the fluid and the solid are equal and hence can intuitively be understood as kind of a viscous force between the solid and the fluid. The buoyancy force causes the solid to float or sink, depending on the density ratio between the fluid and the solid. The penalty force of the solid is computed by first evaluating the drag and buoyancy force restricted to each individual fluid column and then summing up these forces over all the fluid columns.

[1] www.ageia.com/physx

The collision response can be applied to the heightfield of the shallow water fluid in two different ways. Either the height of the water column is reduced explicitly by the submerged volume of the submerged solid or the water height is changed implicitly by applying a penalty force to the velocity field. Ensuring mass conservation is very difficult with the former approach. Hence, the latter approach was chosen in our framework to model the collision response. The penalty force acting on a fluid column is also given by the sum of two forces. The first force equals the drag force of the solid restricted to this fluid column in opposite direction. In contrast, the second force is slightly different than the buoyancy force of the solid, although the underlying idea is the same.

Please consult Figure 6.3 and Figure 6.4 for a graphical sketch of the terms involved in the penalty force. Assume the fluid penetrates the solid by a depth equal to $\Delta\eta$. The following heuristic is then used to push the fluid out of a solid. According to Archimedes' principle, this gives rise to a force equal to $\mathbf{f} = \rho g A \Delta\eta$ where ρ is the fluid density, A is the triangle area and g is the gravitational acceleration. This force is then transformed to a force density

$$\mathbf{f} = \frac{\rho g A \Delta\eta}{A\eta} = \frac{\rho g \Delta\eta}{\eta}$$

and applied to the fluid column in negative normal direction. The water height at the triangle is denoted here as η.

Even though the buoyancy force of the solid is also dictated by Archimedes' principle, this force is slightly trickier to compute since we push the water out of the solid and the actual immersion depth of the solid is thus not equal to the above mentioned penetration depth. To compute the actual immersion depth d, the fluid height right next to the submerged solid must be known. We compute this height by simply looking for the maximum water height at a fluid surface vertex of a fluid column which collides with the solid. The buoyancy force of the solid due to a single fluid column i, which collides with the solid, is thus given by

$$\mathbf{f}_i = \rho g A_i d.$$

The drag force can either be defined as a linear or as a quadratic approximation w.r.t. the relative velocity $\mathbf{v}_{rel} = \mathbf{v}_{solid} - \mathbf{v}_{fluid}$. The linear approximation is commonly used when the relative velocity is small whereas the quadratic approximation is used at high speeds. However, our experiments showed that the latter approximation produces larger waves which look more appealing. Hence, we normally used the quadratic approximation

$$\mathbf{f}_{drag} = -\frac{1}{2} C \rho A_{\perp} \|\mathbf{v}_{rel}\|^2 \frac{\mathbf{v}_{rel}}{\|\mathbf{v}_{rel}\|} \tag{6.1}$$

where C is a user definable drag coefficient and $A_{\perp}$ denotes the surface area of the solid perpendicular to the relative velocity inside the fluid column under consideration. This area is computed in the following way. We assume that the surface of the object at the intersection point between the ray $\mathbf{r}_b$ and the surface of the solid can be adequately approximated with the tangent plane whose orientation is given by the normal $\mathbf{n}_{ip}$ of the solid surface at the intersection point (see also Figure 6.4). Hence, the surface area A_{proj} of the colliding object inside the fluid column is approximately equal to the area A of the simulation domain triangle projected onto

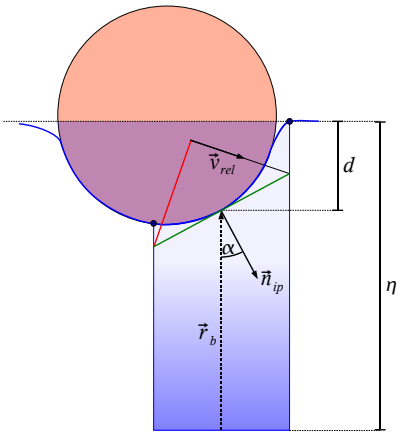


Figure 6.4.: *The actual immersion depth d of the solid is readily computable once the fluid height η at a vertex right next to the submerged solid is known. The submerged volume of the solid is highlighted with a slight blue shading. The drag force in a fluid column is proportional to the surface area $A_\perp$ of the solid perpendicular to the relative velocity $\mathbf{v}_{rel}$. This area is depicted in red color. We approximate the surface area of the solid inside the fluid column by projecting the triangle of the simulation domain onto the tangent plane which is given by the intersection normal $\mathbf{n}_{ip}$ and the intersection point between the bottom ray $\mathbf{r}_b$ and the fluid surface (thick blue curve). This projected area is visualized in green color.*

this tangent plane

$$A = A_{proj} \cos\alpha \quad \rightarrow \quad A_{proj} = \frac{A}{\cos\alpha}$$

where $\cos\alpha = \mathbf{n} \cdot \mathbf{n}_{ip}$ corresponds to the cosine of the angle between the triangle normal and the intersection normal. There is one problem with this approach, though. For angles α close to $\frac{\pi}{2}$, a major fraction of the area of the projected triangle on the tangent plane might poke out either below or above of the fluid column. Therefore, the projected triangle has to be intersected against the fluid bottom and the fluid top of the column and its area has to be reduced by the truncated parts of the projected area. Finally, the projected area then needs to be foreshortened in the direction to the relative velocity

$$A_\perp = \mathbf{n}_{ip} \cdot \frac{\mathbf{v}_{rel}}{\|\mathbf{v}_{rel}\|} A_{proj}.$$

It is worth to point out that the drag force, which gets applied to the fluid column, must not be transformed to a force density since this force already depends on the volume of the fluid column and hence the size of the column is already handled implicitly.

6.3. Filling Effect

A filling effect should give the impression that the simulation domain gets slowly filled with water. Obviously we expect a global acceleration which pulls the fluid downwards. But such an effect is quite difficult to achieve with the SWE. To see why, consider the SWE without external

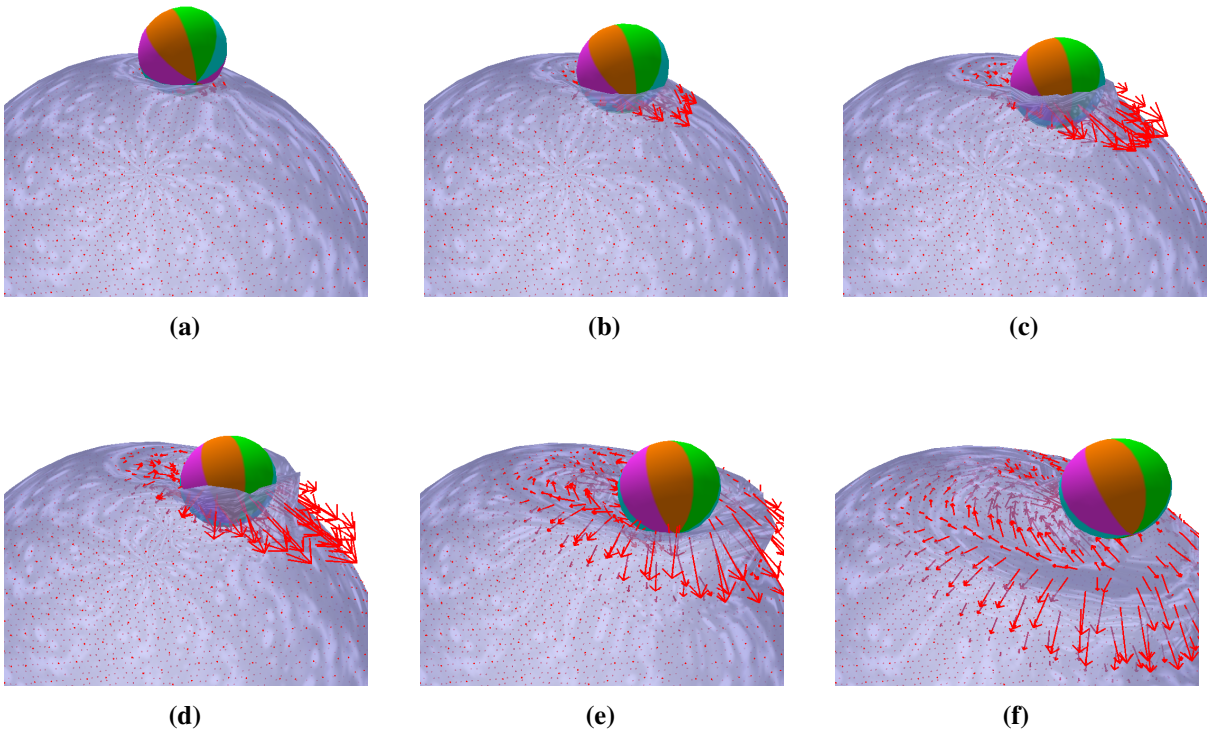
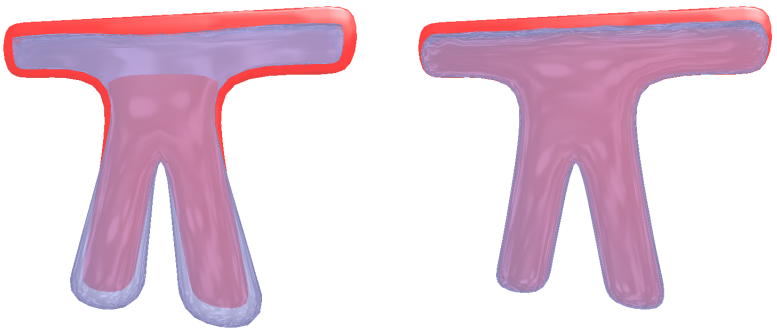


Figure 6.5.: *Example of the collision handling. A rigid body sphere is falling onto a sphere of water. The velocity field is visualized with red arrows. At the beginning the fluid velocity is vanishing everywhere. Note the emerging wave due to the tangential speed of the solid.*

forces. The simulation then reaches its equilibrium state when the height gradient ∇h vanishes everywhere, i.e., when the water height $h = \eta - H$ is uniform all over the simulation domain. Therefore a small droplet of water would spread all over the simulation domain until there is a thin film of water everywhere. Note that this happens regardless whether or not the simulation domain is curved because the simulation domain is always considered internally as flat.

In [WMT07], a surface tension force and a viscosity term is used to prevent a fluid drop from spreading all over the simulation domain. Although effects due to viscosity might not disturb the simulation of small droplets, a viscous force considerably impairs the simulation of a large body of an inviscid liquid. The surface tension force gives a droplet of water indeed its typical bulged shape, but this surface tension force is not suitable to enforce a steep water increase between empty and full cells because the surface tension force would have to be increased considerably to counteract the height gradient at the interface. Doing so would make the whole fluid bulge like a large droplet (see also Figure 7.2 in Section 7.2). Therefore, the surface tension force is not suitable for the filling effect because this effect should give rise to a large height gradient at the interface between empty and full cells. A different external force field thus has to be defined to impose an equilibrium state such that the fluid is gathered and stays in a certain region without spreading all over.

The filling process starts by introducing a source at a given dual cell. A source is simply implemented by adding a certain amount to the divergence of this dual cell. The simplest way to avoid that the fluid of this source spreads all over the mesh is to introduce a constant



(a) **(b)**

Figure 6.6.: *Comparison between a constant artificial acceleration (a) and the non-constant artificial acceleration depending on the water depth (b). The acceleration points downwards to make the fluid gather at the lower part of the simulation domain. The desired shape with a depth of α_{art} is highlighted in red color. Note that both examples use exactly the same amount of fluid. It is clearly visible that the constant acceleration is not able to reproduce a sharp interface between wet and dry cells whereas the non-linear artificial acceleration can reproduce a sharp interface separating the fluid with a depth of α_{art} from the dry areas.*

artificial acceleration $\mathbf{a}_{art}$ pointing toward the region where the fluid should gather. Note that this artificial acceleration is defined w.r.t. a three dimensional Cartesian coordinate system of the embedding space and does not correspond to the normal acceleration a_n which is defined w.r.t. the orthogonal direction at a triangle of the mesh. Such a constant artificial acceleration would mean that all the fluid is accelerated in this direction and hence, even that fluid gets accelerated which already is contained in the region where the fluid should gather. This not only looks implausible (see Figure 6.6(a) for an example) but might also introduce stability problems because a large acceleration might be required to counteract the arising height gradients at the interface between full and empty cells. Whenever the artificial acceleration $\mathbf{a}_{art}$ gives rise to a normal acceleration a_n that outweighs the gravity g (which acts in normal direction of the triangle) then the dynamics prescribed by the SWE become inadequate. Note that this problem is independent of the used time integration method. It is the PDE itself that amplifies a height gradient if the normal acceleration a_n is positive. A quick fix is to simply clamp the normal acceleration a_n to zero whenever the acceleration becomes positive. But this does not solve the fact that the simulation would look implausible due to the bulging.

We solved this problem by introducing a spatially non-constant and nonlinear artificial acceleration depending on the water height. The following function was chosen to define the magnitude of the artificial acceleration, see also Figure 6.7:

$$\|\mathbf{a}_{art}\| = e^{-\frac{h-\alpha_{art}}{\sigma_{art}}} - 1 \tag{6.2}$$

In this way, the artificial acceleration makes the fluid flow in the desired direction while nonetheless prescribing an equilibrium state at a water depth of α_{art} in the region where the fluid gathers. This exactly leads to the sought-after effect. The parameter σ_{art} controls the sharpness of interface between empty and full fluid cells. See Figure 6.6 for a comparison between the spatially constant and the spatially non-constant artificial acceleration.

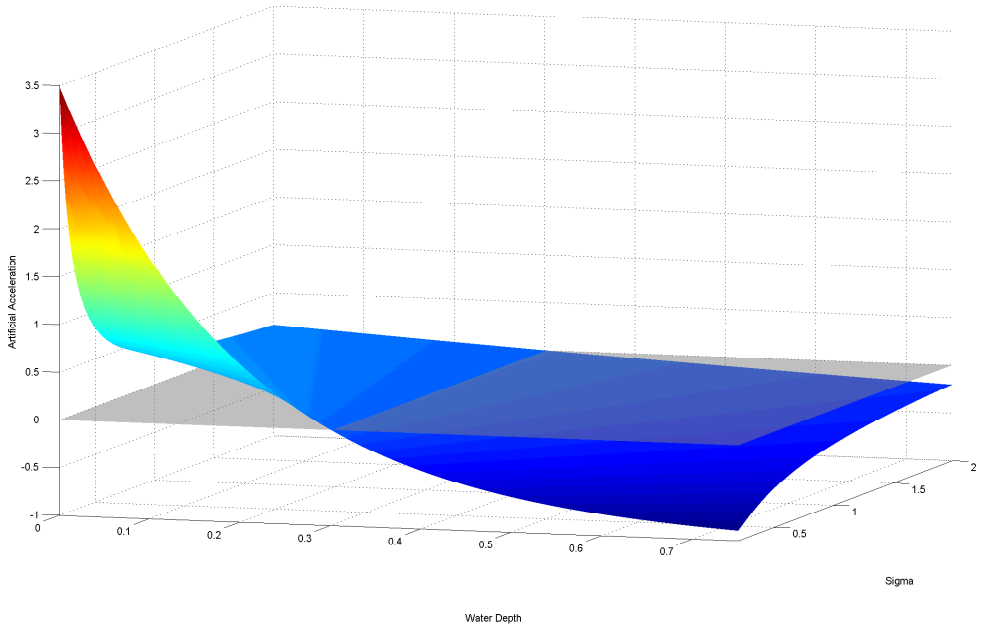


Figure 6.7.: *An artificial accelerations forces the fluid to gather at a certain region. The magnitude of this artificial acceleration is plotted against the water depth and its parameter σ which controls the steepness of the function. The function is designed in such a way that the fluid is in equilibrium when it has gathered in a region with a water depth of α_{art}. This parameter was set to a value of 0.3 in this plot. The zero level is highlighted with a transparent gray plane.*

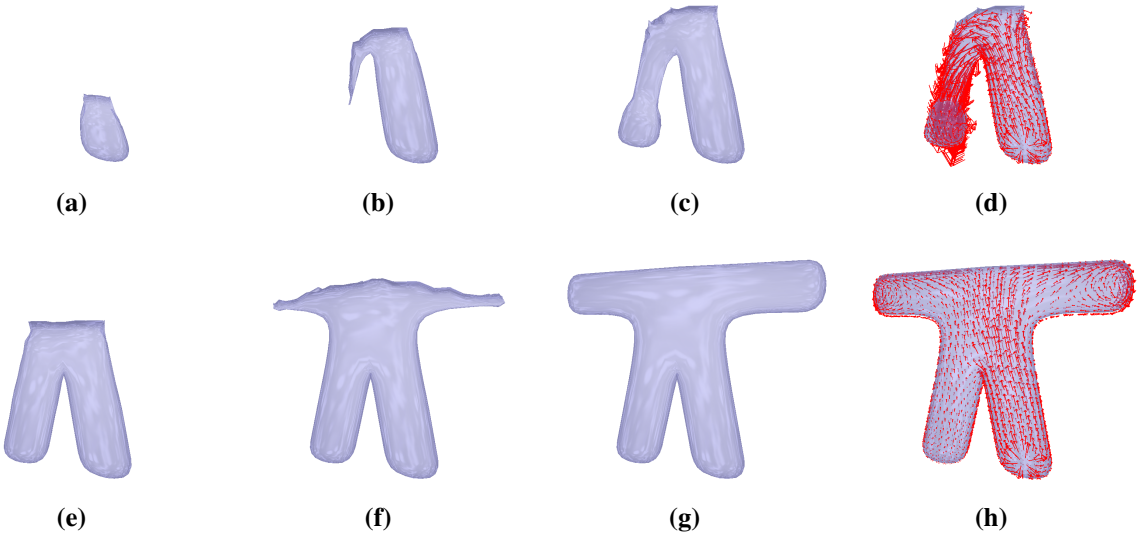


Figure 6.8.: *A source at the right leg first fills this leg until the fluid reaches the torso. At this point, the fluid starts two flow downwards and fills the second leg, thanks to the artificial acceleration which points downwards. Only after this leg is completely filled, the water level rises again to finally fill the whole figure. Note that no undesirable bulging appears throughout the sequence which is crucial for the impression of a three dimensional filling effect.*

A filling effect can now be simulated using this artificial acceleration. See Figure 6.8 for an example. It is remarkable that, although the simulation domain is actually two dimensional, we can achieve nonetheless such a three dimensional effect.

Although we only used the artificial acceleration to demonstrate a filling effect, this approach is quite general. The fluid can be gathered at any desirable shape, given the directions of a suitable control force field. The magnitudes of the accelerations due to these control forces can be computed using Equation (6.2).

6.4. Animation of Simulation Domain

A new feature of our SWE simulation is that the simulation domain itself is deforming. Our robust discretization and the general formulation of the SWE with external forces and spatially-varying accelerations in normal direction admit a straight forward adoption of the simulation domain animation. For simplicity, a skeleton based animation method [2] was chosen to move the vertices of the simulation domain through time. But it is important to point out that the simulation domain can be deformed and animated with any method of choice. Furthermore, note that the animation sequences are not required to be known statically because no pre-computations are necessary. Animations can therefore be generated on-the-fly during the simulation. For example several smaller animations can be blended together by using weighted sums which is common practice in compute games. The animation of the simulation domain implies several challenges which are presented in the following subsections.

6.4.1. Stability

Even though there might be no badly shaped triangles in the initial simulation mesh, it is almost unavoidable that such badly shaped triangles appear during the simulation sequence. This is troublesome for explicit as well as for implicit time integration methods. Remember that the gradient of a linear basis function is inverse proportional to the triangle height w.r.t. the associated sampling node of the basis function. Thus, the differential operators might give rise to very large values for elongated triangles. To avoid excessive overshooting in explicit methods and badly conditioned system matrices in implicit methods, the angles of the triangles are clamped at a certain lower and upper threshold, as has already been described. We did not observe that this harms the visual quality of the simulation.

6.4.2. Mass Conservation

The movement of the vertices of the simulation obviously induces a change in the triangle areas. Since the mass of a fluid column is proportional to the triangle area, this change in the areas must be handled somehow. The fluid mass associated with a dual cell k at time t^n is given by $m_k^n = \rho V_k^n = \rho A_k^n \eta_k^n$. Therefore, given the new dual cell area A_k^{n+1}, the conservation of mass

[2] home.gna.org/cal3d/

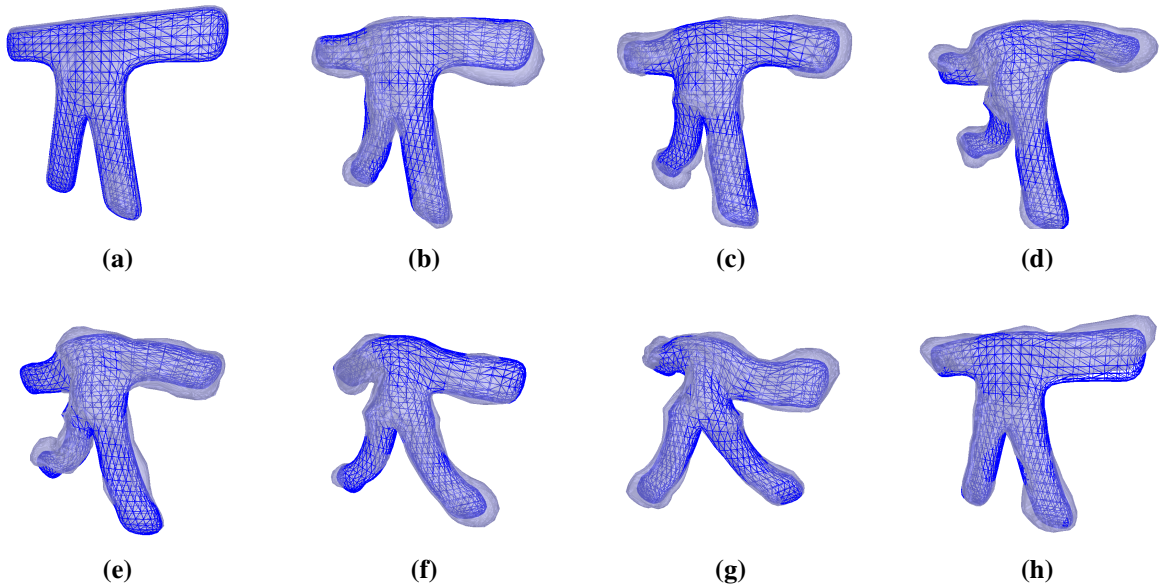


Figure 6.9.: *Several frames from an animation sequence with an animated simulation domain. The simulation domain is visualized as a blue grid. The fluid is following the grid and due to inertia effects, waves emerge which give the two dimensional fluid simulation a volumetric appearance.*

can then be ensured by changing the water height to

$$\eta_k^{n+1} = \frac{A_k^n \eta_k^n}{A_k^{n+1}}.$$

This simple and straight forward approach works very well and ensures mass conservation exactly.

6.4.3. Inertia

Effects implied by inertia are an important factor for a visual convincing fluid simulation. Hence, we would like to capture these kind of effects. Note that the animation of the simulation domain can be classified in two categories. On one side, the simulation domain is deforming non-rigidly as prescribed by the animation sequence. In an interactive environment on the other hand, the simulation domain can as well be attached to a controller which rigidly transforms the domain. We expect that both kind of motions should generate waves and make the fluid flow. Thanks to the general formulation of the SWE in Equation (3.14) and Equation (3.15), it is quite easy to couple the SWE with external accelerations due to such motions. The acceleration at a vertex v of the simulation mesh is approximated using a backward finite difference scheme

$$\mathbf{a}^n(v) = \frac{\mathbf{x}^n(v) - 2\mathbf{x}^{n-1}(v) + \mathbf{x}^{n-2}(v)}{\Delta t^2}. \tag{6.3}$$

Note that the inertia of the fluid acts against the direction of acceleration of the simulation domain and thus the negative of Equation (6.3) is actually used for further computations. This acceleration is then split again into a horizontal and a vertical part. The horizontal part is added

to the normal acceleration a_n in Equation (3.15) whereas the vertical part $\mathbf{a}_\tau$ is combined with the external tangential force $\mathbf{f}_\tau$ of Equation (3.15). In this way, the velocity field will take care of inertia effects.

There is one subtlety with this approach, though. We observed that the magnitude of the finite difference approximation to the acceleration of a vertex of the simulation domain can be quite large. This especially happens at the transition from a vanishing to a non-vanishing external acceleration. We solved this problem by smoothing the acceleration over several time steps

$$\mathbf{a}^n(v) = (1-\sigma)\frac{\mathbf{x}^n(v) - 2\mathbf{x}^{n-1}(v) + \mathbf{x}^{n-2}(v)}{\Delta t^2} + \sigma\mathbf{a}^{n-1}(v) \tag{6.4}$$

where σ is a parameter controlling the actual size of the smoothing window.

7

Results and Discussion

7.1. Results

Besides the images given throughout this thesis, please have a look at the accompanying videos to get a better impression of our simulation.

We implemented the previously described techniques using C++. Due to the irregular simulation domain the computations are entirely performed on the CPU, except the rendering, which is done on the GPU. Note that all the images and all the frames of the videos are captured from the actual real-time simulation: no offline rendering has been used. The performance was measured on a computer with an Intel^{TM}Core Duo 2.4 GHz CPU with 2GB of RAM. However, the current implementation is not yet multi threaded. Table 7.1 shows the computation times of the individual steps of our shallow water simulation for different meshes. The frame rate is an averaged value over several frames and can vary slightly because the time for the backtracing in the Semi-Lagrangian method depends on the magnitude of the velocities. But in general, the frame rate only varies by up to 2 and 3 frames. The animation column includes the time for the update of the vertex positions of the simulation domain using the skeleton based animation, the computation of new vertex and face normals and the areas of primal and dual cells. The flattening column lists the time required to compute the mapped edges $\tilde{\mathbf{u}}_\mathbf{i} = \|\mathbf{u}_\mathbf{i}\|\hat{\mathbf{u}}_\mathbf{i}$ according to Equation (5.1) and Equation (5.4). The time measurements for the velocity advection includes the velocity interpolation to the vertices according to Equation (5.3), the back tracing, the velocity interpolation given in Equation (5.5) and the forward transportation using the quaternion. The height integration and the velocity integration corresponds to the time integration given in Equation (5.12) resp. Equation (5.13). Finally, the last columns lists the time to decompose the updated velocity vectors into the barycentric representation w.r.t. the triangle to which they belong to. As previously mentioned, due to the time step restriction of the explicit

Model	Resolution Vertices / Triangles	Frames per Second	Animation	Flattening	Velocity Advection	Height Integration	Velocity Integration	Barycentric Velocity Computation
Torus (Fig 6.1)	768 / 1536	75.5	2.0	1.95	1.3	1.47	1.41	0.33
Figure (Fig 6.9)	1346 / 2688	48.7	3.79	3.4	2.36	2.58	2.43	0.57
Bunny (Fig 7.1 a)	2461 / 4918	33.6	6.9	6.4	4.6	3.75	4.53	1.08
Sphere (Fig 6.5)	2562 / 5120	33.5	6.2	6.4	4.5	3.66	4.65	1.11
Letters (Fig 7.1 b)	3832 / 7648	23.0	8.84	9.6	6.4	6.12	6.96	1.62

Table 7.1.: *Performance measurements of the simulation. The individual steps are listed in the order in which they appear in the main simulation loop and are measured in milliseconds. The height integration and the velocity integration are sequentially performed three times. The values listed in these columns correspond to the time required for all three iterations together. Note that the time for the visualization is not reflected in the frame rate. Including the rendering time decreases the indicated frame rate by roughly a fourth.*

time integration scheme, the actual simulation performs three sequential explicit Euler steps for every frame. Note that the values listed for the height and velocity integration correspond to the required time for all the three explicit Euler steps. Obviously, the flattening operation is solely a geometrical operation and does not involve the state of the SWE. Hence, the flattening operations only have to be applied once after the simulation domain has changed. The Semi-Lagrangian advection can be done either three times with a third of the actual time step size right before each explicit Euler substep or just once before the three explicit Euler substeps with the full time step size. The former approach is slightly more stable, mainly thanks to the increased numerical damping of the three Semi-Lagrangian advection steps. But the latter approach gains an noticeable speedup w.r.t. the former approach since the velocity advection requires a major fraction of the simulation time. Furthermore, the results look more accurate since waves are not dampened out that fast. Hence, we chose to use the latter approach. Note that the time for the collision handling is not listed in this table because the collision handling depends a lot on the shape and size of the colliding object. The collision handling can decrease the frame rate considerably, but in general this only happens for very complex collision partners.

The operations required for the Semi-Lagrangian advection, i.e., the flattening, the back tracing, the interpolation, and the forward transportation, clearly require most of the simulation time. But keeping the complexity of these operations in mind, the high efficiency of the Semi-Lagrangian advection on a discrete manifold is surprising. A drawback of this advection scheme is that the presented optimizations imply a rather involved implementation, although the most difficult part of the vector interpolation at a vertex boils down to a weighted vector sum (see Algorithm B.2.1 to facilitate an implementation).

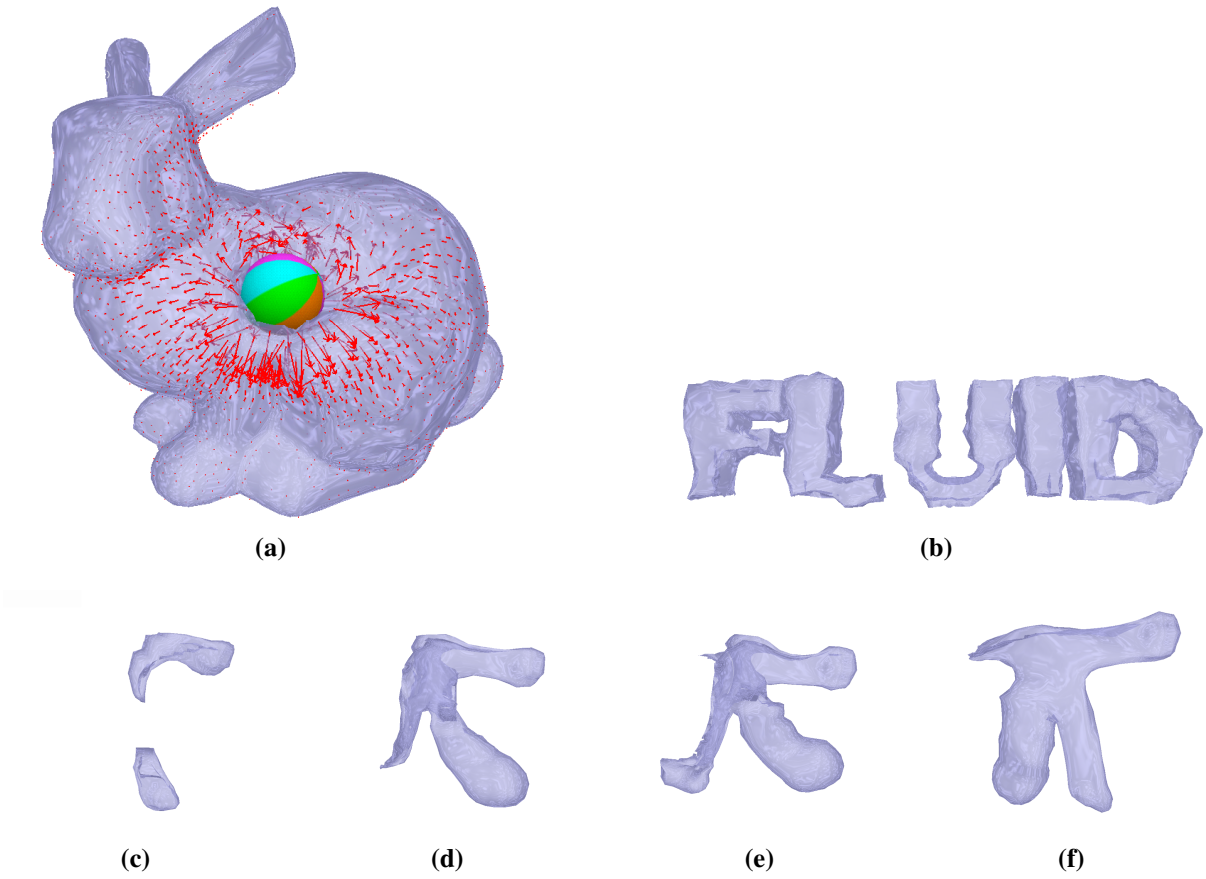
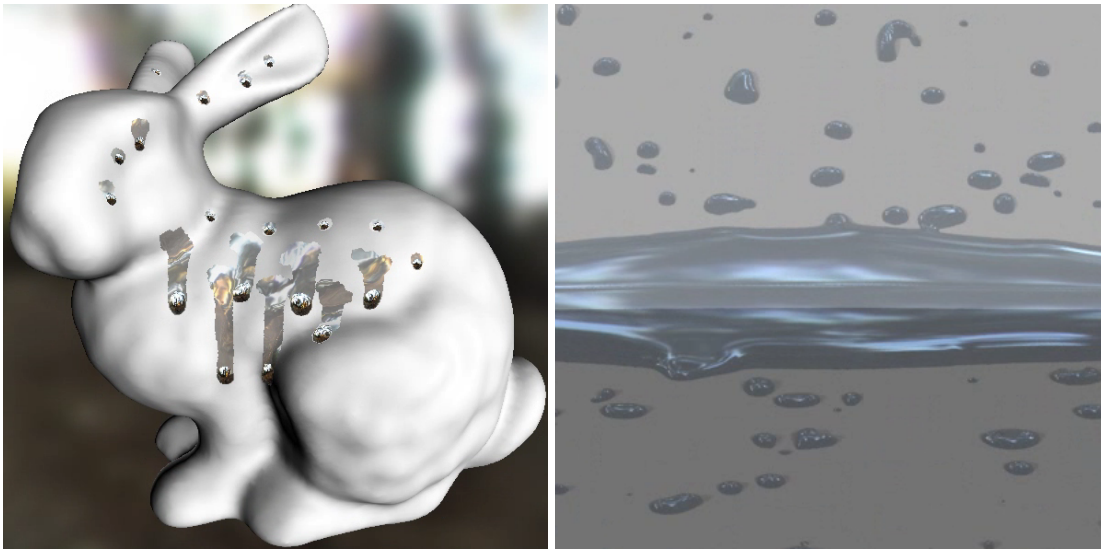


Figure 7.1.: *Some more complex scenarios.*
(a) *and* **(b)** *More complex meshes are handled by our framework as well, as long as the mesh is closed.*
(c) - **(f)** *The filling process also works simultaneously with an animated simulation domain. Two sources at the left leg resp. arm were inserted for this example.*

7.2. Discussion

7.2.1. Comparison

We compare our approach to solve the SWE with the approaches presented in [KM90], [LvdP02] and [WMT07]. [KM90] and [LvdP02] both solve the SWE on a flat, regular grid. To the best of our knowledge, [WMT07] is the first paper dealing with the SWE on curved surfaces. In [KM90], the advectional part of the SWE is ignored whereas [LvdP02] and [WMT07] both use the Semi-Lagrangian advection scheme as well. In [LvdP02] both the fluid depth and the fluid velocity are advected whereas in [WMT07] only the velocity gets advected and a linearized version of the continuity equation $\eta_t = -\eta_0 \nabla \cdot \mathbf{v}$ is used. This latter form of the continuity equation is derived by assuming that the water depth inside the divergence term $\nabla \cdot (\eta \mathbf{v})$ of Equation (3.11) is a local and temporal invariant η_0. All these three approaches derive an implicit integration scheme for the water depth, similar to derivation given in Subsection 5.3.1. Therefore, the SWE must be linearized at some point. Since the continuity equation is nonlinear in both formulations (Equation (5.10) and Equation (5.11)) the conservation of mass is



(a) (b)

Figure 7.2.: *Two frames of the video of [WMT07]. Note that only a small fraction of the vertices needs to be simulated because large parts of the mesh are dry. According to the bulging of the fluid, quite a large surface tension force seems to be necessary to prevent the fluid from spreading all over the simulation domain.*

not ensured in these implicit approaches which can lead to noticeable artifacts during longer simulations. Our explicit integration scheme on the other hand preserves mass perfectly thanks to the discrete divergence presented in Subsection 4.4.2 and adapted to capture the mass flux in Equation (5.12). Furthermore, in contrast to the implicit approaches, our framework can robustly handle the full nonlinear SWE.

The SWE simulation on curved triangle meshes in [WMT07] focuses on the simulation of droplets and, as the authors point out, large parts of the simulation mesh do not need to be simulated since there is no fluid anyway. This allows to use much higher resolution meshes. The authors report a frame rate slightly above 10fps for meshes with about 160'000 vertices. However, they do not mention the fraction of actually simulated vertices. Although the authors present quite impressive results of droplets flowing down a curved mesh (see Figure 7.2), they do not show any results of an irregular and curved mesh which is filled by large parts with fluid. Our approach exactly tries to fill this gap. Furthermore, our approach handles animated simulation domains which is a novel feature. Thanks to our discretization, we can efficiently and robustly handle arbitrary deformations of the simulation domain with the same topology.

7.2.2. Limitations

The interpolation step of the Semi-Lagrangian advection introduces a certain amount of damping which is conducive to the stability but harms the accuracy of the simulation. For example, if the water heights are advected with the Semi-Lagrangian approach as well, the conservation of mass is not ensured due to exactly this interpolation step. As pointed out already earlier, this is the reason why our framework solves the continuity equation without Semi-Lagrangian

advection. However, the velocity is advected with the Semi-Lagrangian approach and hence the conservation of momentum is not ensured. Nevertheless, we observed that the velocity damping is quite low such that the simulation does not give the impression of a viscous fluid.

Obviously the heightfield representation of the fluid imposes several restrictions. For example, breaking waves or splashes can not be handled with such an approach. But note that there exist methods to augment a shallow water simulation with breaking waves [TMFSG07]. Another drawback of the heightfield representation of the fluid surface is that self intersections between the surfaces which belong to the front and back side of the closed two dimensional manifold might occur. This mainly happens for very thin mesh structures. Furthermore, at the current stage of development, only closed triangle meshes are possible as a simulation domain. But an extension of our framework to handle non-closed meshes would be straight forward.

The explicit time integration method imposes a restriction to the admissible time step size and makes the time step dependent on the mesh triangulation. We observed that if the Semi-Lagrangian advection step for the velocity is ignored, the simulation suffers from a quite substantial decrease in stability. Hence, the damping introduced by the Semi-Lagrangian velocity advection seems to be important for a robust simulation. Note that the aforementioned approaches to solve the SWE ([LvdP02], [WMT07]) also make use of the Semi-Lagrangian scheme to take care of the nonlinear advection but due to their implicit nature to solve the remaining parts of the SWE, even more numerical damping is introduced.

7.2.3. Future Work

To alleviate the restrictions imposed by the heightfield representation of the SWE, the SWE simulation could be coupled with other fluid simulation methods such that additional effects could be produced, like for example fluid dropping from the fluid surface whenever the vertical acceleration a_n becomes positive. More specifically, a SPH fluid simulation could be coupled to the SWE simulation. Though, one problem is to find a suitable surface representation to combine a particle based surface representation with a heightfield surface representation.

Throughout the simulation, we avoid to remesh our simulation domain. However, such a remeshing step would allow new discretization alternatives since the mesh triangulation could be guaranteed to fulfill certain quality aspects. For example, circumcenters could be used instead of triangle barycenters and dual edges would be perpendicular to primal edges. Such a high quality triangulation would be beneficial to the stability and accuracy of the simulation method. Moreover, topological changes in the deformation of the simulation domain could be handled with a remeshing step, as well.

The time integration surely deserves further investigation. Implicit integration methods are advantageous with respect to the stability of the simulation. However, increased stability at the expense of excessive numerical damping is not desirable. Hence, we would like to further investigate symplectic integrators. Symplectic integrators conserve the Hamiltonian of a physical system. The interested reader is referred to [MW01] for a detailed introduction to symplectic integrators. Note that the Newmark integration scheme is indeed symplectic and if this scheme is applied to a system of linear dynamic equations of motion, the error of the energy conservation remains bounded. This is a highly desirable property because it allows explicit control over

the amount of damping by explicitly introducing an additional damping term. Normally, the numerical damping introduced by an implicit integration scheme can not be controlled explicitly and is too high, as has been exemplified by the implicit Euler integration scheme. Therefore, it would be very interesting to come up with a energy conserving integration scheme for the SWE. This requires the SWE being formulated in Lagrangian form as has already been done in [Sal83]. In particular, this approach proposes a particle based, variational formulation of the SWE. This particle based simulation is very similar to an SPH method (see [MCG03] and references therein) and handles moving and disconnecting boundaries very easily. Considering these facts, it would be very interesting to simulate the SWE on a curved manifold by simulating a set of fluid particles which move on this manifold. Maybe it would even be possible to derive a variational time integration method for the SWE which works directly on a irregular, curved grid rather than having to resort to a particle based method.

7.2.4. Conclusion

We have presented a framework for the real time and robust physically based simulation of fluid flow on curved surfaces. The fluid is represented as a heightfield and the dynamics are described by a partial differential equation known as the shallow water equations. The curved simulation domain can be discretized by an arbitrary closed triangle mesh. As a unique feature, our simulation framework can handle animated simulation domains. Our general derivation of the shallow water equations with non-constant vertical accelerations allows to couple the two dimensional simulation domain with the embedding three dimensional space. Several interesting effects have been presented to demonstrate this coupling, amongst them collision handling, a filling effect and inertia effects due to the animated simulation domain. The approach proved to be quite versatile, robust and very efficient.

A

Fluid Surface Representation

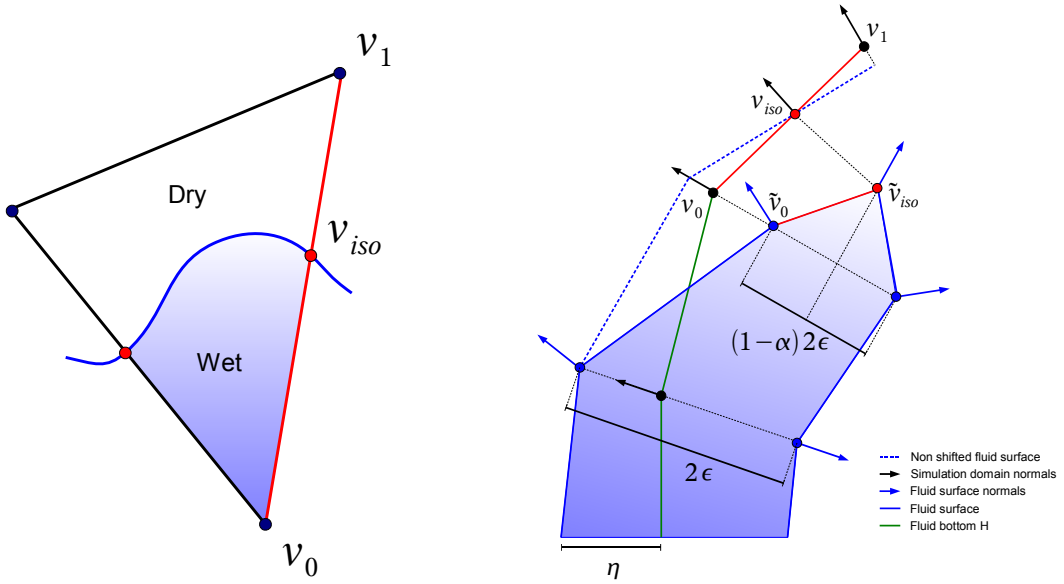
The effects described in Chapter 6 require the representation of empty fluid cells as well. But the SWE do not provide a method to handle the discontinuity at the interface between empty and full cells in a proper way. We explored different approaches to keep track of this interface. For example, a boundary layer of cells between empty and full cells was explicitly carried along. However, the conservation of mass could not be ensured whenever a state transitions from a boundary to a full cell or from a full cell to a boundary cell occurred.

We thus resorted to the following simple approach. Negative fluid heights are allowed but an outward mass flux across the dual cell boundary is clamped to zero such that the negative fluid height can not decrease even more. The mass is thus conserved at the expense of slightly negative fluid depths. The interface between empty and full cells is then defined as an isoline of the fluid height. A marching triangle algorithm is used to extract a piecewise linear representation of this interface. An isolevel η_{iso} equal to zero is not suitable because sudden jumps of the interface appear whenever a formerly empty cell becomes wet. Hence, we chose a small positive value for the isolevel. Note that the larger the value of the isolevel, the smoother is the movement of the interface. However, fluid at very shallow depths can visually disappear whenever the fluid depth falls below the isolevel which might give an impression of fluid oozing away. This artifact is more distinctive for larger isolevels.

To give the fluid a more volumetric look, the interior surface of the fluid object is visualized as well. See also Figure A.1 for a graphical depiction of the following explanations. The surface heightfield is copied and shifted a constant distance 2ϵ along the negative simulation domain normals to represent the interior surface

$$\mathbf{d}_{back}(v) = \mathbf{d}(v) - 2\epsilon\mathbf{n}(v)$$

where the displacement from the simulation domain of a vertex v of the front surface is given by $\mathbf{d}(v) = (\eta - H)\mathbf{n}(v)$ (see also Figure 3.1). The animated fluid object hence seems to consist



(a) *Top view* **(b)** *Side view*

Figure A.1.: *Stitching together the front and the back surface at the isoline.*
(a) *The isoline is defined by an isolevel value η_{iso}. The intersection point v_{iso} is given by the convex combination $\mathbf{x}(v_{iso}) = \alpha\mathbf{x}(v_0) + (1-\alpha)\mathbf{x}(v_1)$.*
(b) *The fluid is given a more volumetric look by copying and shifting the front surface by 2ϵ along the negative simulation domain normals. This figure shows a cross section through the fluid layer defined by the front and back surface. The virtual fluid bottom H is visualized in green lines whereas the edge of the bottom which gets intersected by the isoline is visualized again in red color. The simulation domain normals are shown as black arrows whereas the fluid surface normals appear in blue color. The fluid surface is depicted as a blue line. The dashed blue line corresponds to the non-shifted front fluid surface.*

of a thin layer of fluid. Care has to be taken that both surfaces meet at the isoline, i.e., that the isoline of the front and back surface is congruent. This means that a front surface vertex v_{iso} on an edge $e(v_0, v_1)$ which gets crossed by the isoline is displaced by

$$\mathbf{d}(v_{iso}) = (1-\alpha)(-H(v_1) - \epsilon)\mathbf{n}(v_1) + \alpha(-H(v_0) - \epsilon)\mathbf{n}(v_0)$$

where $\alpha = \frac{\eta_{iso} - \eta_1}{\eta_0 - \eta_1}$ denotes the barycentric coordinate of the intersection point. To avoid discontinuities when the isoline sweeps over a vertex v_0 of the simulation domain which has an adjacent edge $e(v_0, v_1)$ which gets crossed by the isoline, this vertex v_0 is also displaced in normal direction according to the barycentric coordinate α of the intersection point v_{iso}

$$\mathbf{d}(v_0) = ((1-\alpha)(\eta(v_0) - H(v_0)) + \alpha(-H(v_0) - \epsilon))\mathbf{n}(v_0).$$

After displacing all the fluid surface vertices, the fluid surface normals can be computed. Note that attention has to be paid near the seam of the isoline to ensure a continuous transition of the normals from the front to the back surface.

As a future work, we would like to investigate other methods to generate a closed surface than copying and shifting the fluid surface in normal direction. To give the fluid an even more volumetric look, it would be nice to come up with a method to triangulate and fill the holes

which are bounded by the isoline. As an entry point to remeshing and model repair algorithms, we refer to [BPK+07].

B

Implementation Hints

The following sections provide some hints how to implement the core routines of our framework.

B.1. Main Simulation Loop

Procedure `simulate(`Δt`)`

begin
  `UpdateSimulationDomainAnimation` (Δt)
  foreach *Vertex* v *in simulation mesh* **do**
    `FlattenNeighborhood` (v)
  end
  foreach *Face* f *in simulation mesh* **do**
    `AdvectVelocity(`Δt, f`)`
  end
  `CollisionHandling` (Δt)
  `DecomposeExternalForces(`Δt`)`
  for $i \leftarrow 1$ **to** k **do**
    `IntegrateFluidHeights` ($\Delta t/k$)
    `IntegrateVelocities` ($\Delta t/k$)
  end
  `ComputeBarycentricVelocities` (Δt)
end

The main simulation loop clearly reflects the fractional step approach. Updating the simulation

mesh according to the next frame of the simulation domain animation changes the geometry of the simulation domain. Hence, the local neighborhoods must be flattened such that the velocities can be advected. Afterwards, the collision response and other external forces are decomposed into a normal and a tangential component. These external forces are then used in the subsequent k explicit Euler integration steps. Finally, the barycentric representation of the velocity field is computed for the advection in the next time step.

B.2. Flattening and Advection

B.2.1. Flattening

Procedure `FlattenNeighborhood(`*Vertex* v`)`

begin
  foreach *Vertices* $v_i \in \mathcal{N}(v)$ **do**
    $\theta(v)_i \leftarrow$ `CalculateSectorAngle` $(\angle(v_i, v, v_{i+1}))$
    $\alpha(v)_i \leftarrow \alpha_{i-1} + \theta_i$
  end
  $s \leftarrow 2\pi / \sum_{i \in \mathcal{N}(v)} \theta(v)_i$
  foreach *Vertex* $v_i \in \mathcal{N}(v)$ **do**
    $\hat{\theta}(v)_i \leftarrow s\theta(v)_i$
    $\hat{\alpha}(v)_i \leftarrow s\alpha(v)_i$
    $\tilde{\mathbf{u}}(v)_i \leftarrow \|$ `Edge` $(v, v_i)\, \|(\cos \hat{\alpha}(v)_i, \sin \hat{\alpha}(v)_i)^T$
  end
end

The flattening operation at a vertex v computes the representation of the adjacent edges in the local parameterization plane of the vertex v. See Section 5.2.2 for the notation, formulas and for graphical depictions. Note that the sector angles can be reused for the cotangent weights of the differential operators.

B.2.2. Advection

Procedure `AdvectVelocity(`Δt, *Face* f`)`

begin
  $(f_{end}, \mathbf{x}_{end}, rot) \leftarrow$ `BackTrace` $(\Delta t, f)$
  $\mathbf{v} \leftarrow$ `Interpolate` $(f_{end}, \mathbf{x}_{end})$
  $\mathbf{v} \leftarrow rot(\mathbf{v})$
end

The advection of the velocity at a face f starts by tracing the face barycenter backwards in time. The backtracing returns the end point of the tracing $\mathbf{x}_{end}$ in barycentric coordinates w.r.t. the end triangle f_{end} and the quaternion rot which stores the necessary rotation to transport the interpolated velocity $\mathbf{v}$ to the starting triangle f.

C

Software User Interface

The executable of the framework is contained in the folder `Code/bin/win32` and is called `FluidMonster.exe`. Before running the program, the PhysX framework[1] must be set up according to Figure C.1 by executing `PhysXSettings.exe` which is also contained in the folder `Code/bin/win32`. The executable `FluidMonster.exe` requires one parameter, namely the folder containing the Cal3D[2] animation files. Note that this folder is given relative to the directory `Code/BriX/Media`. Several sample meshes and animations are provided in `Code/BriX/Media/Models`. For example, run `FluidMonster.exe Models/sphereLarge` to load the sphere data set.

Once the program is running, the keys listed in Table C.1 are available. The animation of the simulation domain is controlled with several parameters as visualized in Figure C.2 (b). The visible-parameter controls the alpha value of the simulation domain. The freeze-option freezes the skeleton-based animation immediately. Note that the fluid simulation continues while the animation of the simulation domain is frozen. Parameters and options controlling the SWE simulation are listed in Table C.2 and are visualized in Figure C.2 (a).

[1] www.ageia.com/physx
[2] home.gna.org/cal3d/

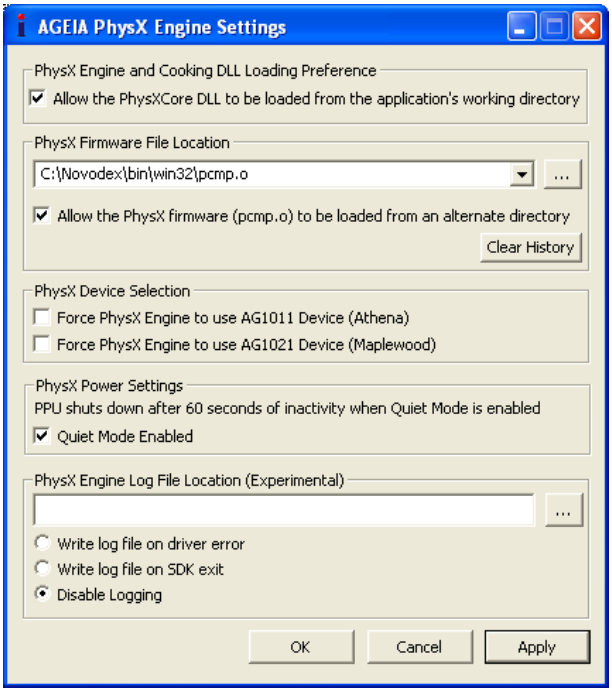
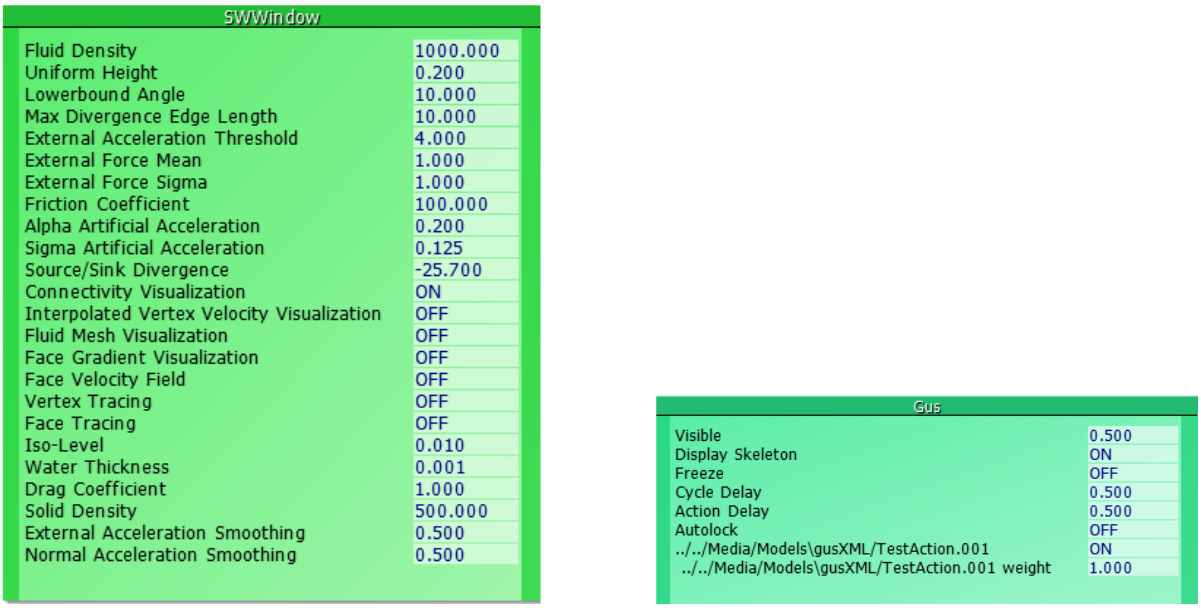


Figure C.1.: *The framework makes use of the PhysX SDK which must be set up according to the settings given above.*



(a) **(b)**

Figure C.2.: *The user interface of the software which was developed during this thesis. The display of the user interface can be switched on and off by pressing function key 3 `F3`.*

Key	Description
w, a, s, d	Fly around with the well-known wasd-control
space	Throw a weapon
m	Insert source or sink at the current cursor position
p	Pause the simulation
o	Perform a single time step of the simulation
f3	Toggle the display of parameter and options windows
f5	Toggle the display of shadows
esc	Quits the program
Left mouse button	Look around
Mouse wheel	Select a weapon to throw at the fluid
Middle mouse button	Exert external force at the current cursor position. The magnitude of the force is modeled by a Gaussian falloff function.
Right mouse button	Drag around the simulation domain by pressing and holding the right mouse button.

Table C.1.: *Keyboard and mouse assignments.*

Parameter	Description	Formula
Fluid Density	Density of the fluid	ρ
Uniform Height	Sets a uniform bottom height H	H
Lowerbound Angle	Threshold in degrees to limit the minimal angle in badly shaped triangles	
Max Divergence Edge Length	Threshold to limit the maximal edge length of a dual cell in the divergence computation	
External Acceleration Threshold	Threshold to limit the maximal external acceleration	
External Force Mean	Maximal value of the force which is exerted when the middle mouse button is pressed	
ExternalForce Sigma	Controls the steepness of the Gaussian falloff function of the force which is exerted when the middle mouse button is pressed	
Friction Coefficient	Coefficient of a friction force which can be used to artificially dampen the velocity field. Set to 0 if the velocity field should vanish.	
Alpha Artificial Acceleration	Equilibrium depth of artificial acceleration	Equation (6.2)
Sigma Artificial Acceleration	Steepness parameter of artificial acceleration	Equation (6.2)
Source/Sink Divergence	Sets the divergence of an artificial source or sink	
Connectivity Visualization	Toggles the visualization of the triangle mesh of the simulation domain.	
Interpolated Vertex Velocity Visualization	Toggles the visualization of the interpolated velocity vectors at the vertices. The interpolated velocity at the vertex is mapped back to each adjacent triangle using the inverse angle ratio preserving mapping and is then visualized.	
Fluid Mesh Visualization	Toggles the visualization of the fluid surface mesh and of the fluid bottom mesh.	
Face Gradient Visualization	Toggles the visualization of the tangential acceleration acting on the tangential velocity field.	Equation (5.13)
Face Velocity Field	Toggles the visualization of the face velocity field.	
Vertex Tracing	Toggles the visualization of the back tracing paths of the Semi-Lagrangian advection at the vertices.	
Face Tracing	Toggles the visualization of the back tracing paths of the Semi-Lagrangian advection at the faces.	
Iso-Level	Iso-level of the fluid depth for the marching triangle algorithm (see Appendix A).	η_{iso}
Fluid Thickness	Thickness of the fluid layer defined by the front and back surface (see Appendix A).	ϵ
Drag Coefficient	Drag coefficient of the drag force.	Equation (6.1)
Solid Density	Density of the solid which can be thrown at the fluid object.	
External Acceleration Smoothing	Exponential decay factor of external acceleration smoothing according to Equation (6.4).	Equation (6.4)
Normal Acceleration Smoothing	Exponential decay factor of accelerations in normal direction, analog to external acceleration smoothing in Equation (6.4). Set this parameter to a large value to make the fluid more inert.	

Table C.2.: *Listing of the available options in the user interface.*

Bibliography

[AB05] Emmanuel Audusse and Marie-Odile Bristeau. A well-balanced positivity preserving "second-order" scheme for shallow water flows on unstructured meshes. *J. Comput. Phys.*, 206(1):311–333, 2005.

[AC97] K. Anastasiou and C. T. Chan. Solution of the 2D shallow water equations using the finite volume method on unstructured triangular meshes. *International Journal for Numerical Methods in Fluids*, 24:1225–1245, June 1997.

[AL77] A. Arakawa and V. R. Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. In *Methods in Computational Physics, volume 17*, pages 174–267. Academic Press, 1977.

[BMF07] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007 course notes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 1–81, New York, NY, USA, 2007. ACM Press.

[BPK+07] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Levy, Stephan Bischoff, and Christian Rossl. Geometric modeling based on polygonal meshes. In *SIGGRAPH Course Notes*, 2007.

[Car45] Elie Cartan. *Les systèmes différentiels exterieurs et leurs applications geométriques.* Paris: Hermann; Cie. 214 p. , 1945.

[Cat74] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, 1974.

[Cho69] Alexandre Joel Chorin. On the convergence of discrete approximations to the navier-stokes equations. *Mathematics of Computation*, 23(106):341–353, apr 1969.

[CLT07] Keenan Crane, Ignacio Llamas, and Sarah Tariq. Real-time simulation and rendering of 3d fluids. In *GPU Gems 3*. Addison-Wesley Professional, 2007.

[Cra] M. S. Cramer. Navier-stokes equations. Web page. http://www.eng.vt.edu/fluids/msc/ns/nsintro.htm.

[DFM07] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[DKT06] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 39–54, New York, NY, USA, 2006. ACM Press.

[EKH02] K. S. Erduran, V. Kutija, and C. J. M. Hewett. Performance of finite volume solutions to the shallow water equations with shock-capturing schemes. *International Journal for Numerical Methods in Fluids*, 40:1237–1273, December 2002.

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[ETK+07] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1):4, 2007.

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[FL04] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 441–448, New York, NY, USA, 2004. ACM Press.

[FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing: GMIP*, 58(5):471–483, 1996.

[FM97] Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[FOK05] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 904–909, New York, NY, USA, 2005. ACM Press.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 2001. ACM Press.

[GDP+06] Eitan Grinspun, Mathieu Desbrun, Konrad Polthier, Peter Schröder, and Ari Stern. Discrete differential geometry: an applied introduction. In *SIGGRAPH '06: ACM SIGGRAPH 2006 courses*, New York, NY, USA, 2006. ACM Press.

[HKSP07] D. A. Ham, S. C. Kramer, G. S. Stelling, and J. Pietrzak. The symmetry and stability of unstructured mesh C-grid shallow water models under the influence of Coriolis. *Ocean Modelling*, 16:47–60, 2007.

[KFCO06] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 820–825, New York, NY, USA, 2006. ACM Press.

[KLLR07] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):135–144, 2007.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM Press.

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 457–462, New York, NY, USA, 2004. ACM Press.

[LMW02] H. P. Langtangen, K.-A. Mardal, and R. Winther. Numerical methods for incompressible viscous flow. *Advances in Water Resources*, 25:1125–1146, 2002.

[LvdP02] Anita T. Layton and Michiel van de Panne. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*, 18(1):41–53, 2002.

[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.

[MDSB02] M. Meyer, M. Desbrun, P. Schroder, and A. Barr. Discrete differential geometry operators for triangulated 2-manifolds, 2002.

[Mun93] J.R. Munkres. Simplicial complexes and simplicial maps, 1993.

[MW01] J. Marsden and M. West. Discrete mechanics and variational integrators, 2001.

[Nem59] N. M. Nemark. A method of computation for structural dynamics. *ASCE Journal of the Engineering Mechanics Division*, 85(EM3):67–94, 1959.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PP02] K. Polthier and E. Preuss. Identifying vector fields singularities using a discrete hodge decomposition, 2002.

[Sal83] R. Salmon. Practical use of Hamilton's principle. *Journal of Fluid Mechanics*, 132:431–444, 1983.

[SFK+07] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An unconditionally stable maccormack method. J. Sci. Comput. (in review), 2007.

[Sta99] Jos Stam. Stable fluids. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles, 1999. Addison Wesley Longman.

[Sta03] Jos Stam. Flows on surfaces of arbitrary topology. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 724–731, New York, NY, USA, 2003. ACM Press.

[SY04] Lin Shi and Yizhou Yu. Inviscid and incompressible fluid simulation on triangle meshes: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):173–181, 2004.

[SY05a] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Trans. Graph.*, 24(1):140–164, 2005.

[SY05b] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–236, New York, NY, USA, 2005. ACM Press.

[TKPR06] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 7–12, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[TLHD03] Y. Tong, S. Lombeyda, A. Hirani, and M. Desbrun. Discrete multiscale vector field decomposition, 2003.

[TMFSG07] Nils Thuerey, Matthias Mueller-Fischer, Simon Schirm, and Markus Gross. Real-time breaking waves for shallow water simulations. In *Pacfific Graphics 2007*, 2007.

[TMPS03] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 716–723, New York, NY, USA, 2003. ACM Press.

[WBOL07] Jeremy D. Wendt, William Baxter, Ipek Oguz, and Ming C. Lin. Finite volume flow simulations on arbitrary domains. *Graph. Models*, 69(1):19–32, 2007.

[WMT07] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–238, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.