A Navier-Stokes Solver for Single- and Two-Phase Flow

by

Kim Motoyoshi Kalland

THESIS

for the degree of

MASTER OF SCIENCE

(Master i Anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

September 2008

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this master's thesis, I have implemented a 2D Navier-Stokes solver, documented in detail the numerical methods used, explained how the solver works and how it can be used to solve flow problems. The Navier-Stokes equations have been solved numerically since the 1960s, and consequently there exists lots of codes. Most of these are commercial, the best known being Fluent, STAR-CD and ANSYS CFX, which have proved to tackle very complicated fluid dynamics problems. One problem with these commercial codes is the difficulty in modifying the source code and make new software interacting with the codes. The user interface is also geared towards comprehensive GUIs.

There are surprisingly few open source codes for solving the incompressible Navier-Stokes equations. OpenFOAM is perhaps the best known open source code in this category. The code is huge and requires the user to program in C++ to solve a new problem. LifeV is a similar package. FEATFLOW is another free alternative, written in Fortran 77. The mentioned open source codes have a quite steep learning curve and aim at professional activity in computational fluid dynamics (CFD). For educational purposes, and for numerical methods research, it is handy to have a quite simple code, written in a high-level ("Matlab-like") language, supplied with a documentation of all numerical methods and "tricks" in the code (this latter point is missing in the literature and in most of the available codes I have seen). With such a simple code, one can learn the solution technology and quite easily extend the program to more advanced problems. This experience may be very valuable before approaching professional CFD codes.

There is in fact already available an open source Navier-Stokes solver NaSt2D [9] of the type I have implemented. Actually, a fair amount of what I have implemented is based on the book [9]. However, I would say that the solver I present here is more generic, more user friendly, more up-to-date by using C++ and Python instead of C. On the other hand, it also lacks some features included in NaSt2D. My solver is set up and run from

a Python script, which gives much more control and "power", but requires some programming skills from the user. Hopefully, this text will show how simple the programming can be to solve new problems with this solver.

In section 5.5.3, I introduce a new method to reduce spurious currents caused by inaccurate surface tension. The method is built on the direction averaged curvature method described in [17, 16].

The solver is restricted to rectangular domains, and the discretisation is based on finite differences. Both single- and two-phase laminar, incompressible flow can be simulated.

1.1 Symbols

In general, I will explain the symbols in the text where they are used so that the reader does not have to turn pages back and forth, but there are some symbols used throughout the text that I will explain here.

- $\boldsymbol{x}$: space variable in 2D or 3D.
- x, y, z: the components of the space variable mentioned above.
- t: time variable.
- p or $p(\boldsymbol{x}, t)$: pressure in the fluid as function of space and time.
- $\boldsymbol{u}$ or $\boldsymbol{u}(\boldsymbol{x}, t)$: fluid velocity as function of space and time.
- u, v, w or $u(\boldsymbol{x}, t), v(\boldsymbol{x}, t), w(\boldsymbol{x}, t)$: the components of the velocity function mentioned above.
- μ or $\mu(\boldsymbol{x}, t)$: viscosity as function of space and time.
- ρ or $\rho(\boldsymbol{x}, t)$: density as function of space and time.
- σ: surface tension coefficient.

Boldface, italicised symbols represent matrices if written in upper-case and vectors or vector functions if written in lower-case. Scalar values and functions are represented by italicised symbols.

1.2 Operators

In this text, I write equations both in 2D component form and in vector form. When the same equations are expressed both in component and vector form, I put boxes around each of the forms to group the equations. If the reader doesn't have much experience with the ∇ (del) operator, I believe it is easier to follow the equations in component form. However, these equations will

Property	Symbol	Unit of measurement	
Velocity	$\boldsymbol{u}$	m/s	metres per second
Pressure	p	Pa=N/m^2	pascal
Density	ρ	kg/m^3	kilograms per cubic metre
Dynamic viscosity	μ	Pa·s	pascal seconds
Kinematic viscosity	ν	m^2/s	sq. metres per sec.
Surface tension coeff.	σ	N/m	newton per metre
Gravitational acc.	$\boldsymbol{g}$	m/s^2	metres per sec. squared
Time	t	s	seconds

Table 1.1: Units of measurement used for fluid properties.

only be valid in 2D. The operator is defined as $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]^T$ in 2D and $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}]^T$ in 3D. The equations in vector form are valid in 1D, 2D and 3D, they are more compact and generally easier to understand and remember.

Chapter 2

Navier-Stokes equations

Navier-Stokes equations are a set of partial differential equations that describe the motion of fluids as a relationship between flow velocity (or momentum) and pressure. The fluid can be a gas or a liquid.

The general Navier-Stokes equations can be written as:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla p + \nabla\cdot\boldsymbol{T} + \boldsymbol{f} \tag{2.1}$$

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}) = 0 \tag{2.2}$$

where

- ρ = density
- $\boldsymbol{u}$ = velocity
- p = pressure
- $\boldsymbol{T}$ = stress tensor
- $\boldsymbol{f}$ = sum of external forces

The momentum equation (2.1) is equivalent to Newton's second law of motion. The left hand side of the equation is the product of density and acceleration, the right hand side is the sum of forces per volume acting on a fluid particle, an infinitesimal fluid volume.

The continuity equation (2.2) ensures conservation of mass; mass cannot appear out of or disappear into thin air.

All the variables above are functions of time t and space (x, y, z). To avoid too many symbols, I omit writing the time variable when the time is fixed, and I omit the space variables when the position is fixed. The velocity vector $\boldsymbol{u}$ is sometimes split into its components u, v, w, and the stress tensor matrix $\boldsymbol{T}$ into its rows $\boldsymbol{T}_x, \boldsymbol{T}_y, \boldsymbol{T}_z$ or elements $T_{i,j}, 1 \leq i, j \leq 3$. The 2D case follows analogously.

In the following sections, each of the terms in the equations is explained.

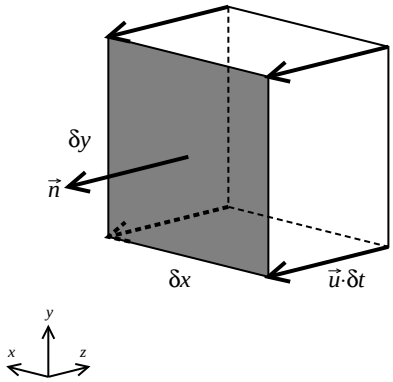


Figure 2.1: Volume passing through a surface during a time step δt.

2.1 Conservation of mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) \tag{2.3}$$

The amount of mass flowing through a surface during a time interval is given by:

$$\int_{t_0}^{t_1} \int_A \rho \ \boldsymbol{u} \cdot \boldsymbol{n} \ dA \ dt \tag{2.4}$$

where A is the surface, $\boldsymbol{n}$ is the surface normal, t_0 is the initial time and t_1 is the final time.

The above integral can be approximated with the product:

$$\delta t \ |A| \ \rho \ \boldsymbol{u} \cdot \boldsymbol{n} \tag{2.5}$$

where $\delta t = (t_1 - t_0)$ and $|A|$ is the surface area, and ρ and $\boldsymbol{u}$ are the density and velocity in the middle of the surface (see figure 2.1).

We look at a small cuboid volume with size $\delta x \times \delta y \times \delta z$. The net amount of mass flowing out of the volume must equal the mass loss inside the volume. To find the net amount of mass flowing out of the volume, we must sum together the mass flowing through each of the six faces (see figure 2.2).

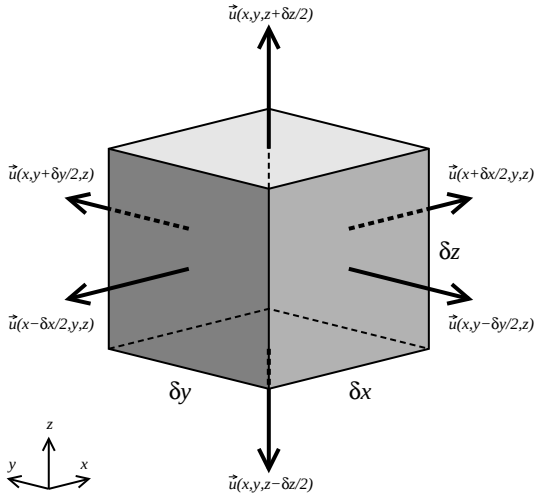


Figure 2.2: Flow through the faces of a cube.

$$\begin{aligned}
\delta m &= \delta t\delta y\delta z(\boldsymbol{e}_x \cdot (\boldsymbol{u}\rho)(x+\delta x/2,y,z) + (-\boldsymbol{e}_x)\cdot(\boldsymbol{u}\rho)(x-\delta x/2,y,z)) \\
&+ \delta t\delta z\delta x(\boldsymbol{e}_y \cdot (\boldsymbol{u}\rho)(x,y+\delta y/2,z) + (-\boldsymbol{e}_y)\cdot(\boldsymbol{u}\rho)(x,y-\delta y/2,z)) \\
&+ \delta t\delta x\delta y(\boldsymbol{e}_z \cdot (\boldsymbol{u}\rho)(x,y,z+\delta z/2) + (-\boldsymbol{e}_z)\cdot(\boldsymbol{u}\rho)(x,y,z-\delta z/2)) \\
&= \delta t\delta y\delta z((u\rho)(x+\delta x/2,y,z) - (u\rho)(x-\delta x/2,y,z)) \\
&+ \delta t\delta x\delta z((v\rho)(x,y+\delta y/2,z) - (v\rho)(x,y-\delta y/2,z)) \\
&+ \delta t\delta x\delta y((w\rho)(x,y,z+\delta z/2) - (w\rho)(x,y,z-\delta z/2))
\end{aligned}$$

where δm is the mass loss and $(\boldsymbol{u}\rho)(x,y,z)$ is the same as $\boldsymbol{u}(x,y,z)\rho(x,y,z)$.

The mass loss inside the volume can be expressed as:

$$-\int_\Omega (\rho(x,y,z,t_1) - \rho(x,y,z,t_0))d\Omega \tag{2.6}$$

where Ω is the volume, t_0 is the initial time and t_1 is the final time.

This can be approximated by

$$\delta m = -|\Omega|(\rho(t_1) - \rho(t_0)) = -\delta x\delta y\delta z(\rho(t_1) - \rho(t_0)) \tag{2.7}$$

where $|\Omega| = \delta x\delta y\delta z$ is the volume size and ρ is the density in the middle of the volume.

By equating the net mass outflow and the mass loss we get the following:

$$\begin{aligned}
-\delta x\delta y\delta z(\rho(t_1)-\rho(t_0)) &= \delta t\delta y\delta z((u\rho)(x+\delta x/2,y,z)-(u\rho)(x-\delta x/2,y,z))\\
&+\delta t\delta x\delta z((v\rho)(x,y+\delta y/2,z)-(v\rho)(x,y-\delta y/2,z))\\
&+\delta t\delta x\delta y((w\rho)(x,y,z+\delta z/2)-(w\rho)(x,y,z-\delta z/2))\\
-\frac{\rho(t_1)-\rho(t_0)}{\delta t} &= \frac{(u\rho)(x+\delta x/2,y,z)-(u\rho)(x-\delta x/2,y,z)}{\delta x}\\
&+\frac{(v\rho)(x,y+\delta y/2,z)-(v\rho)(x,y-\delta y/2,z)}{\delta y}\\
&+\frac{(w\rho)(x,y,z+\delta z/2)-(w\rho)(x,y,z-\delta z/2)}{\delta z}
\end{aligned}$$

When δt, δx, δy and δz approach zero, we get:

$$-\frac{\partial\rho}{\partial t}=\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}+\frac{\partial(\rho w)}{\partial z}=\nabla\cdot(\rho\boldsymbol{u}) \tag{2.8}$$

By reordering the terms, we get equation (2.2).

If the fluid is modelled as incompressible, the volume flowing into a cuboid must be equal to the volume flowing out. The net volume outflow, which must be zero, can be approximated as:

$$\begin{aligned}
0 &= \delta y\delta z(u(x+\delta x/2,y,z)-u(x-\delta x/2,y,z))\\
&+\delta x\delta z(v(x,y+\delta y/2,z)-v(x,y-\delta y/2,z))\\
&+\delta x\delta y(w(x,y,z+\delta z/2)-w(x,y,z-\delta z/2))\\
&=\frac{u(x+\delta x/2,y,z)-u(x-\delta x/2,y,z)}{\delta x}\\
&+\frac{v(x,y+\delta y/2,z)-v(x,y-\delta y/2,z)}{\delta y}\\
&+\frac{w(x,y,z+\delta z/2)-w(x,y,z-\delta z/2)}{\delta z}
\end{aligned}$$

When δx, δy and δz approach zero, we get:

$$0=\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=\nabla\cdot\boldsymbol{u} \tag{2.9}$$

This equation is used as continuity equation for incompressible flow.

2.2 Acceleration

$$\rho\left(\frac{\partial\boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) \tag{2.10}$$

To see that the left hand side of equation (2.1) is the product of density and acceleration, begin by looking at a particle. Assume that the particle at at any given time has the position $\boldsymbol{q}(t)=[q_x(t),q_y(t),q_z(t)]^T$. Let

$\boldsymbol{u}(x, y, z, t)$ be the velocity field, that is, the function $\boldsymbol{u}$ will return the velocity of a fluid particle at any given time and position. Since the particle velocity $\boldsymbol{v}(t)$ is the time derivative of the position $\boldsymbol{q}(t)$, we must have

$$\boldsymbol{v}(t) = \boldsymbol{q}'(t) = [q_x'(t), q_y'(t), q_z'(t)]^T \tag{2.11}$$

By the definition of the velocity field, we also have

$$\boldsymbol{v}(t) = \boldsymbol{u}(q_x(t), q_y(t), q_z(t), t) \tag{2.12}$$

The particle acceleration $\boldsymbol{a}(t)$ is the time derivative of the velocity $\boldsymbol{v}(t)$:

$$\boldsymbol{a}(t) = \boldsymbol{v}'(t) = \frac{d}{dt}\boldsymbol{u}(q_x(t), q_y(t), q_z(t), t) \tag{2.13}$$

Differentiate the velocity field function $\boldsymbol{u}$ with respect to time t:

$$\begin{aligned}
\boldsymbol{v}'(t) &= \frac{d}{dt}\boldsymbol{u}(q_x(t), q_y(t), q_z(t), t) \\
&= \lim_{s\to t} \frac{1}{s-t} \left(\boldsymbol{u}(q_x(s), q_y(s), q_z(s), s) - \boldsymbol{u}(q_x(t), q_y(t), q_z(t), t)\right) \\
&= \lim_{s\to t} \Bigg[\frac{1}{s-t} \left(\boldsymbol{u}(q_x(s), q_y(s), q_z(s), s) - \boldsymbol{u}(q_x(s), q_y(s), q_z(s), t)\right) \\
&+ \frac{1}{s-t} \left(\boldsymbol{u}(q_x(s), q_y(s), q_z(s), t) - \boldsymbol{u}(q_x(s), q_y(s), q_z(t), t)\right) \\
&+ \frac{1}{s-t} \left(\boldsymbol{u}(q_x(s), q_y(s), q_z(t), t) - \boldsymbol{u}(q_x(s), q_y(t), q_z(t), t)\right) \\
&+ \frac{1}{s-t} \left(\boldsymbol{u}(q_x(s), q_y(t), q_z(t), t) - \boldsymbol{u}(q_x(t), q_y(t), q_z(t), t)\right)\Bigg] \\
&= \lim_{s\to t} \Bigg[\frac{\boldsymbol{u}(q_x(s), q_y(s), q_z(s), s) - \boldsymbol{u}(q_x(s), q_y(s), q_z(s), t)}{s-t} \\
&+ \frac{q_z(s) - q_z(t)}{s-t} \cdot \frac{\boldsymbol{u}(q_x(s), q_y(s), q_z(s), t) - \boldsymbol{u}(q_x(s), q_y(s), q_z(t), t)}{q_z(s) - q_z(t)} \\
&+ \frac{q_y(s) - q_y(t)}{s-t} \cdot \frac{\boldsymbol{u}(q_x(s), q_y(s), q_z(t), t) - \boldsymbol{u}(q_x(s), q_y(t), q_z(t), t)}{q_y(s) - q_y(t)} \\
&+ \frac{q_x(s) - q_x(t)}{s-t} \cdot \frac{\boldsymbol{u}(q_x(s), q_y(t), q_z(t), t) - \boldsymbol{u}(q_x(t), q_y(t), q_z(t), t)}{q_x(s) - q_x(t)}\Bigg] \\
&= \frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial z}\frac{\partial q_z}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial y}\frac{\partial q_y}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial x}\frac{\partial q_x}{\partial t}
\end{aligned}$$

The result can also be obtained by using the chain rule for several variables. Since $\frac{\partial \boldsymbol{q}}{\partial t}$ is the velocity, we can replace it with $\boldsymbol{u}$.

$$\begin{aligned}
\boldsymbol{v}'(t) &= \frac{d}{dt}\boldsymbol{u}(q_x(t), q_y(t), q_z(t), t) \\
&= \frac{\partial \boldsymbol{u}}{\partial x}\frac{\partial q_x}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial y}\frac{\partial q_y}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial z}\frac{\partial q_z}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial t} \quad = \frac{\partial \boldsymbol{u}}{\partial x}u + \frac{\partial \boldsymbol{u}}{\partial y}v + \frac{\partial \boldsymbol{u}}{\partial z}w + \frac{\partial \boldsymbol{u}}{\partial t} \\
&= \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{\partial \boldsymbol{u}}{\partial t}
\end{aligned}$$

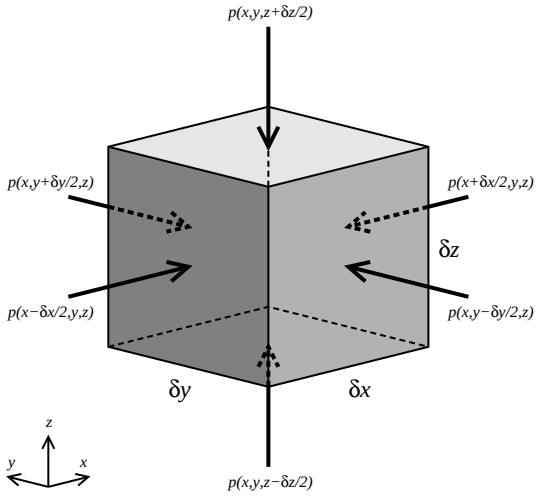


Figure 2.3: Control volume with face centred pressure values.

This shows how the left hand side of the momentum equation (2.1) is derived.

The 2D case is similar. By multiplying the acceleration with the density ρ, we get the same expression as the left hand side of equation (2.1).

2.3 Pressure

$$-\nabla p \tag{2.14}$$

Let $p(t, x, y, z)$ be the pressure field. To find the pressure force acting on a particle, we construct a small, cuboid volume with size $\delta x \times \delta y \times \delta z$. The volume and the direction of the pressure forces are shown in figure 2.3.

Pressure is an amount of force per area, so to find the force, we must integrate the pressure over the area of each face. The integral is approximately the product of the area and the face centred pressure.

By summing the contribution from all six faces, we get the net pressure force. Since we are interested in the force per volume, we divide by $\delta x \delta y \delta z$.

Let $\boldsymbol{f}_p$ be the pressure force per volume.

$$\begin{aligned}\boldsymbol{f}_p(x,y,z) &= \frac{\delta y\delta z}{\delta x\delta y\delta z}(-\boldsymbol{e}_x)p(x+\delta x/2,y,z) + \frac{\delta y\delta z}{\delta x\delta y\delta z}\boldsymbol{e}_x p(x-\delta x/2,y,z)\\ &+ \frac{\delta z\delta x}{\delta x\delta y\delta z}(-\boldsymbol{e}_y)p(x,y+\delta y/2,z) + \frac{\delta z\delta x}{\delta x\delta y\delta z}\boldsymbol{e}_y p(x,y-\delta y/2,z)\\ &+ \frac{\delta x\delta y}{\delta x\delta y\delta z}(-\boldsymbol{e}_z)p(x,y,z+\delta z/2) + \frac{\delta x\delta y}{\delta x\delta y\delta z}\boldsymbol{e}_z p(x,y,z-\delta z/2)\\ &= -\begin{bmatrix} \frac{1}{\delta x}(p(x+\delta x/2,y,z)-p(x-\delta x/2,y,z)) \\ \frac{1}{\delta y}(p(x,y+\delta y/2,z)-p(x,y-\delta y/2,z)) \\ \frac{1}{\delta z}(p(x,y,z+\delta z/2)-p(x,y,z-\delta z/2)) \end{bmatrix}\end{aligned}$$

where $\boldsymbol{e}_x$ etc. are the unit vectors along each axis. As δx, δy and δz approach zero, we get:

$$\boldsymbol{f}_p(x,y,z) = -\begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \\ \frac{\partial p}{\partial z} \end{bmatrix} = -\nabla p \tag{2.15}$$

which is equal to the pressure term in the momentum equation (2.1).

2.4 Viscosity

$$\nabla \cdot \boldsymbol{T} \tag{2.16}$$

Viscosity is the friction within the fluid and depends on velocity differences, or the deformation rate. The viscous stress acting on a surface depends on the surface orientation and can be expressed as a product of the stress tensor matrix $\boldsymbol{T}$ and the surface normal vector $\boldsymbol{n}$. In figure 2.4, the viscous stress is shown for each face.

Stress is, like pressure, an amount of force per area. The net viscous force exerted on the volume is found the same way as for pressure. Since we are interested in the force per volume, we divide by $\delta x\delta y\delta z$. Let $\boldsymbol{f}_v$ be the viscous force per volume.

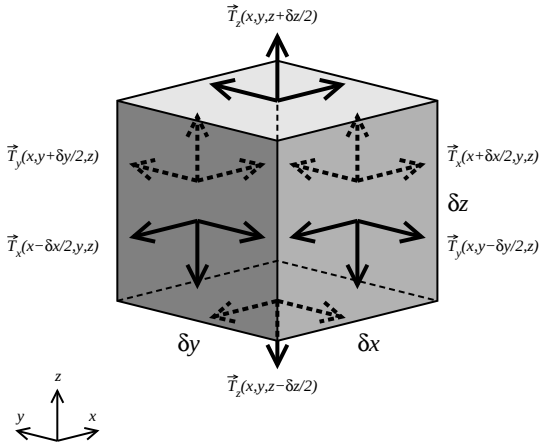


Figure 2.4: Control volume with face centred stress vectors decomposed into x, y and z-components.

$$
\begin{aligned}
\boldsymbol{f}_v(x,y,z) &= \frac{\delta y \delta z}{\delta x \delta y \delta z}\boldsymbol{T}_x(x+\delta x/2,y,z) - \frac{\delta y \delta z}{\delta x \delta y \delta z}\boldsymbol{T}_x(x-\delta x/2,y,z) \\
&+ \frac{\delta z \delta x}{\delta x \delta y \delta z}\boldsymbol{T}_y(x,y+\delta y/2,z) - \frac{\delta z \delta x}{\delta x \delta y \delta z}\boldsymbol{T}_y(x,y-\delta y/2,z) \\
&+ \frac{\delta x \delta y}{\delta x \delta y \delta z}\boldsymbol{T}_z(x,y,z+\delta z/2) - \frac{\delta x \delta y}{\delta x \delta y \delta z}\boldsymbol{T}_z(x,y,z-\delta z/2) \\
&= \frac{\boldsymbol{T}_x(x+\delta x/2,y,z) - \boldsymbol{T}_x(x-\delta x/2,y,z)}{\delta x} \\
&+ \frac{\boldsymbol{T}_y(x,y+\delta y/2,z) - \boldsymbol{T}_y(x,y-\delta y/2,z)}{\delta y} \\
&+ \frac{\boldsymbol{T}_z(x,y,z+\delta z/2) - \boldsymbol{T}_z(x,y,z-\delta z/2)}{\delta z}
\end{aligned}
$$

where $\boldsymbol{T}_x$ is the transpose of the first row of $\boldsymbol{T}$ etc.

As δx, δy and δz approach zero, we get:

$$
\boldsymbol{f}_v(x,y,z) = \left(\frac{\partial \boldsymbol{T}_x}{\partial x} + \frac{\partial \boldsymbol{T}_y}{\partial y} + \frac{\partial \boldsymbol{T}_z}{\partial z}\right) = \nabla \cdot \boldsymbol{T} \tag{2.17}
$$

which is equal to the viscosity term in the momentum equation (2.1).

Many fluids can be modelled as Newtonian fluids. Newtonian fluids are defined as fluids where the viscous stress is proportional to the deformation

rate[33]:

$$\begin{aligned}
\boldsymbol{T}_{1,1} &= 2\mu\frac{\partial u}{\partial x} + \lambda\nabla\cdot\boldsymbol{u} \\
\boldsymbol{T}_{2,2} &= 2\mu\frac{\partial v}{\partial y} + \lambda\nabla\cdot\boldsymbol{u} \\
\boldsymbol{T}_{3,3} &= 2\mu\frac{\partial w}{\partial z} + \lambda\nabla\cdot\boldsymbol{u} \\
\boldsymbol{T}_{1,2} = \boldsymbol{T}_{2,1} &= \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\
\boldsymbol{T}_{2,3} = \boldsymbol{T}_{3,2} &= \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\
\boldsymbol{T}_{3,1} = \boldsymbol{T}_{1,3} &= \mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)
\end{aligned}$$

where μ is the dynamic viscosity and λ is the second viscosity. λ is not well known, but for gases, $\lambda = -\frac{2}{3}\mu$ is a good working approximation [33].

Fluids flowing at low speeds (less than about 0.3 times the speed of sound[34, 5]) can be modelled as incompressible. For incompressible, Newtonian fluids, the $\lambda\nabla\cdot\boldsymbol{u}$ term is zero.

If the viscosity μ is constant, it can be moved in front of the divergence operator. The viscosity term for an incompressible, Newtonian fluid with constant viscosity now becomes:

$$\begin{aligned}
\nabla\cdot\boldsymbol{T} &= \frac{\partial \boldsymbol{T}_x}{\partial x} + \frac{\partial \boldsymbol{T}_y}{\partial y} + \frac{\partial \boldsymbol{T}_z}{\partial z} \\
&= \frac{\partial}{\partial x}\begin{bmatrix} \mu\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\right) \\ \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\ \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \end{bmatrix} + \frac{\partial}{\partial y}\begin{bmatrix} \mu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right) \\ \mu\left(\frac{\partial v}{\partial y} + \frac{\partial v}{\partial y}\right) \\ \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \end{bmatrix} + \frac{\partial}{\partial z}\begin{bmatrix} \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \\ \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\ \mu\left(\frac{\partial w}{\partial z} + \frac{\partial w}{\partial z}\right) \end{bmatrix} \\
&= \mu\left(\begin{bmatrix} \frac{\partial}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial}{\partial z}\frac{\partial w}{\partial x} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial}{\partial z}\frac{\partial w}{\partial y} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial}{\partial z}\frac{\partial w}{\partial z} \end{bmatrix} + \begin{bmatrix} \frac{\partial}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\frac{\partial u}{\partial y} + \frac{\partial}{\partial z}\frac{\partial u}{\partial z} \\ \frac{\partial}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial}{\partial z}\frac{\partial v}{\partial z} \\ \frac{\partial}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial}{\partial y}\frac{\partial w}{\partial y} + \frac{\partial}{\partial z}\frac{\partial w}{\partial z} \end{bmatrix}\right) \\
&= \mu\left(\begin{bmatrix} \frac{\partial}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial}{\partial x}\frac{\partial v}{\partial y} + \frac{\partial}{\partial x}\frac{\partial w}{\partial z} \\ \frac{\partial}{\partial y}\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial}{\partial y}\frac{\partial w}{\partial z} \\ \frac{\partial}{\partial z}\frac{\partial u}{\partial x} + \frac{\partial}{\partial z}\frac{\partial v}{\partial y} + \frac{\partial}{\partial z}\frac{\partial w}{\partial z} \end{bmatrix} + \begin{bmatrix} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \\ \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \\ \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \end{bmatrix}\right) \\
&= \mu\left(\nabla\underbrace{\nabla\cdot\boldsymbol{u}}_{0} + \nabla^2\boldsymbol{u}\right) = \mu\nabla^2\boldsymbol{u}
\end{aligned}$$

2.5 External forces

$$\boldsymbol{f} \tag{2.18}$$

External forces are forces that are not caused by the interaction between fluid particles. Examples are gravity, electromagnetic force, centrifugal force and Coriolis force.

For gravity, the force is the product of mass and the gravitational acceleration. The force per volume is thus the density ρ times the gravitational acceleration $\boldsymbol{g}$.

2.6 Other forces

It is possible to append even more forces to the right hand side of equation (2.1). In multi-phase flow, surface tension is one such force. See section 5.5 for more information on surface tension.

Chapter 3

The projection method

Since there is no known analytical solution of the momentum and continuity equations, they must be solved numerically. If the fluid is incompressible and Newtonian, the equations can be written as:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla p + \nabla\cdot\left(\mu\left((\nabla\boldsymbol{u}) + (\nabla\boldsymbol{u})^T\right)\right) + \boldsymbol{f} \tag{3.1}$$

$$\nabla\cdot\boldsymbol{u} = 0 \tag{3.2}$$

Discretise the momentum equation with respect to time using an explicit scheme (forward difference). In an explicit scheme, the following approximates the time derivative:

$$\left(\frac{\partial \boldsymbol{u}}{\partial t}\right)^n \approx \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} \tag{3.3}$$

where the superscripts n and $n+1$ indicate the time level, not exponents. The equations are discretised in space later in section 4. For the time being, it is easier to handle the continuous functions.

The naive approach, which will not work, is to try to insert the time derivative approximation into equation (3.1):

$$\begin{aligned}\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} + \boldsymbol{u}^n\cdot\nabla\boldsymbol{u}^n = -\frac{1}{\rho^n}\nabla p^n \\ +\frac{1}{\rho^n}\nabla\cdot\left(\mu^n\left((\nabla\boldsymbol{u}^n) + (\nabla\boldsymbol{u}^n)^T\right)\right) + \frac{1}{\rho^n}\boldsymbol{f}^n\end{aligned} \tag{3.4}$$

$$\frac{u^{n+1}-u^n}{\Delta t}+u^n\frac{\partial u^n}{\partial x}+v^n\frac{\partial u^n}{\partial y}=-\frac{1}{\rho^n}\frac{\partial p^n}{\partial x}+\frac{2}{\rho^n}\frac{\partial}{\partial x}\left(\mu^n\frac{\partial u}{\partial x}\right)+\frac{1}{\rho^n}\frac{\partial}{\partial y}\left(\mu^n\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{1}{\rho^n}f_x^n \quad (3.5)$$

$$\frac{v^{n+1}-v^n}{\Delta t}+u^n\frac{\partial v^n}{\partial x}+v^n\frac{\partial v^n}{\partial y}=-\frac{1}{\rho^n}\frac{\partial p^n}{\partial y}+\frac{1}{\rho^n}\frac{\partial}{\partial x}\left(\mu^n\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{2}{\rho^n}\frac{\partial}{\partial y}\left(\mu^n\frac{\partial v}{\partial y}\right)+\frac{1}{\rho^n}f_y^n \quad (3.6)$$

where $\boldsymbol{u}^{n+1}$ is the unknown velocity at the next time level, and $\boldsymbol{u}^n$ and p^n are the known velocity and pressure at the current time level. ρ^n and μ^n are the known density and viscosity. Δt is the difference in time between time level n and $n+1$.

These equations are easy to solve for $\boldsymbol{u}^{n+1}$. ρ^{n+1} and μ^{n+1} are calculated separately as described in section 5. However, there are two problems with the naive approach:

- The continuity equation $\nabla\cdot\boldsymbol{u}=0$ has not been applied, so in general $\nabla\cdot\boldsymbol{u}^{n+1}\neq 0$.

- There is no way to find the unknown pressure at the next time level, p^{n+1}, with the naive discretisation since it does not appear in the equations.

Instead of scrapping the naive approach completely, modify the momentum equation slightly. Insert the unknown pressure p^{n+1} instead of the known pressure p^n. Now both the unknown velocity and the unknown pressure at the next time level appear:

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^n}{\Delta t}+\boldsymbol{u}^n\cdot\nabla\boldsymbol{u}^n=-\frac{1}{\rho^n}\nabla p^{n+1}+\frac{1}{\rho^n}\nabla\cdot\left(\mu^n\left((\nabla\boldsymbol{u}^n)+(\nabla\boldsymbol{u}^n)^T\right)\right)+\frac{1}{\rho^n}\boldsymbol{f}^n \quad (3.7)$$

$$\nabla\cdot\boldsymbol{u}^{n+1}=0 \quad (3.8)$$

$$\frac{u^{n+1}-u^n}{\Delta t}+u^n\frac{\partial u^n}{\partial x}+v^n\frac{\partial u^n}{\partial y}=-\frac{1}{\rho^n}\frac{\partial p^{n+1}}{\partial x}+\frac{2}{\rho^n}\frac{\partial}{\partial x}\left(\mu^n\frac{\partial u}{\partial x}\right)+\frac{1}{\rho^n}\frac{\partial}{\partial y}\left(\mu^n\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{1}{\rho^n}f_x^n \quad (3.9)$$

$$\frac{v^{n+1}-v^n}{\Delta t}+u^n\frac{\partial v^n}{\partial x}+v^n\frac{\partial v^n}{\partial y}=-\frac{1}{\rho^n}\frac{\partial p^{n+1}}{\partial y}+\frac{1}{\rho^n}\frac{\partial}{\partial x}\left(\mu^n\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{2}{\rho^n}\frac{\partial}{\partial y}\left(\mu^n\frac{\partial v}{\partial y}\right)+\frac{1}{\rho^n}f_y^n \quad (3.10)$$

$$\frac{\partial u^{n+1}}{\partial x}+\frac{\partial v^{n+1}}{\partial y}=0 \quad (3.11)$$

It looks like one has to solve the equations for $\boldsymbol{u}^{n+1}$ and p^{n+1} simultaneously, which is fully possible, but with the projection method one can solve the equations in steps. This is more efficient because the linear equation systems that need to be solved become much smaller.

In the projection method one first finds an approximation of the velocity and then finds the pressure and velocity corrections that are needed to fulfil the equations.

The first step is to find the approximate velocity, called intermediate or tentative velocity $\boldsymbol{u}^*$. Though p^{n+1} is unknown in equation (3.7), one can estimate the velocity from the terms that are known. Instead of p^{n+1}, either use p^n or ignore the pressure.

$$\frac{\boldsymbol{u}^*-\boldsymbol{u}^n}{\Delta t}+\boldsymbol{u}^n\cdot\nabla\boldsymbol{u}^n=-\frac{\beta}{\rho^n}\nabla p^n+\frac{1}{\rho^n}\nabla\cdot\left(\mu^n\left((\nabla\boldsymbol{u}^n)+(\nabla\boldsymbol{u}^n)^T\right)\right)+\frac{\boldsymbol{f}^n}{\rho^n} \quad (3.12)$$

where β can be 0 or 1 depending on whether the pressure is included or not[14].

To ensure that $\nabla\cdot\boldsymbol{u}^{n+1}=0$, "project" the tentative velocity $\boldsymbol{u}^*$ into the solenoidal vector function space ($\{\boldsymbol{f}\mid\nabla\cdot\boldsymbol{f}=0$ everywhere$\}$). First, subtract equation (3.12) from equation (3.7):

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^*}{\Delta t}=-\frac{1}{\rho^n}\nabla(p^{n+1}-\beta p^n) \quad (3.13)$$

$$\frac{u^{n+1} - u^*}{\Delta t} = -\frac{1}{\rho^n}\frac{\partial}{\partial x}(p^{n+1} - \beta p^n) \tag{3.14}$$

$$\frac{v^{n+1} - v^*}{\Delta t} = -\frac{1}{\rho^n}\frac{\partial}{\partial y}(p^{n+1} - \beta p^n) \tag{3.15}$$

For simplicity, let $\phi \stackrel{\text{def}}{=} p^{n+1} - \beta p^n$. Take the divergence on each side, then apply the continuity equation $\nabla \cdot \boldsymbol{u}^{n+1} = 0$:

$$\frac{\nabla \cdot \boldsymbol{u}^{n+1} - \nabla \cdot \boldsymbol{u}^*}{\Delta t} = -\nabla \cdot \left(\frac{1}{\rho^n}\nabla(p^{n+1} - \beta p^n)\right) \tag{3.16}$$

$$\frac{1}{\Delta t}\nabla \cdot \boldsymbol{u}^* = \nabla \cdot \left(\frac{1}{\rho^n}\nabla\phi\right) \tag{3.17}$$

$$\frac{1}{\Delta t}\left(\frac{\partial u^{n+1}}{\partial x} - \frac{\partial u^*}{\partial x}\right) = -\frac{\partial}{\partial x}\left(\frac{1}{\rho^n}\frac{\partial}{\partial x}(p^{n+1} - \beta p^n)\right) \tag{3.18}$$

$$\frac{1}{\Delta t}\left(\frac{\partial v^{n+1}}{\partial y} - \frac{\partial v^*}{\partial y}\right) = -\frac{\partial}{\partial y}\left(\frac{1}{\rho^n}\frac{\partial}{\partial y}(p^{n+1} - \beta p^n)\right) \tag{3.19}$$

$$\frac{1}{\Delta t}\left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y}\right) = \frac{\partial}{\partial x}\left(\frac{1}{\rho^n}\frac{\partial}{\partial x}\phi\right) + \frac{\partial}{\partial y}\left(\frac{1}{\rho^n}\frac{\partial}{\partial y}\phi\right) \tag{3.20}$$

If ρ is constant in space, it can be moved outside the divergence operator, and the equation becomes a Poisson equation. This equation is often called the pressure Poisson equation (PPE). I will hereafter refer to equation (3.17) as the PPE also when ρ is not constant, even if not strictly correct. How the PPE is solved is covered in section 4.5. The PPE can be written as $\nabla \cdot (\frac{1}{\rho}\nabla\phi) = f$ where

$$f = \frac{1}{\Delta t}\nabla \cdot \boldsymbol{u}^* = \frac{1}{\Delta t}\left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y}\right) \tag{3.21}$$

Since $\boldsymbol{u}^*$ is known, ϕ can be found and $\boldsymbol{u}^{n+1}$ and p^{n+1} can be calculated by using equation (3.13) and $\phi \stackrel{\text{def}}{=} p^{n+1} - \beta p^n$ as follows:

$$p^{n+1} = \beta p^n + \phi \tag{3.22}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \frac{\Delta t}{\rho^n}\nabla\phi \tag{3.23}$$

$$u^{n+1} = u^* - \frac{\Delta t}{\rho^n} \frac{\partial \phi}{\partial x} \tag{3.24}$$

$$v^{n+1} = v^* - \frac{\Delta t}{\rho^n} \frac{\partial \phi}{\partial y} \tag{3.25}$$

The tentative velocity $\boldsymbol{u}^*$ makes the projection method elegant and compact, but does not have a particular physical meaning. It may be easier to understand the derivation of the projection method without introducing the tentative velocity: $\boldsymbol{u}^{n+1}$ can be eliminated from equation (3.7) by applying the divergence operator directly on both sides. This results in a Poisson-like equation which can be solved with respect to the unknown pressure p^{n+1}. The pressure p^{n+1} can then be plugged back into equation (3.7) which is solved for $\boldsymbol{u}^{n+1}$.

3.1 Velocity boundary conditions

As usual with differential equations, one needs to supplement the Navier-Stokes equations with boundary conditions. The tentative velocity $\boldsymbol{u}^*$ can be found everywhere in the interior of the domain from equation (3.12), but not on the boundary. Though the velocity boundary conditions may be known, one cannot necessarily apply those to the tentative velocity, since the tentative velocity does not have a physical interpretation. How velocity boundary conditions, tentative velocity boundary conditions and PPE boundary conditions are connected is explained in section 4.7.

There are a few typical types of boundaries, each with its own set of velocity boundary conditions.

3.1.1 No-slip

The no-slip boundary is used for walls. The velocity is set to zero at the wall boundary. There is no flow across the wall boundary and the flow does not slip along the wall.

3.1.2 Symmetry

The symmetry boundary is used for symmetry axes or planes. Because of symmetry, the flow through the plane must be zero. The tangential velocity must be equal on each side of the symmetry plane, therefore the normal derivative of the tangential velocity must be zero.

3.1.3 Inlet

The inlet boundary is used if the flow into (or out of) the domain is known. The normal component of the velocity is set to some prescribed value and the tangential component is set to zero.

3.1.4 Outlet

When fluid is to flow freely across the boundary, the velocity gradient at the boundary is set to zero.

Outlets should be placed far away from any obstacles such that the flow is allowed to develop and stabilise before reaching the boundary. If the flow is not fully developed at the boundary, the Neumann boundary conditions used for the velocity would be physically incorrect and lead to inaccurate results[33].

3.1.5 Prescribed pressure

The pressure can be fixed at the boundary to induce flow driven by a pressure difference. The normal derivative of the tangential velocity is set to zero.

Pressure boundaries should also be placed far away from any obstacles such that the flow is allowed to develop and stabilise before reaching the boundary.

Condition	2D		3D		
	x-axis	y-axis	xy-plane	yz-plane	xz-plane
No-slip	$u = 0$ $v = 0$	$u = 0$ $v = 0$	$u = 0$ $v = 0$ $w = 0$	$u = 0$ $v = 0$ $w = 0$	$u = 0$ $v = 0$ $w = 0$
Symmetry	$\frac{\partial u}{\partial y} = 0$ $v = 0$	$u = 0$ $\frac{\partial v}{\partial x} = 0$	$\frac{\partial u}{\partial z} = 0$ $\frac{\partial v}{\partial z} = 0$ $w = 0$	$u = 0$ $\frac{\partial v}{\partial x} = 0$ $\frac{\partial w}{\partial x} = 0$	$\frac{\partial u}{\partial y} = 0$ $v = 0$ $\frac{\partial w}{\partial y} = 0$
Inlet	$u = 0$ $v = C$	$u = C$ $v = 0$	$u = 0$ $v = 0$ $w = C$	$u = C$ $v = 0$ $w = 0$	$u = 0$ $v = C$ $w = 0$
Outlet	$\frac{\partial u}{\partial y} = 0$ $\frac{\partial v}{\partial y} = 0$	$\frac{\partial u}{\partial x} = 0$ $\frac{\partial v}{\partial x} = 0$	$\frac{\partial u}{\partial z} = 0$ $\frac{\partial v}{\partial z} = 0$ $\frac{\partial w}{\partial z} = 0$	$\frac{\partial u}{\partial x} = 0$ $\frac{\partial v}{\partial x} = 0$ $\frac{\partial w}{\partial x} = 0$	$\frac{\partial u}{\partial y} = 0$ $\frac{\partial v}{\partial y} = 0$ $\frac{\partial w}{\partial y} = 0$
Pressure	$\frac{\partial u}{\partial y} = 0$ $p = C$	$p = C$ $\frac{\partial v}{\partial x} = 0$	$\frac{\partial u}{\partial z} = 0$ $\frac{\partial v}{\partial z} = 0$ $p = C$	$p = C$ $\frac{\partial v}{\partial x} = 0$ $\frac{\partial w}{\partial x} = 0$	$\frac{\partial u}{\partial y} = 0$ $p = C$ $\frac{\partial w}{\partial y} = 0$

Table 3.1: Velocity boundary conditions for different boundaries. C is an arbitrary, prescribed value.

3.2 Poisson boundary conditions

The PPE needs boundary conditions consistent with the velocity boundary conditions. The no-slip condition is described in detail in [8]. A no-slip boundary condition for the PPE can be found in two ways. One way is to multiply each side of equation (3.13) with a unit normal $\boldsymbol{n}$ on the domain boundary:

$$\frac{1}{\Delta t}\boldsymbol{n}\cdot(\boldsymbol{u}^{n+1}-\boldsymbol{u}^{*})=-\frac{1}{\rho^{n}}\boldsymbol{n}\cdot\nabla(p^{n+1}-\beta p^{n})=-\frac{1}{\rho^{n}}\boldsymbol{n}\cdot\nabla\phi \tag{3.26}$$

which is a Neumann boundary condition. If $\boldsymbol{u}^{*}$ is set equal to the prescibed velocity $\boldsymbol{u}^{n+1}$ on the boundary, we get a homogeneous Neumann boundary condition which is easy to implement.

Another way to derive a PPE boundary condition is to multiply each side of equation (3.13) with a unit tangent $\boldsymbol{t}$ on the domain boundary:

$$\frac{1}{\Delta t}\boldsymbol{t}\cdot(\boldsymbol{u}^{n+1}-\boldsymbol{u}^{*})=-\frac{1}{\rho^{n}}\boldsymbol{t}\cdot\nabla(p^{n+1}-\beta p^{n})=-\frac{1}{\rho^{n}}\boldsymbol{t}\cdot\nabla\phi \tag{3.27}$$

which indirectly leads to a Dirichlet boundary condition. Assume that $(\boldsymbol{u}^{n+1}-\boldsymbol{u}^{*})$ is known on the boundary. If ϕ is fixed at one point on the boundary, one can integrate tangentially to find ϕ anywhere else on the boundary.

I choose to use the homogeneous Neumann boundary condition, since according to [8], only the Neumann boundary condition is always appropriate. I use the Neumann boundary condition for the PPE on no-slip, symmetry and inlet boundaries. The motivation behind the Neumann boundary condition is mathematical, not physical. Thus, the normal derivative of the pressure on the boundary has no physical meaning.

A Dirichlet boundary condition can be used in the PPE to create a pressure difference between two or more boundaries, causing a flow from high to low pressure.

The mathematical rationale behind the implementation of the outlet boundary is vague to me. Anyhow, the homogeneous Neumann boundary condition is used for the PPE on outlets[9].

The PPE with only the Neumann boundary condition has either no solution or infinitely many solutions. Let Ω be the fluid domain and let $\delta\Omega$ be its boundary. As usual, $\boldsymbol{n}$ is the unit normal on the boundary. We have the following problem:

$$\nabla\cdot\left(\frac{1}{\rho^{n}}\nabla\phi\right)=f \text{ where } \boldsymbol{n}\cdot\nabla\phi=0 \text{ on } \delta\Omega \tag{3.28}$$

where $f=\frac{1}{\Delta t}\nabla\cdot\boldsymbol{u}^{*}$.

Integrate both sides over the domain Ω:

$$\int_\Omega \nabla \cdot \left(\frac{1}{\rho^n} \nabla \phi \right) d\Omega = \int_\Omega f d\Omega \tag{3.29}$$

The divergence theorem states that:

$$\int_\Omega \nabla \cdot \left(\frac{1}{\rho^n} \nabla \phi \right) d\Omega = \int_{\delta\Omega} \boldsymbol{n} \cdot \left(\frac{1}{\rho^n} \nabla \phi \right) dS = \int_{\delta\Omega} \frac{1}{\rho^n} \underbrace{\boldsymbol{n} \cdot \nabla \phi}_{=0} dS \tag{3.30}$$

The last integral is a surface integral over the domain boundary. Since $\boldsymbol{n} \cdot \nabla \phi = 0$, the left hand side of equation (3.29) must be zero, and hence the right hand side must also be zero. Now, we have shown that if ϕ is a solution of the Poisson equation, then $\int_\Omega f d\Omega$ must be zero. In other words, $\int_\Omega f d\Omega = 0$ is necessary for the existence of a solution. This means that the amount flowing into the domain – as defined by the tentative velocity $\boldsymbol{u}^*$ – must equal the amount flowing out of the domain. If this is not the case, the tentative velocity $\boldsymbol{u}^*$ must be adjusted somehow on the boundary such that the net flow becomes zero. How this can be done is described in section 4.7.2.

If there is a solution ϕ to the PPE with only Neumann boundary conditions, we can add any constant C and still have a solution $\phi + C$. Thus, there are infinitely many solutions. If the solution method used on the PPE requires one unique solution, ϕ must be fixed in exactly one point, for instance by setting ϕ in the lower left corner to zero: $\phi_{0,0} = 0$.

$\int_\Omega f d\Omega = 0$ is not necessary when using Dirichlet boundary condition instead of Neumann on parts of the boundary.

Chapter 4

Discretisation

The continuous problem must be reduced to a discrete problem so that a computer can digest it. There are several methods for doing this, but I will use the simplest one, the finite difference method (FDM).

I will focus on the 2D Navier-Stokes equations from here, but the discretisation in 3D follows analogously. Instead of solving the continuous problem, the problem is solved in a limited number of points or nodes. The nodes are arranged in a grid of rows and columns, and a function value is stored in each node.

Both the unknown functions and their derivatives appear in differential equations. We get rid of the unknown derivatives by using finite difference approximations in their place. The accuracy of the solution depends on the distance between the nodes and the approximations used for the derivatives.

Mostly central difference is used in the interior of the domain. One-sided difference is sometimes used on the boundary and for upwind differencing in the interior. Upwind differencing is used to stabilise the simulation.

4.1 Staggered grid

In a non-staggered grid, values of different functions are stored at the same location. Each node contains both a velocity vector and a pressure value (see figure 4.1). This is the most obvious grid to choose, but will not be used here for reasons explained below.

The problem with the non-staggered grid when solving Navier-Stokes equations is that we easily get pressure oscillations. We wish to use a central difference since it has higher order of accuracy than a one-sided (upwind) difference. The pressure gradient at position (x, y) with a central difference is:

$$\frac{\partial}{\partial x}p(x,y,t) \approx \frac{p(x+\Delta x,y,t) - p(x-\Delta x,y,t)}{2\Delta x} \tag{4.1}$$

$$\frac{\partial}{\partial y}p(x,y,t) \approx \frac{p(x,y+\Delta y,t) - p(x,y-\Delta y,t)}{2\Delta y} \tag{4.2}$$

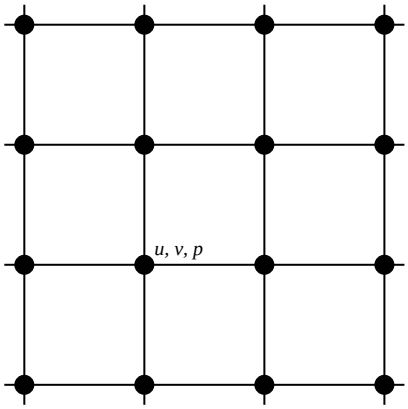


Figure 4.1: Non-staggered grid. Velocity and pressure values are stored in common nodes, here shown as black dots.

where Δx and Δy are the horizontal and vertical node spacing respectively.

We see that the centre node $p(x, y, t)$ itself does not appear in the approximation of $\nabla p(x, y, t)$. Let the nodes be alternately black and white like on a chess board. Let each white node have a constant value C_W and every black node have a different constant value C_B. Since every second node is black and every other second is white, $p(x + \Delta x, y, t) - p(x - \Delta x, y, t)$ will either be $C_W - C_W$ or $C_B - C_B$, which both evaluates to zero. Similarly for the y-component. Evaluating ∇p in any node will now result in a zero vector no matter what value C_W and C_B have. A zero pressure gradient field suggests that the pressure is constant, even though we know it oscillates between C_W and C_B. The oscillation problem is described in numerous books about numerical solution of differential equations. To learn more, the reader can look up "convection-diffusion equation" or "advection-diffusion equation" and "upwind differences" in any of these books (e.g. [15, 33, 34]).

To avoid the oscillation problem, we use a staggered grid. In a staggered grid, each function is stored in its own subgrid shifted half a cell in one or more directions relative to the other subgrids. All the subgrids have the same grid spacing. In 2D, the staggered grid we are going to use has three subgrids, one for the pressure p, one for the horizontal velocity component u and one for the vertical velocity component v. The grid is not staggered in time. The placement of the subgrids is shown in figure 4.2 where the horizontal arrows indicate u-nodes, vertical arrows indicate v-nodes and the dots indicate p-nodes. In figure 4.2, each pressure node is placed in the middle of a square which is called a "cell". In the staggered grid, the velocity components can be thought of as flow though cell walls from one cell to its neighbouring cells.

In the staggered grid, one can approximate the pressure gradient by

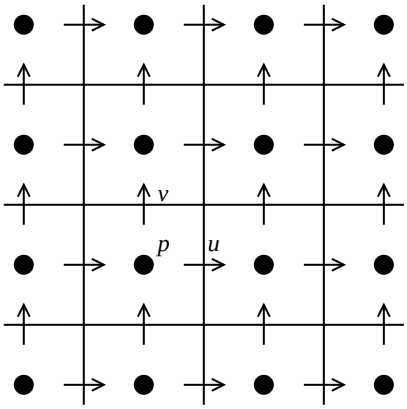


Figure 4.2: Staggered grid. p-nodes are shown as black dots, u-nodes as horizontal arrows and v-nodes as vertical arrows.

using adjacent nodes:

$$\frac{\partial}{\partial x}p(x,y,t) \approx \frac{p(x+\Delta x/2,y,t)-p(x-\Delta x/2,y,t)}{\Delta x} \tag{4.3}$$

$$\frac{\partial}{\partial y}p(x,y,t) \approx \frac{p(x,y+\Delta y/2,t)-p(x,y-\Delta y/2,t)}{\Delta y} \tag{4.4}$$

To evaluate $\frac{\partial p}{\partial x}$ at some location, a pressure node half a cell width to the left and to the right of this location are needed. Looking at figure 4.2, one can see that this location coincides with the u-nodes. Similarly, $\frac{\partial p}{\partial y}$ can be evaluated at v-nodes.

4.2 Ghost cells

Around the simulation domain, a strip of ghost cells is added (see figure 4.3). The ghost cells are so called because they are not part of the computational domain, but still contain values that contribute to the calculations. Ghost cells are not strictly needed, but make it easier to impose boundary conditions. How this is done is explained in section 4.7.

4.3 Indexing

It is common to assign indices to the nodes instead of using their coordinates. For some function f, I define the indexing as follows:

$$f^n_{i,j} \stackrel{\text{def}}{=} f(x_0+i\Delta x, y_0+j\Delta y, t_0+n\Delta t) \tag{4.5}$$

where (x_0, y_0) is the location of the bottom left cell centre in the domain and t_0 is the initial time. When the time level is obvious, I will omit the

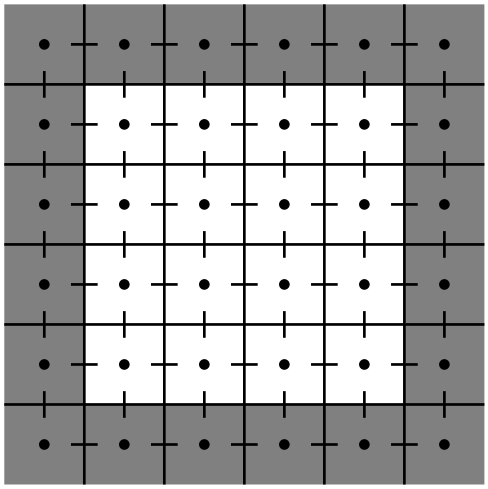


Figure 4.3: Ghost cells are shown in grey around a domain of 4×4 cells. Dots indicate p-nodes, horizontal bars indicate u-nodes and vertical bars indicate v-nodes.

superscript. Looking at figure 4.4, we see that the pressure nodes are indexed as $p_{i,j}$, u-nodes are indexed as $u_{i+1/2,j}$ and v-nodes as $v_{i,j+1/2}$ where i and j are integers.

4.4 Calculating the tentative velocity

The tentative velocity is found by solving equation (3.12).

$$\frac{u^* - u^n}{\Delta t} + \underbrace{u^n \frac{\partial u^n}{\partial x} + v^n \frac{\partial u^n}{\partial y}}_{\text{convection}} = \underbrace{-\frac{\beta}{\rho^n} \frac{\partial p^n}{\partial x}}_{\text{pressure gradient}} + \underbrace{\frac{f_x^n}{\rho^n}}_{\text{other forces}} + \underbrace{\frac{1}{\rho^n} \left[\frac{\partial}{\partial x} \left(2\mu^n \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu^n \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right) \right]}_{\text{viscosity}} \tag{4.6}$$

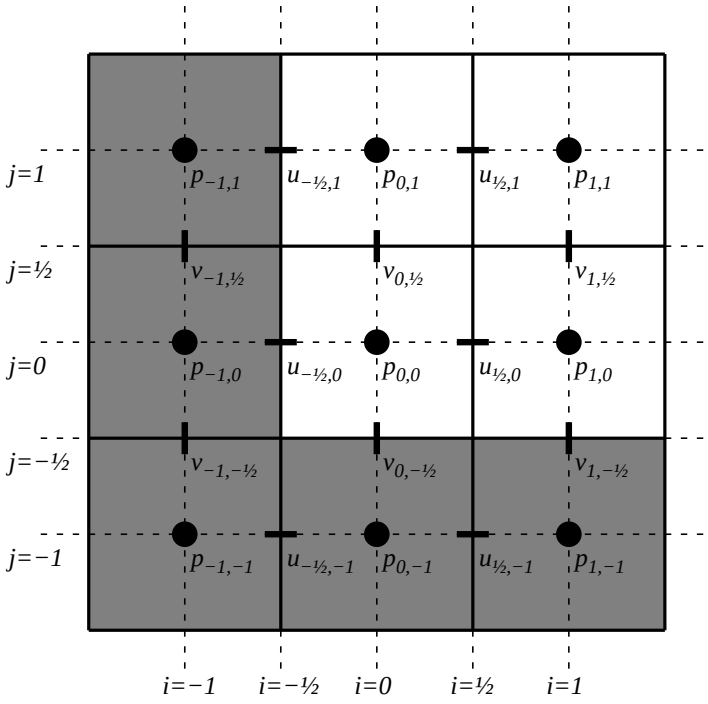


Figure 4.4: Node indexing in the lower left corner of the grid.

$$\frac{v^* - v^n}{\Delta t} + \underbrace{u^n \frac{\partial v^n}{\partial x} + v^n \frac{\partial v^n}{\partial y}}_{\text{convection}} = \underbrace{-\frac{\beta}{\rho^n}\frac{\partial p^n}{\partial y}}_{\text{pressure gradient}} + \underbrace{\frac{f_y^n}{\rho^n}}_{\text{other forces}} + \underbrace{\frac{1}{\rho^n}\left[\frac{\partial}{\partial x}\left(\mu^n\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right) + \frac{\partial}{\partial y}\left(2\mu^n \frac{\partial v}{\partial y}\right)\right]}_{\text{viscosity}} \tag{4.7}$$

The tentative velocity nodes have the same location as the usual velocity nodes, that is, u^*-nodes coincide with u-nodes, v^*-nodes with v-nodes. In this section, I will describe how the tentative velocity is calculated in the domain interior. The tentative velocity on the boundary and in the ghost cells are set by imposing boundary conditions which will be covered later.

I will only show how to approximate $u^*_{i,j}$, the x-component of the tentative velocity. The y-component is approximated analogously.

4.4.1 Density

The density nodes are located with the pressure nodes. The density at velocity nodes is approximated by the average of the two closest density nodes:

$$\rho_{i+1/2,j} \approx \frac{1}{2}(\rho_{i,j} + \rho_{i+1,j}) \tag{4.8}$$

4.4.2 Convection

The convective term does not fit snugly into the staggered grid scheme. If one uses the usual central difference to approximate derivatives for the horizontal component, one finds:

$$\begin{aligned}\left(u^n \frac{\partial u^n}{\partial x} + v^n \frac{\partial u^n}{\partial y}\right)_{i+1/2,j} &\approx u^n_{i+1/2,j} \frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x} \\ &+ v^n_{i+1/2,j} \frac{u^n_{i+1/2,j+1/2} - u^n_{i+1/2,j-1/2}}{\Delta y}\end{aligned} \tag{4.9}$$

But u-nodes does not exist every half cell width and height, and the v-nodes are not in the same location as the u-nodes. Solve the problem by rewriting the expression as $\frac{\partial(u^2)}{\partial x} + \frac{\partial(uv)}{\partial y}$ and averaging neighbouring nodes[9]. By simple differentiation using the product rule and applying the continuity

equation (3.2):

$$\begin{aligned}\frac{\partial(u^2)}{\partial x}+\frac{\partial(uv)}{\partial y} &= 2u\frac{\partial u}{\partial x}+u\frac{\partial v}{\partial y}+v\frac{\partial u}{\partial y}\\ &= \underbrace{u\frac{\partial u}{\partial x}+u\frac{\partial v}{\partial y}}_{=u\nabla\cdot\boldsymbol{u}=0}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\\ &= u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\end{aligned} \tag{4.10}$$

This shows that the left hand side is indeed an alternative expression for the convective term. Similarly for the vertical component, $u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=\frac{\partial(uv)}{\partial x}+\frac{\partial(v^2)}{\partial y}$. The horizontal component of the convective term can thus be approximated as:

$$\begin{aligned}\left(\frac{\partial(u^2)}{\partial x}+\frac{\partial(vu)}{\partial y}\right)_{i+1/2,j} &\approx \frac{u_{i+1,j}^2-u_{i,j}^2}{\Delta x}\\ &\quad+\frac{u_{i+1/2,j+1/2}v_{i+1/2,j+1/2}-u_{i+1/2,j-1/2}v_{i+1/2,j-1/2}}{\Delta y}\end{aligned} \tag{4.11}$$

Average neighbouring nodes, for instance $u_{i,j}\approx(u_{i-1/2,j}+u_{i+1/2,j})/2$, to get (see figure 4.5):

$$\begin{aligned}\left(\frac{\partial(u^2)}{\partial x}+\frac{\partial(vu)}{\partial y}\right)_{i+1/2,j} &\approx \frac{1}{\Delta x}\left(\frac{u_{i+1/2,j}+u_{i+3/2,j}}{2}\right)^2\\ &\quad-\frac{1}{\Delta x}\left(\frac{u_{i-1/2,j}+u_{i+1/2,j}}{2}\right)^2\\ &\quad+\frac{1}{\Delta y}\left(\frac{u_{i+1/2,j}+u_{i+1/2,j+1}}{2}\right)\left(\frac{v_{i,j+1/2}+v_{i+1,j+1/2}}{2}\right)\\ &\quad-\frac{1}{\Delta y}\left(\frac{u_{i+1/2,j}+u_{i+1/2,j-1}}{2}\right)\left(\frac{v_{i,j-1/2}+v_{i+1,j-1/2}}{2}\right)\end{aligned} \tag{4.12}$$

This scheme is a second order approximation.

In [9], Griebel adds upwind differencing to improve stability, and I choose to do the same. An upwind difference is a one-sided difference shifted in negative flow (upwind) direction. Upwind differencing is added to the right hand side of equation (4.12):

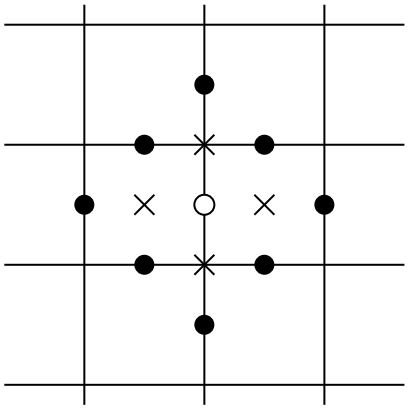


Figure 4.5: The dots indicate nodes used in the discretisation of the convective term. The crosses are the position of the averaged values.

$$\begin{aligned}\left(\frac{\partial(u^2)}{\partial x}+\frac{\partial(vu)}{\partial y}\right)_{i+1/2,j} &\approx \dots \\ &+\frac{\gamma}{4\Delta x}\left(|u_{i+1/2,j}+u_{i+3/2,j}|(u_{i+3/2,j}-u_{i+1/2,j})\right. \\ &-\left.|u_{i-1/2,j}+u_{i+1/2,j}|(u_{i+1/2,j}-u_{i-1/2,j})\right) \\ &+\frac{\gamma}{4\Delta y}\left(|v_{i,j+1/2}+v_{i+1,j+1/2}|(u_{i+1/2,j+1}-u_{i+1/2,j})\right. \\ &-\left.|v_{i,j-1/2}+v_{i+1,j-1/2}|(u_{i-1/2,j}-u_{i-1/2,j-1})\right)\end{aligned} \tag{4.13}$$

where γ is an constant factor between 0 and 1 deciding how much upwind differencing to use. This factor must be adjusted manually by the user. The upwind differencing scheme is only a first order approximation, so if stable, $\gamma = 0$ should give the most accurate result.

4.4.3 Pressure gradient

The pressure gradient is approximated with a central difference (see figure 4.6):

$$\left(\frac{\partial p}{\partial x}\right)_{i,j} \approx \frac{p_{i+1/2,j}-p_{i-1/2,j}}{\Delta x} \tag{4.14}$$

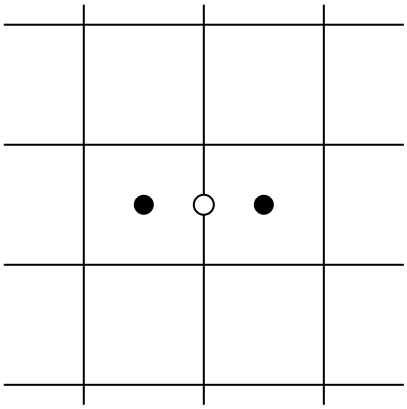


Figure 4.6: Nodes used in the discretisation of the pressure gradient term.

4.4.4 Viscosity

Viscosity terms are approximated by (see figure 4.7):

$$\left(\frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right]\right)_{i+1/2,j} \approx \frac{2}{\Delta x}\left[\left(\mu\frac{\partial u}{\partial x}\right)_{i+1,j} - \left(\mu\frac{\partial u}{\partial x}\right)_{i,j}\right] \tag{4.15}$$

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} \approx \frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} \tag{4.16}$$

$$\begin{aligned}\left(\frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right]\right)_{i+1/2,j} &\approx \frac{1}{\Delta y}\left(\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right)_{i+1/2,j+1/2} \\ &\quad - \frac{1}{\Delta y}\left(\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right)_{i+1/2,j-1/2}\end{aligned} \tag{4.17}$$

$$\left(\frac{\partial u}{\partial y}\right)_{i+1/2,j+1/2} \approx \frac{u_{i+1/2,j+1} - u_{i+1/2,j}}{\Delta y} \tag{4.18}$$

$$\left(\frac{\partial v}{\partial x}\right)_{i+1/2,j+1/2} \approx \frac{v_{i+1,j+1/2} - v_{i,j+1/2}}{\Delta x} \tag{4.19}$$

$$\mu_{i+1/2,j1/2} \approx \frac{1}{4}(\mu_{i,j} + \mu_{i+1,j} + \mu_{i,j+1} + \mu_{i+1,j+1}) \tag{4.20}$$

$$\tag{4.21}$$

The tentative velocity is then the combination of convection, pressure gradient, viscosity terms and additional forces:

$$u^* = u^n + \Delta t(-\text{convection} - \text{pressure gradient} + \text{viscosity} + \frac{f_x^n}{\rho^n}) \tag{4.22}$$

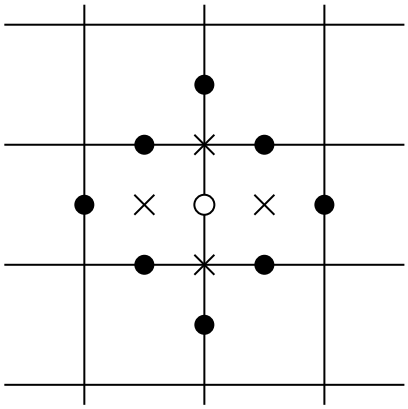


Figure 4.7: Nodes used in the discretisation of the viscosity term. The dots indicate velocity nodes needed in the discretisation. The crosses indicate the location of intermediate velocity derivatives.

4.5 The pressure Poisson equation

The next step is to find the pressure at the next time level by solving the PPE (3.17). Substituting derivatives with discrete approximations, the PPE becomes:

$$\begin{aligned}
&\frac{1}{\Delta t}\left(\frac{u^*_{i+1/2,j} - u^*_{i-1/2,j}}{\Delta x} + \frac{v^*_{i,j+1/2} - v^*_{i,j-1/2}}{\Delta y}\right) \\
&= \rho^n_{i+1/2,j}\frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x^2} - \rho^n_{i-1/2,j}\frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x^2} \\
&+\rho^n_{i,j+1/2}\frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y^2} - \rho^n_{i,j-1/2}\frac{\phi_{i,j} - \phi_{i,j-1}}{\Delta y^2}
\end{aligned} \tag{4.23}$$

Since the tentative velocity is known, the left hand side can be calculated. It is not possible to find the value of ϕ in a single node explicitly from this equation – a linear equation system must be solved, where ϕ in each node is treated as a separate scalar unknown. When solving the linear equation system resulting from the Poisson equation, one should use a method suitable for sparse systems, such as stationary (classic) iterative methods or Krylov subspace methods.

4.5.1 Boundary conditions

For pressure nodes near the boundary, some of the neighbouring nodes lie outside the fluid domain. Boundary conditions are used to eliminate any reference to these nodes.

If the normal derivative of the pressure is zero on the boundary, the corresponding term in equation (4.23) can be eliminated. For instance,

assume that there is a vertical boundary or obstacle wall passing between nodes $\phi_{i-1,j}$ and $\phi_{i,j}$. Since the normal derivative of the pressure is zero, we have:

$$\frac{\partial \phi}{\partial x} \approx \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x} = 0 \tag{4.24}$$

This boundary condition eliminates the second of the four addends on the right hand side of equation (4.23), thereby eliminating the reference to $\phi_{i-1,j}$[9].

Dirichlet boundary conditions are handled in a similar manner. Assume that the pressure value on the boundary is p_{bc}^{n+1}. First of all, calculate $\phi_{bc} = p_{bc}^{n+1} - \beta p_{bc}^{n}$. Since the pressure nodes do not coincide with the boundary, the boundary condition must be enforced in an averaged sense. As before, let the boundary pass between the nodes $\phi_{i-1,j}$ and $\phi_{i,j}$:

$$\frac{\phi_{i-1,j} + \phi_{i,j}}{2} = \phi_{bc} \Rightarrow \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x^2} = \frac{2(\phi_{i,j} - \phi_{bc})}{\Delta x^2} \tag{4.25}$$

Again, the boundary condition changes the corresponding term on the right hand side of equation (4.23).

If the normal derivative is zero ($\boldsymbol{n} \cdot \nabla p = 0$) all around the boundary, the linear equation system will only be solvable if the total amount of flow (defined by the tentative velocity $\boldsymbol{u}^*$) into the domain is equal to the total amount flowing out of the domain, that is, the net outflow is zero. How the outflow can be adjusted to achieve zero net outflow is described in section 4.7. If the net outflow is not zero, the best option is probably not to use Neumann boundary condition along the entire boundary, but to prescribe the pressure on some part of it.

Another problem with Neumann boundary condition on the entire boundary is that if there is a solution, there are infinitely many solutions. It is preferable if the pressure stays in the area around zero. If the pressure becomes large, the accuracy of the floating point calculations is reduced. The problem can be solved by requiring that a certain pressure node should be zero or that the average pressure should be zero.

4.6 Correcting pressure and velocity

The pressure at the next time level is found by the definition of ϕ.

$$p_{i,j}^{n+1} = \beta p_{i,j}^{n} + \phi_{i,j} \tag{4.26}$$

The velocity is found by solving equation (3.13) with respect to $\boldsymbol{u}^{n+1}$:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \frac{\Delta t}{\rho^n} \nabla \phi \tag{4.27}$$

Using a central difference for the derivatives, these equations can be approximated as:

$$u^{n+1}_{i+1/2,j} = u^*_{i+1/2,j} - \frac{\Delta t}{\rho^n_{i+1/2,j}} \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{4.28}$$

$$v^{n+1}_{i,j+1/2} = v^*_{i,j+1/2} - \frac{\Delta t}{\rho^n_{i,j+1/2}} \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{4.29}$$

Velocity nodes on the boundary should also be corrected. If the normal pressure derivative is zero, for instance on the west boundary, then $u^{n+1} = u^*$ there. If ϕ is known on the west boundary, then:

$$u^{n+1}_{-1/2,j} = u^*_{-1/2,j} - \frac{2\Delta t}{\rho^n_{-1/2,j}} \frac{\phi_{0,j} - \phi_{bc}}{\Delta x} \tag{4.30}$$

where ϕ_{bc} is the known value of ϕ on the boundary. Note the factor 2.

4.7 Velocity boundary conditions

Boundary conditions are imposed simply by setting the values of the nodes on the boundary and in the ghost cells directly. To calculate the tentative velocity $\boldsymbol{u}^*$ in a node, one needs to access neighbouring velocity nodes as shown in the figures in section 4.4. This could be problematic near the boundary where the nodes do not have neighbours in all directions. This is the reason behind the ghost cells. Instead of trying to access non-existent neighbour nodes, one can read from nodes in the ghost cells. The values in the ghost cells are adjusted to fit the boundary conditions. First, the tentative velocity $\boldsymbol{u}^*$ is calculated for all interior nodes. The tentative velocity $\boldsymbol{u}^*$ on the boundary is updated depending on the boundary conditions. The PPE is then solved and the velocity at the next time level $\boldsymbol{u}^{n+1}$ is calculated. Finally, boundary conditions are applied to $\boldsymbol{u}^{n+1}$ on the boundary and in the ghost cells.

I explain how boundary conditions are implemented for the west boundary. This can be extended analogously to the east, north and south boundary.

4.7.1 Dirichlet boundary conditions

The tentative velocity is calculated as usual.

Assume that the normal velocity u^{n+1} is known on the west boundary. The value of u^{n+1} is calculated from u^* and ϕ as shown in equation (3.13). Therefore, u^* and ϕ must be chosen such that the known u^{n+1} is the result. This is accomplished by setting $u^* = u^{n+1}$ and using the homogeneous Neumann condition $\frac{\partial \phi}{\partial x} \approx (\phi_{0,j} - \phi_{-1,j})/\Delta x = 0$ on this boundary[9].

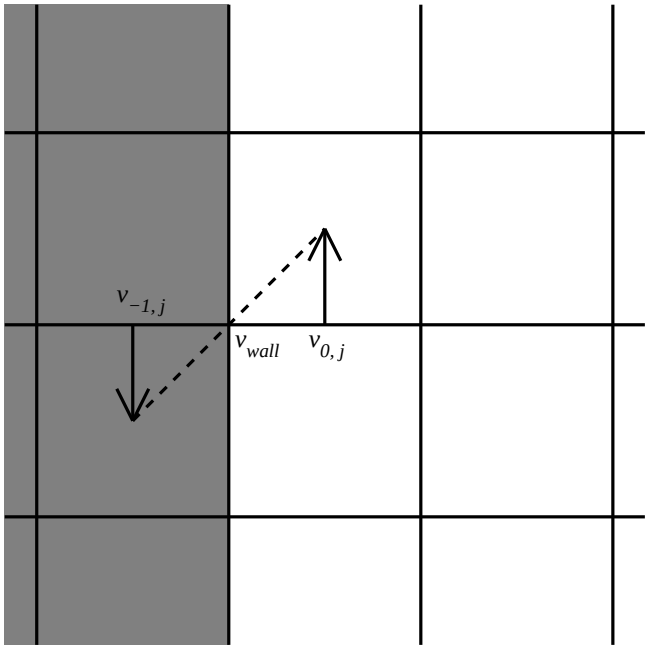


Figure 4.8: Dirichlet boundary condition for v on the west boundary.

If the tangential velocity v^{n+1} has a prescribed value v^{n+1}_{wall} on the west boundary, the values in ghost cell nodes must be adjusted such that the average on the boundary is v^{n+1}_{wall}[9] (see figure 4.8):

$$v^{n+1}_{-1,j} \leftarrow 2v^{n+1}_{wall} - v^{n+1}_{0,j} \tag{4.31}$$

The $\leftarrow$ indicates an explicit assignment.

4.7.2 Neumann boundary conditions

The tentative velocity is calculated as usual.

If the normal derivative of the tangential velocity $\frac{\partial v^{n+1}}{\partial x}$ is zero on the west boundary, the tangential velocity inside the domain is extrapolated to the ghost cells[9]. (see figure 4.9):

$$v^{n+1}_{-1,j} \leftarrow v^{n+1}_{0,j} \tag{4.32}$$

How to implement the Neumann boundary condition for the normal velocity u^{n+1} is described in [9, 33], but the method is somewhat questionable mathematically.

Let $\frac{\partial u}{\partial x} = 0$ on the west boundary. A one sided difference is used to approximate the derivative:

$$0 = \left(\frac{\partial u}{\partial x}\right)_{-1/2,j} \approx \frac{u_{1/2,j} - u_{-1/2,j}}{\Delta x} \tag{4.33}$$

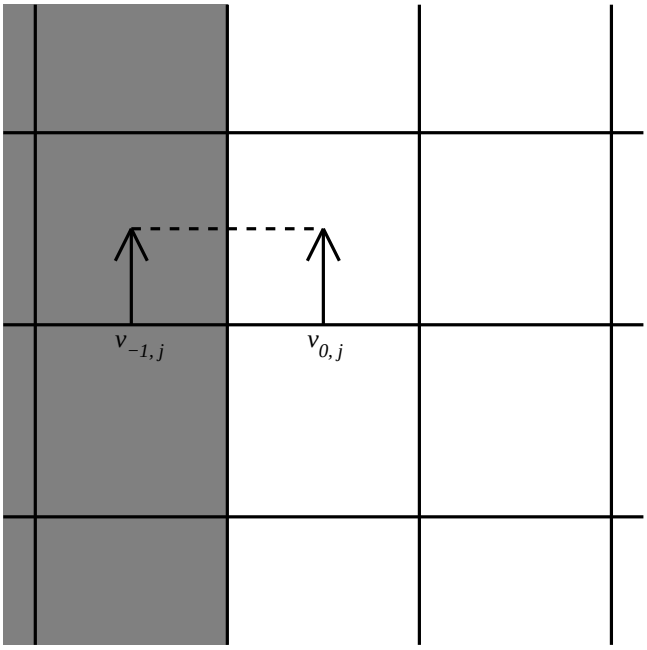


Figure 4.9: Neumann boundary condition for v on the west boundary.

Solving for the boundary velocity $u_{-1/2,j}$:

$$u_{-1/2,j} = u_{1/2,j} \tag{4.34}$$

The question now is whether to impose this condition on u^* or u^{n+1}, and how. In [9, 33], the boundary condition is applied explicitly to u^{n+1}; after u^{n+1} has been calculated for the domain interior, it is extrapolated to the boundary ($u^{n+1}_{-1/2,j} = u^{n+1}_{1/2,j}$). In addition, in [33], the velocity on the boundary is scaled such that the net outflow becomes zero. However, overwriting the velocity on the boundary will prevent the continuity equation from being fulfilled in cells next to the boundary. Nonetheless, if the boundary is placed far away from obstacles such that the flow is allowed to develop and stabilise before reaching the boundary, as advised in [33], the error should become minimal.

The question of how to treat the tentative velocity u^* on the boundary remains. In [9, 33], $\boldsymbol{u}^* = \boldsymbol{u}^n$ on the west boundary. If the PPE is going to be solvable, the net outflow defined by the tentative velocity $\boldsymbol{u}^*$ must be zero. If the normal velocity component is prescribed on the entire boundary, and the net outflow is not zero, the problem is inconsistent and a solution does not exist. If the normal velocity component is not prescribed on some part of the boundary, the flow there can be adjusted somehow to make the net outflow zero. In the description of the SIMPLE algorithm in [33], the velocity at outlets is scaled such that the net outflow becomes zero. However,

the SIMPLE algorithm is not transient[1], so it is not clear if this approach is satisfactory for transient simulations. Another option is to add some constant to the velocity at the outlets such that the net outflow becomes zero. This is done in the NaSt3DGP flow solver. I implemented both methods and let the user choose which one to use.

1. *Scale*: Calculate the inflow M_{in} and outflow M_{out}. The outflow is then scaled by a factor c such that $cM_{out} = M_{in}$. Scale the velocities on outlet boundaries as follows[33]:

$$\boldsymbol{u}^*_{new} \leftarrow \boldsymbol{u}^*_{old} \frac{M_{in}}{M_{out}} \tag{4.35}$$

   Clearly, this will only work if $M_{out} \neq 0$. If $M_{out} = 0$, add a value to all outlets instead, as described below.

2. *Add*: Calculate the inflow M_{in}, outflow M_{out} and the total length L of the outlets. Adjust the velocity at vertical outlets by:

$$u^*_{new} \leftarrow u^*_{old} \frac{\Delta y}{L}(M_{in} - M_{out}) \tag{4.36}$$

   and at horizontal outlets by:

$$v^*_{new} \leftarrow v^*_{old} \frac{\Delta x}{L}(M_{in} - M_{out}) \tag{4.37}$$

Since the method in [9, 33] is probably thoroughly tested, I use it for outlets where the pressure is unknown. However, for pressure driven flows, extrapolating the velocity to the boundary will eliminate the effect of a pressure gradient across the boundary.

4.7.3 Prescribed pressure

I failed to find a good explanation on how to implement pressure driven flows. In [33], the pressure at nodes just inside the boundary are fixed when solving the PPE. This will cause those pressure nodes to work as sources or sinks, and the continuity equation is thus not fulfilled for these cells. Therefore, the velocity on the boundary is adjusted to fulfil the continuity equation afterwards. Unfortunately, the pressure nodes are half a cell width away from the boundary. I prefer a solution where the pressure is set exactly on the boundary. I therefore made my own scheme based on the one in [33].

To allow pressure driven flows, I use the same equation (3.12) to calculate the tentative velocity u^* on the boundary as inside the domain. After solving the PPE, I correct u^* on the boundary to give u^{n+1} the same way as in the

[1]Transient algorithms calculate how the flow develops over time, while non-transient algorithms calculate steady flows.

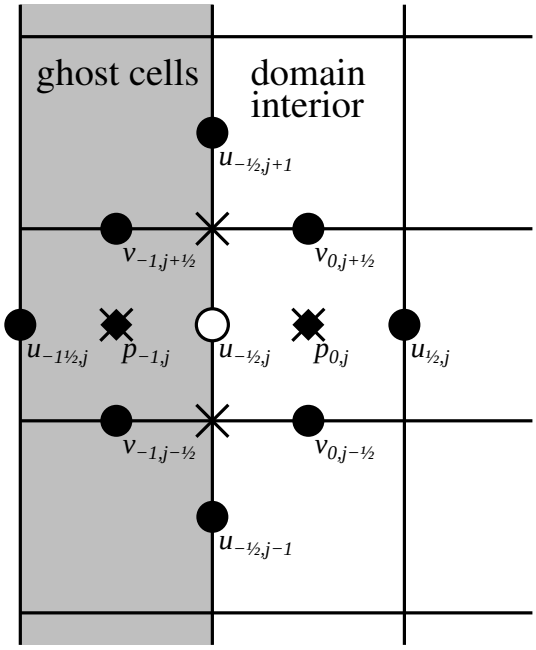


Figure 4.10: Stencil used to calculate the tentative velocity on the boundary. Dots indicate velocity nodes, crosses indicate the location of intermediate values, and the squares indicate pressure nodes.

domain interior. The stencil needed to calculate u^* on the west boundary is shown in figure 4.10. As can be seen in the figure, I have references to non-existent node values $u^n_{-3/2,j}$. Node values $v^n_{-1,j-1/2}$, $v^n_{-1,j+1/2}$ and $p^n_{-1,j}$ exist as ghost cell values. Since the west boundary is an outlet where fluid is flowing across the boundary, I assume that the continuity equation will also apply just beyond the boundary. I therefore solve the discrete continuity equation with respect to the non-existent value $u^n_{-3/2,j}$ for a ghost cell:

$$u^n_{-3/2,j} \stackrel{\text{def}}{=} u^n_{-1/2,j} - \frac{\Delta x}{\Delta y}(v^n_{-1,j+1/2} - v^n_{-1,j-1/2}) \tag{4.38}$$

I plug this into equation (3.12) to eliminate the non-existent value $u^n_{-3/2,j}$.

4.7.4 Obstacle boundary conditions

No-slip boundary condition (homogeneous Dirichlet condition for velocity) is imposed on the boundary between fluid cells and obstacle cells indicated in the mask array. If a velocity node coincides with the boundary of an obstacle, the velocity is simply set to zero. If not, the velocity immediately on the inside of the obstacle is set so that the average velocity on the boundary becomes zero, in the same manner as for ghost cells (see figure 4.11).

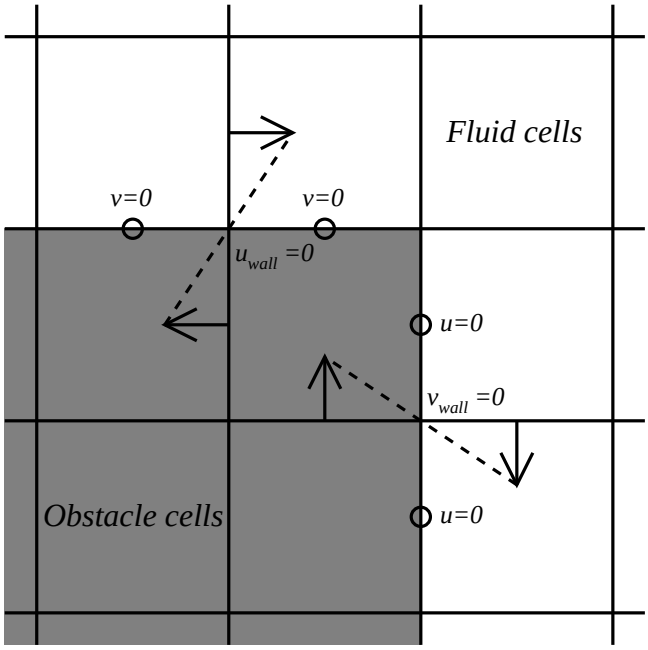


Figure 4.11: The velocity is set to zero on the boundary between obstacle cells and fluid cells.

4.8 Time-step restriction

If Δt is too large, the simulation becomes unstable. The simulation may become non-physical and oscillating at best, or the flotaing point values may become exponentially larger and in the end reach "NaN" and "inf". There are a few criteria that will help us determine a safe Δt value.

The first is the Courant-Friedrichs-Lewy (CFL) condition which states that a fluid particle may not travel farther than one cell during one time step [9, 29]:

$$|\Delta t \cdot u| < \Delta x \tag{4.39}$$

$$|\Delta t \cdot v| < \Delta y \tag{4.40}$$

The viscous term in the momentum equation can be thought of as diffusion of velocity. The viscous term produces a force which evens out sharp changes in velocity. The force causes the velocity to move towards an equilibrium state. However, if the time-step Δt is too large, the velocity will swing past the equilibrium state and farther beyond than in the first place. Thus, the time-step must be restricted based on diffusion. The requirement is [9, 29, 6]:

$$\Delta t < \frac{1}{2\nu_{max}} \frac{\Delta x^2 \Delta y^2}{\Delta x^2 + \Delta y^2} \tag{4.41}$$

where ν is the kinematic viscosity $\frac{\mu}{\rho}$. In two-phase flow, ν can either be $\frac{\mu_L}{\rho_L}$ or $\frac{\mu_D}{\rho_D}$, the kinematic viscosity of the light or dark fluid respectively. I define:

$$\nu_{max} = \max\left(\frac{\mu_L}{\rho_L}, \frac{\mu_D}{\rho_D}\right) \tag{4.42}$$

A third condition is cited in [31, 4], and is a result from approximate stability analysis of the linearised Navier-Stokes equations:

$$\Delta t < \frac{2\nu_{min}}{u^2} \tag{4.43}$$

$$\Delta t < \frac{2\nu_{min}}{v^2} \tag{4.44}$$

where

$$\nu_{min} = \min\left(\frac{\mu_L}{\rho_L}, \frac{\mu_D}{\rho_D}\right) \tag{4.45}$$

A Δt that satisfies all the conditions is thus found by:

$$\Delta t \leftarrow k \cdot \min\left(\frac{\Delta x}{|u|_{max}}, \frac{\Delta y}{|v|_{max}}, \frac{1}{2\nu_{max}}\frac{\Delta x^2 \Delta y^2}{\Delta x^2 + \Delta y^2}, \frac{2\nu_{min}}{|u|^2_{max}}, \frac{2\nu_{min}}{|v|^2_{max}}\right) \tag{4.46}$$

where u_{max} is the maximum horizontal velocity component, v_{max} is the maximal vertical velocity component and k is a safety factor between zero and one. In most cases, it should be safe to let k be close to one.

For two-phase flow, there are additional restrictions covered in chapter 5.

4.9 Algorithm

An algorithm based on the projection method can be sketched:

- Initialise variables such as time, `mask`, `c`, constants and boundary conditions.
- Set initial conditions for $\boldsymbol{u}$ and p
- While running simulation
  - Update time
  - Calculate tentative velocity
  - Impose boundary conditions on $\boldsymbol{u}^*$
  - Solve the PPE with given boundary conditions
  - Update $\boldsymbol{u}$ and p
  - Impose boundary conditions on $\boldsymbol{u}$ and p

4.10 Data structures

Pressure and velocity node values are stored in three 2D-arrays: `p`, `u` and `v`. A 2D-array `c` contains volume-of-fluid information (see section 5). Another 2D-array `mask` contains flags for each cell. The flags are used to define obstacle and fluid cells and boundary conditions for the PPE. For all arrays, the first index is the column number and the second index is the row number. All indexing will start on zero. Let m be width and n the height of the domain in number of cells. The `p`, `mask` and `c` arrays have ghost cell strips all around the domain. Hence, two cell rows and two cell columns come in addition to the cells representing the domain interior. The `v` array needs two additional columns and the `u` array needs two additional rows for ghost cells.

Array	width	height
`p`	$m+2$	$n+2$
`u`	$m+1$	$n+2$
`v`	$m+2$	$n+1$
`mask`	$m+2$	$n+2$
`c`	$m+2$	$n+2$

Table 4.1: Array sizes relative to the domain size. Compare with figure 4.3

Various constants such as density ρ and viscosity μ for two fluids, gravity and surface tension coefficient are stored as scalars or vectors. The density and viscosity fields are calculated from the colour function field `c` and the density and viscosity constants at each time level. Time is stored in `t`. Boundary conditions for the north, south, west and east boundaries are stored in a dictionary or map data structure.

4.10.1 The mask array

The interpretations of the ghost cells and the interior cells in the mask array are different (see figure 4.12).

Interior cells

A 1 in the lowest (0th) bit indicate a fluid cell, while a 0 indicate an obstacle cell. A 1 in the second lowest (1st) bit indicate a fixed pressure value for this cell. If the PPE would would otherwise have infinitely many solutions, one pressure node value is fixed such that there is a unique solution. Of course, the pressure must be fixed for a fluid cell, not an obstacle cell.

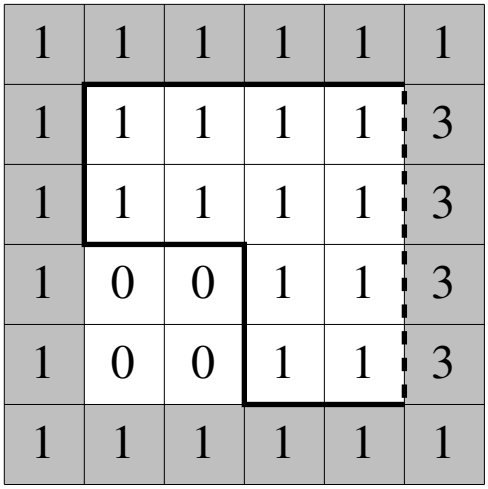


Figure 4.12: The ones in the interior mark fluid cells. The zeros mark obstacles. In the ghost cells, 1 indicates the Neumann condition, and 3 indicates the Dirichlet condition for the PPE. The Neumann condition is also used between fluid and obstacle cells as shown by the thick line.

Ghost cells

When calling the PPE solver, the ghost cell values of `mask` and ϕ upon entry tells the PPE solver what boundary condition to use. Let ϕ_{in} be ϕ as passed to the PPE solver, and ϕ_{out} be ϕ returned from the PPE solver. On the west boundary, for instance, the `mask` and ϕ_{in} values are interpreted as follows:

- `mask` $= 1 : (\phi_{in})_{-1,j} = ((\phi_{out})_{-1,j} - (\phi_{out})_{0,j})$
  This corresponds to a Neumann boundary condition.

- `mask` $= 2 : (\phi_{in})_{-1,j} = ((\phi_{out})_{-1,j})$
  This corresponds to a Diriclet boundary condition where the boundary passes through the cell centres.

- `mask` $= 3 : (\phi_{in})_{-1,j} = ((\phi_{out})_{-1,j} + (\phi_{out})_{0,j})$
  This corresponds to a Dirichlet boundary condition where the boundary passes right between cells.

ϕ_{in} must be set to suitable values each iteration, since the values are overwritten by ϕ_{out} when the PPE solver returns.

Chapter 5

Volume-of-fluid method

The volume-of-fluid method (VOF) is a method for keeping track of two different fluids in two-phase flow simulations. The method tracks the volume of each fluid in each cell. For each time iteration, the interface between the two fluids is reconstructed from the volume data.

A characteristic function χ is defined as 1 in one fluid and 0 in the other. In the figures, I have drawn the first fluid darker than the second, and I will hereafter call the fluids dark fluid and light fluid respectively. I consider the dark fluid to be below the interface, and the light above the interface. A discrete version of χ is called the volume fraction or colour function f and is defined at cell centres. The function value in a cell is in the range $[0, 1]$ and defines how much of the cell is filled with dark fluid. The remaining part of the cell is filled with light fluid (see figure 5.1). Note that "volume" and "area" are used interchangeably in 2D. Mathematically, we have in 2D:

$$f_{i,j} = \frac{1}{\Delta x \Delta y} \int_{y_{j-1/2}}^{y_{j+1/2}} \int_{x_{i-1/2}}^{x_{i+1/2}} \chi(x, y)\ dx\ dy \tag{5.1}$$

where $\Delta x = x_{i+1/2} - x_{i-1/2}$, $\Delta y = y_{j+1/2} - y_{j-1/2}$, and $f_{i,j}$ is the colour function value in the cell which covers the area $[x_{i-1/2}, x_{i+1/2}] \times [y_{j-1/2}, y_{j+1/2}]$.

Either a smoothed colour function or the colour function itself is used to linearly interpolate the density and viscosity fluid properties between the two fluids. The Navier-Stokes equations are solved as usual with the interpolated fluid properties. No boundary conditions need to be applied at the interface [12]. A surface tension term is included in Navier-Stokes equations. Various methods exist for estimating the surface tension term, and two of them are implemented here.

The colour function is advected with the flow. A common problem when solving the advection equation numerically, is non-physical smoothing which is especially noticeable when the function being advected is discontinuous or nearly so. Total variation diminishing methods alleviate the problem, but the smoothing is not removed completely. In the VOF method, the interface

Figure 5.1: Colour function values for an arbitrary interface.

is reconstructed from the colour function before the advection step, and the sharp, reconstructed interface is taken into account to avoid smoothing by using a geometrically based algorithm.

5.1 Density and viscosity smoothing

In most articles I've read about the VOF method, the density ρ and viscosity μ are not smoothed and therefore changes abruptly at the interface[21, 7, 10]. The density and viscosity are linearly interpolated between the two fluids. Let ρ_L and ρ_D be the densities and μ_L and μ_D be the viscosities of the light and dark fluid respectively. The interpolated density and viscosity are:

$$\rho = \rho_L(1 - f) + \rho_D f \tag{5.2}$$
$$\mu = \mu_L(1 - f) + \mu_D f \tag{5.3}$$

Unfortunately, I sometimes experienced non-physical currents tangential to the interface if the time step Δt were too big. For some densities and viscosities, the time step had to be so small that any practical simulation was impossible. I therefore added the option to smooth the density and viscosity slightly:

$$\rho = \rho_L(1 - \tilde{f}) + \rho_D \tilde{f} \tag{5.4}$$
$$\mu = \mu_L(1 - \tilde{f}) + \mu_D \tilde{f} \tag{5.5}$$

where $\tilde{f}$ is a smoothed version of the colour function. The smoothing can be set to on or off in my implementation, where off is the default. Though smoothing of the density and viscosity is generally not described in articles about the VOF method, the density and/or viscosity often varies smoothly

from one fluid to the other in level-set (LS) and coupled level-set and volume-of-fluid (CLSVOF) methods[19, 32, 25]. [17] is one article on VOF where smoothing is actually used.

I smooth the colour function with a 3×3 kernel in 2D. This was sufficient to stabilise the simulations I ran. I tested the smoothing kernel used for curvature estimation in [18]:

$$\boldsymbol{K} = \begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix} \tag{5.6}$$

However, I found that the smoothing effect was a bit weak. I therefore chose to use a smaller weight for the central node. This kernel also gave more accurate results for curvature estimation (see section 5.5.1):

$$\boldsymbol{K} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{5.7}$$

This kernel can easily be extended to 3D:

$$\boldsymbol{K} = \left[\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right] \tag{5.8}$$

A smoothed colour function value $\tilde{f}_{i,j}$ is a weighted average of neighbouring cells.

$$\tilde{f}_{i,j} = \frac{\sum_{k=1}^{3} \sum_{l=1}^{3} K_{k,l} \cdot f_{i+k-2,j+l-2}}{\sum_{k=1}^{3} \sum_{l=1}^{3} K_{k,l}} \tag{5.9}$$

If any of the colour function values $f_{i+k-2,j+l-2}$ falls outside the fluid domain, I set the corresponding element in the kernel matrix to zero such that the outside colour function value will not contribute to the average. For instance, next to a vertical wall, the kernel may be:

$$\boldsymbol{K} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 4 & 2 \\ 0 & 2 & 1 \end{bmatrix} \tag{5.10}$$

5.2 Boundary conditions

Like the pressure and velocity, the colour function field f required boundary conditions.

5.2.1 Contact angle

A common boundary condition for two-phase flow is the contact angle between the interface and the wall. On the boundary, the normal $\boldsymbol{n}$ of the interface is[2]:

$$\boldsymbol{n} = \boldsymbol{n}_{wall} \cos\theta + \boldsymbol{n}_t \sin\theta \tag{5.11}$$

where $\boldsymbol{n}_{wall}$ is the boundary normal, $\boldsymbol{n}_t$ is a tangent to the boundary and normal to the contact line between the interface and the boundary, and θ is the contact angle. The contact angle depends on the fluid and the boundary material, geometry and smoothness, and motion. I have taken the easy way out and assumed that $\theta = 90°$, thus $\boldsymbol{n} = \boldsymbol{n}_t$.

It is difficult to find articles or books where the implementation of boundary conditions is described. Seeing that $\theta = 90°$ resembles the Neumann boundary condition[19], I chose to implement the boundary condition as described in [9] for pressure; copy colour function values f from fluid cells to neighbouring ghost cells. If a ghost cell has more than one neighbouring fluid cell, use the average.

5.2.2 Inlets

In two-phase flows, one needs to know which fluid is flowing into the domain at inlets (Dirichlet boundary condition). I chose to set the colour function values f in ghost cells, which borders an inlet, to either zero or one indicating the fluid flowing from this inlet.

The observant reader may have noticed that both the contact angle boundary condition and inlet boundary condition use the ghost cells, and the colour function cannot have two different values in the ghost cells to satisfy both conditions. However, contact angles are only needed at walls, not inlets. I therefore use the Neumann boundary condition as default, and then apply a Dirichlet boundary condition at inlets to overwrite the ghost cell values.

5.3 Piecewise linear interface construction

Piecewise linear interface construction (PLIC) is a technique where the interface is approximated with a straight line or plane within each cell containing the interface. The reconstructed interface is usually discontinuous at the cell boundaries (see figure 5.2) Cells with a colour function value of 0 or 1 contain only one type of fluid and do not contain the interface. Such cells are therefore ignored at this stage of the algorithm. There are a number of different algorithms for PLIC. Pilliod and Puckett [13] compares some of them.

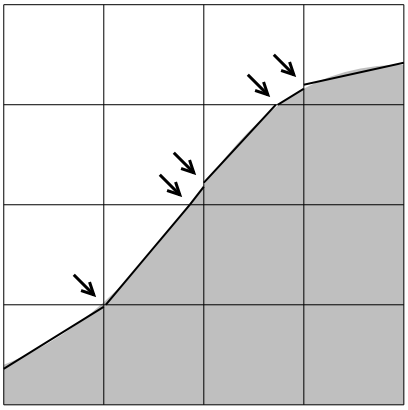


Figure 5.2: Piecewise linear interface construction. Discontinuities are marked with arrows.

5.3.1 Representing a reconstructed interface

Since the reconstructed interface is a straight line or a plane within each cell, it can be represented by the line equation or plane equation:

$$ax + by + c = 0 \tag{5.12}$$

$$ax + by + cz + d = 0 \tag{5.13}$$

Thus three coefficients per cell are needed for a line and four for a plane[1]. In 2D, $[a, b]^T$ is the line's normalised normal, pointing away from the dark fluid, and c is the signed distance from some point $\boldsymbol{p}$ to the line (see figure 5.3). In 3D, $[a, b, c]^T$ is the plane's normalised normal, and d is the signed distance from some point $\boldsymbol{p}$ to the plane. I define the point $\boldsymbol{p}$ to be the centre of the cell for the sake of symmetry. The signed distance is the positive distance if the cell centre is in the light fluid and negative if in the dark fluid.

Hereafter, I will use the symbols $\boldsymbol{n}$ for the line or plane normal, and d for the signed distance, that is, the line and plane equations become (2D and 3D):

$$n_x x + n_y y + d = 0 \tag{5.14}$$

$$n_x x + n_y y + n_z z + d = 0 \tag{5.15}$$

Sometimes, it is necessary to reposition the point $\boldsymbol{p}$ without moving the interface line or plane. For instance, assume that $\boldsymbol{n}$ and d_1 represents an interface line or plane in one cell, and that we wish to use the same line

[1] Actually, only two coefficients are needed for a line and three for a plane, for instance by representing the normal with an angle in 2D and latitude and longitude in 3D, but using the line and plane equations is easier.

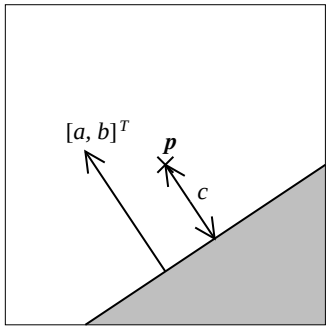


Figure 5.3: The linear, reconstructed interface can be represented by three scalars a, b, c for each cell.

or plane in a neighbouring cell. The normal is of course the same in both cells, but the signed distances are generally different. Let $\boldsymbol{p}_1$ be the centre of the first cell, and d_1 be the known signed distance from $\boldsymbol{p}_1$ to the interface plane. Let $\boldsymbol{p}_2$ be the centre of the second cell and d_2 be the unknown signed distance from $\boldsymbol{p}_2$ to the line or plane. d_2 is then (2D and 3D):

$$d_2 = n_x(p_{2x} - p_{1x}) + n_y(p_{2y} - p_{1y}) + d_1 \tag{5.16}$$

$$d_2 = n_x(p_{2x} - p_{1x}) + n_y(p_{2y} - p_{1y}) + n_z(p_{2z} - p_{1z}) + d_1 \tag{5.17}$$

$\boldsymbol{n}$ and d_2 now represent the same interface line or plane in the second cell.

5.3.2 Calculating the volume fraction

Calculating the volume fraction for a given interface line or plane is an important building block in the PLIC method, both for reconstruction and advection. To find the volume fraction, the volume within the cell below the line or plane is divided by the cell's total volume.

How to find the volume below a line or plane is well described in [10]. Formulae for the volume fraction below a plane are also given in [17]. However, I think the algorithm I describe here is slightly simpler to implement, though it is based on the same idea as the algorithm in the article.

First, to simplify the calculations, all the normal components are made non-negative. This is done simply by taking the absolute value of each of the normal components. Because of symmetry, where the cell centre is treated as the origin, this operation only changes the orientation of the line or plane, not the volume fraction or the signed distance from the cell centre to the line or plane.

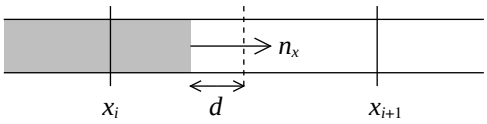


Figure 5.4: n_x and d define an interface in 1D.

It is easiest to begin in 1D and extend the algorithm to 2D and 3D afterwards.

1D: The i-th cell is $[x_{i-1/2}, x_{i+1/2}]$. The interface is represented by a sign n_x, which is -1 or 1, and a signed distance d from the cell centre $(x_{i-1/2} + x_{i+1/2})/2$ to the interface (see figure 5.4). Since all normal components are made non-negative, only $n_x = 1$ needs to be considered. Let f_i be the volume fraction for the i-th cell.

Let the hat symbol $\hat{}$ denote coordinates relative to the cell centre $(x_{i-1/2}+ x_{i+1/2})/2$:

$$\hat{x}_{i-1/2} = x_{i-1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = \frac{1}{2}(x_{i-1/2} - x_{i+1/2}) = -\frac{\Delta x}{2} \tag{5.18}$$

$$\hat{x}_{i+1/2} = x_{i+1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = \frac{1}{2}(x_{i+1/2} - x_{i-1/2}) = \frac{\Delta x}{2} \tag{5.19}$$

The position of the interface $\hat{x}_{interface}$ relative to the cell centre is simply $-d$. The amount of the dark fluid within the cell is:

$$f_i \Delta x = \begin{cases} 0 & : \ \hat{x}_{interface} < \hat{x}_{i-1/2} \\ \hat{x}_{interface} - \hat{x}_{i-1/2} & : \ \hat{x}_{i-1/2} \le \hat{x}_{interface} \le \hat{x}_{i+1/2} \\ \hat{x}_{i+1/2} - \hat{x}_{i-1/2} & : \ \hat{x}_{i+1/2} < \hat{x}_{interface} \end{cases} \tag{5.20}$$

However, this is not easily extended to 2D and 3D. Instead, let $L(\hat{x}; d)$ be a function which returns the amount of dark fluid above $\hat{x}$ given the interface:

$$L(\hat{x}; d) = \begin{cases} 0 & : \ \hat{x}_{interface} < \hat{x} \\ \hat{x}_{interface} - \hat{x} & : \ \hat{x}_{interface} \ge \hat{x} \end{cases} \tag{5.21}$$

where $\hat{x}_{interface} = -d$. I will hereafter only write $L(\hat{x})$ instead of $L(\hat{x}; d)$ when it is clear which interface is used.

The amount of dark fluid inside the cell can be found by:

$$f_i \Delta x = L(\hat{x}_{i-1/2}) - L(\hat{x}_{i+1/2}) \tag{5.22}$$

First, calculate the amount of dark fluid above $x_{i-1/2}$, then subtract the amount which overshoots $x_{i+1/2}$. This is easily extended to 2D and 3D if one can find functions equivalent to $L(\hat{x})$.

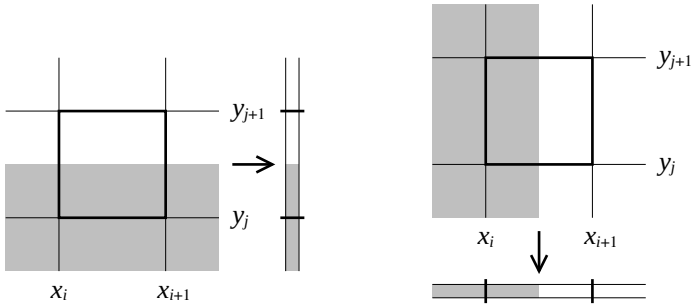


Figure 5.5: If the line is horizontal or vertical, the 2D problem is reduced to 1D.

2D: Let $A(\hat{x}, \hat{y}; n_x, n_y, d)$ be the 2D-equivalent of $L(\hat{x}; d)$. $A(\hat{x}, \hat{y}; n_x, n_y, d)$ returns the area to the right of $\hat{x}$, above $\hat{y}$ and below the interface line with the coefficients n_x, n_y, d, where $\hat{x}$ and $\hat{y}$ are relative to the cell centre. More precisely:

$$\begin{aligned} A(\hat{x}, \hat{y}; n_x, n_y, d) &= \int_{\hat{x}}^{\infty} \int_{\hat{y}}^{\infty} \tilde{\chi}(x, y; n_x, n_y, d) dy\ dx \\ &= \lim_{X \to \infty} \int_{\hat{x}}^{X} \lim_{Y \to \infty} \int_{\hat{y}}^{Y} \tilde{\chi}(x, y n_x, n_y, d) dy\ dx \end{aligned} \tag{5.23}$$

where

$$\tilde{\chi}(x, y; n_x, n_y, d) = \begin{cases} 1 & : \ n_x x + n_y y + d \leq 0 \\ 0 & : \ n_x x + n_y y + d > 0 \end{cases} \tag{5.24}$$

However, $A(\hat{x}, \hat{y}; n_x, n_y, d)$ is only defined if the limits exist. The limits exist if the normal components are all strictly positive. If n_x or n_y is zero, that is, the line is horizontal or vertical, $A(\hat{x}, \hat{y}; n_x, n_y, d)$ may go towards infinity. Fortunately, when n_x or n_y is zero, the problem can be reduced to 1D (see figure 5.5).

I will hereafter only write $A(\hat{x}, \hat{y})$ instead of $A(\hat{x}, \hat{y}; n_x, n_y, d)$ when it is clear which interface is used. Let the i, j-th cell be $[x_{i-1/2}, x_{i+1/2}] \times [y_{j-1/2}, y_{j+1/2}]$. Let $\hat{x}_{i-1/2}$, $\hat{x}_{i+1/2}$, $\hat{y}_{j-1/2}$ and $\hat{y}_{j+1/2}$ be the cell coordinates relative to the cell centre:

$$\hat{x}_{i-1/2} = x_{i-1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = -\frac{\Delta x}{2} \tag{5.25}$$

$$\hat{x}_{i+1/2} = x_{i+1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = \frac{\Delta x}{2} \tag{5.26}$$

$$\hat{y}_{j-1/2} = y_{j-1/2} - \frac{1}{2}(y_{j-1/2} + y_{j+1/2}) = -\frac{\Delta y}{2} \tag{5.27}$$

$$\hat{y}_{j+1/2} = y_{j+1/2} - \frac{1}{2}(y_{j-1/2} + y_{j+1/2}) = \frac{\Delta y}{2} \tag{5.28}$$

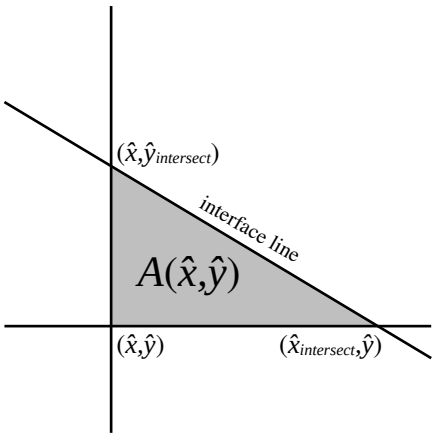


Figure 5.6: $A(\hat{x}, \hat{y})$ returns the area of the triangle below the interface line, to the right of $\hat{x}$ above $\hat{y}$.

If $n_x = 0$, the line is horizontal, and the volume fraction is:

$$f_{i,j} = \frac{L(\hat{y}_{j-1/2}; d) - L(\hat{y}_{j+1/2}; d)}{\Delta y} \tag{5.29}$$

If $n_y = 0$, the line is vertical, and the volume fraction is:

$$f_{i,j} = \frac{L(\hat{x}_{i-1/2}; d) - L(\hat{x}_{i+1/2}; d)}{\Delta x} \tag{5.30}$$

If all the normal components are strictly positive, $A(\hat{x}, \hat{y})$ becomes the area of the triangle shown in figure 5.6.

If the point $(\hat{x}, \hat{y})$ is above the interface line, that is, $n_x\hat{x} + n_y\hat{y} + d > 0$, the triangle area is zero. Otherwise, $\hat{x}_{intersect}$ and $\hat{y}_{intersect}$ must be found to calculate the area. Insert the known and unknown values into the line equation:

$$n_x\hat{x}_{intersect} + n_y\hat{y} + d = 0 \tag{5.31}$$

$$n_x\hat{x} + n_y\hat{y}_{intersect} + d = 0 \tag{5.32}$$

Solve for $\hat{x}_{intersect}$ and $\hat{y}_{intersect}$:

$$\hat{x}_{intersect} = -\frac{n_y\hat{y} + d}{n_x} \tag{5.33}$$

$$\hat{y}_{intersect} = -\frac{n_x\hat{x} + d}{n_y} \tag{5.34}$$

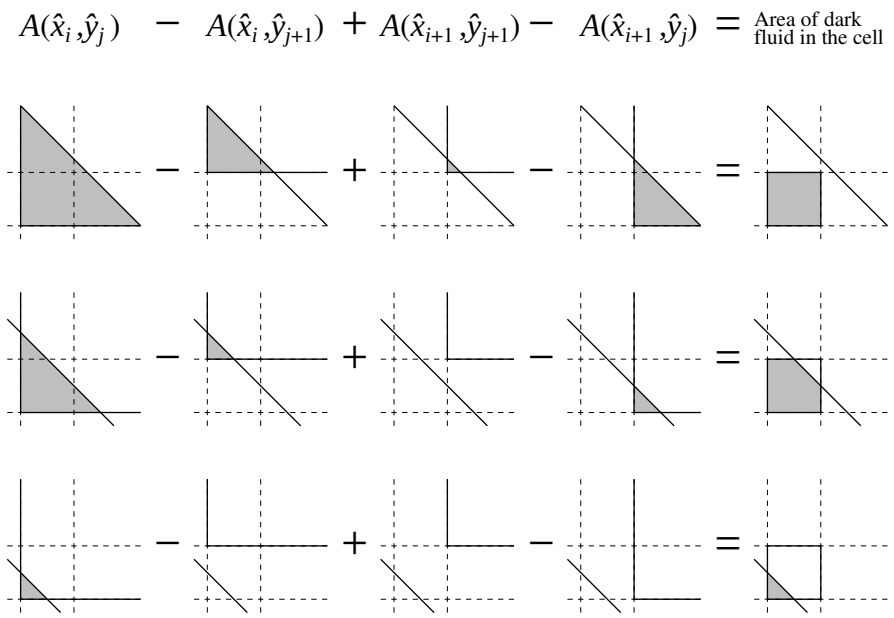


Figure 5.7: By adding and subtracting the right areas, the area of dark fluid inside the cell is found. Three examples with different interface lines are shown.

Calculate the triangle area:

$$\begin{aligned}\frac{1}{2}(\hat{x}_{intersect} - \hat{x})(\hat{y}_{intersect} - \hat{y}) &= \frac{1}{2}\left(\hat{x} + \frac{n_y\hat{y} + d}{n_x}\right)\left(\hat{y} + \frac{n_x\hat{x} + d}{n_y}\right) \\ &= \frac{(n_x\hat{x} + n_y\hat{y} + d)^2}{2n_xn_y} \end{aligned} \tag{5.35}$$

$A(\hat{x}, \hat{y}; n_x, n_y, d)$ is therefore:

$$A(\hat{x}, \hat{y}; n_x, n_y, d) = \begin{cases} \frac{(n_x\hat{x}+n_y\hat{y}+d)^2}{2n_xn_y} & : \; n_x\hat{x} + n_y\hat{y} + d \leq 0 \\ 0 & : \; n_x\hat{x} + n_y\hat{y} + d > 0 \end{cases} \tag{5.36}$$

Let $f_{i,j}$ be the volume fraction for the i, j-th cell. The area of dark fluid inside the cell is (see figure 5.7):

$$\begin{aligned} f_{i,j}\Delta x\Delta y = A(\hat{x}_{i-1/2}, \hat{y}_{j-1/2}) - A(\hat{x}_{i+1/2}, \hat{y}_{j-1/2}) \\ -A(\hat{x}_{i-1/2}, \hat{y}_{j+1/2}) + A(\hat{x}_{i+1/2}, \hat{y}_{j+1/2}) \end{aligned} \tag{5.37}$$

If the interface line is almost horizontal or vertical, the calculations can become unacceptably inaccurate. It might be better to treat such lines as

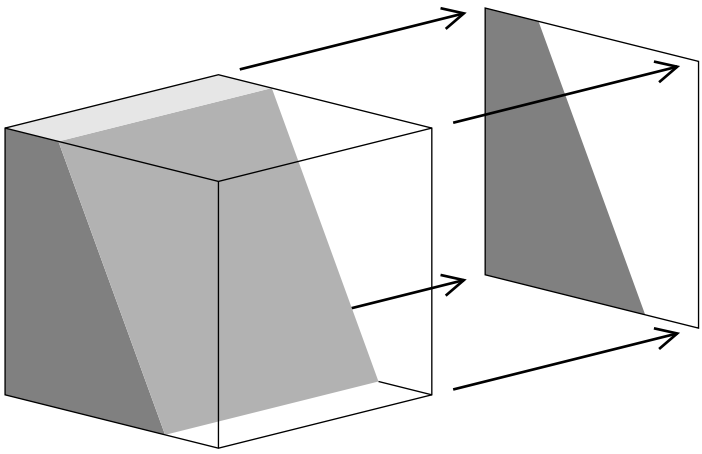


Figure 5.8: If the plane is parallel to one of the main axes, the 3D problem is reduced to 2D.

completely horizontal or vertical. If one of the normal components is below some small value ϵ, set the component to exact zero and reduce the problem to 1D.

3D: Let $V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d)$ be the 3D-equivalent of $A(\hat{x}, \hat{y}; n_x, n_y, d)$ and $L(\hat{x}; d)$. Assume that the x-axis points to the right, the y-axis into the paper and the z-axis up. $V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d)$ returns the volume to the right of $\hat{x}$, farther away than $\hat{y}$, above $\hat{z}$ and below the interface plane. More precisely:

$$\begin{aligned} V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d) &= \int_{\hat{x}}^{\infty} \int_{\hat{y}}^{\infty} \int_{\hat{z}}^{\infty} \tilde{\chi}(x, y, z; n_x, n_y, n_z, d) dz\ dy\ dx \\ &= \lim_{X \to \infty} \int_{\hat{x}}^{X} \lim_{Y \to \infty} \int_{\hat{y}}^{Y} \lim_{Z \to \infty} \int_{\hat{z}}^{Z} \tilde{\chi}(x, y, z; n_x, n_y, n_z, d) dz\ dy\ dx \end{aligned} \tag{5.38}$$

where

$$\tilde{\chi}(x, y, z; n_x, n_y, n_z, d) = \begin{cases} 1 & : \ n_x x + n_y y + n_z z + d \leq 0 \\ 0 & : \ n_x x + n_y y + n_z z + d > 0 \end{cases} \tag{5.39}$$

$V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d)$ is only defined if the limits exist. If n_x, n_y or n_z is zero, that is, the plane is parallel to the x-, y- or z-axis, the problem is reduced to 2D (see figure 5.8).

I will hereafter only write $V(\hat{x}, \hat{y}, \hat{z})$ instead of $V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d)$ when it is clear which interface is used.

Let the i,j,k-th cell be $[x_{i-1/2}, x_{i+1/2}] \times [y_{j-1/2}, y_{j+1/2}] \times [z_{k-1/2}, z_{k+1/2}]$. Let $f_{i,j,k}$ be the volume fraction for the i,j,k-th cell. Let $\hat{x}_{i-1/2}$, $\hat{x}_{i+1/2}$, $\hat{y}_{j-1/2}$, $\hat{y}_{j+1/2}$, $\hat{z}_{k-1/2}$ and $\hat{z}_{k+1/2}$ be the cell coordinates relative to the cell centre:

$$\hat{x}_{i-1/2} = x_{i-1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = -\frac{\Delta x}{2} \tag{5.40}$$

$$\hat{x}_{i+1/2} = x_{i+1/2} - \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) = \frac{\Delta x}{2} \tag{5.41}$$

$$\hat{y}_{j-1/2} = y_{j-1/2} - \frac{1}{2}(y_{j-1/2} + y_{j+1/2}) = -\frac{\Delta y}{2} \tag{5.42}$$

$$\hat{y}_{j+1/2} = y_{j+1/2} - \frac{1}{2}(y_{j-1/2} + y_{j+1/2}) = \frac{\Delta y}{2} \tag{5.43}$$

$$\hat{z}_{k-1/2} = z_{k-1/2} - \frac{1}{2}(z_{k-1/2} + z_{k+1/2}) = -\frac{\Delta z}{2} \tag{5.44}$$

$$\hat{z}_{k+1/2} = z_{k+1/2} - \frac{1}{2}(z_{k-1/2} + z_{k+1/2}) = \frac{\Delta z}{2} \tag{5.45}$$

If only $n_x = 0$, the volume fraction is:

$$\begin{aligned} f_{i,j,k} = &\frac{A(\hat{y}_{j-1/2}, \hat{z}_{k-1/2}; n_y, n_z, d) - A(\hat{y}_{j+1/2}, \hat{z}_{k-1/2}; n_y, n_z, d)}{\Delta y \Delta x} \\ &+ \frac{A(\hat{y}_{j+1/2}, \hat{z}_{k+1/2}; n_y, n_z, d) - A(\hat{y}_{j-1/2}, \hat{z}_{k+1/2}; n_y, n_z, d)}{\Delta y \Delta x} \end{aligned} \tag{5.46}$$

If only $n_y = 0$, the volume fraction is:

$$\begin{aligned} f_{i,j,k} = &\frac{A(\hat{x}_{i-1/2}, \hat{z}_{k-1/2}; n_x, n_z, d) - A(\hat{x}_{i+1/2}, \hat{z}_{k-1/2}; n_x, n_z, d)}{\Delta x \Delta z} \\ &+ \frac{A(\hat{x}_{i+1/2}, \hat{z}_{k+1/2}; n_x, n_z, d) - A(\hat{x}_{i-1/2}, \hat{z}_{k+1/2}; n_x, n_z, d)}{\Delta x \Delta z} \end{aligned} \tag{5.47}$$

If only $n_z = 0$, the volume fraction is:

$$\begin{aligned} f_{i,j,k} = &\frac{A(\hat{x}_{i-1/2}, \hat{y}_{j-1/2}; n_x, n_y, d) - A(\hat{x}_{i+1/2}, \hat{y}_{j-1/2}; n_x, n_y, d)}{\Delta x \Delta y} \\ &+ \frac{A(\hat{x}_{i+1/2}, \hat{y}_{j+1/2}; n_x, n_y, d) - A(\hat{x}_{i-1/2}, \hat{y}_{j+1/2}; n_x, n_y, d)}{\Delta x \Delta y} \end{aligned} \tag{5.48}$$

If both $n_x = n_y = 0$, the problem is reduced to 1D:

$$f_{i,j,k} = \frac{L(\hat{z}_{k-1/2}; d) - L(\hat{z}_{k+1/2}; d)}{\Delta z} \tag{5.49}$$

etc.

If all the normal components are strictly positive, $V(\hat{x}, \hat{y}, \hat{z})$ becomes the volume of the tetrahedron shown in figure 5.9.

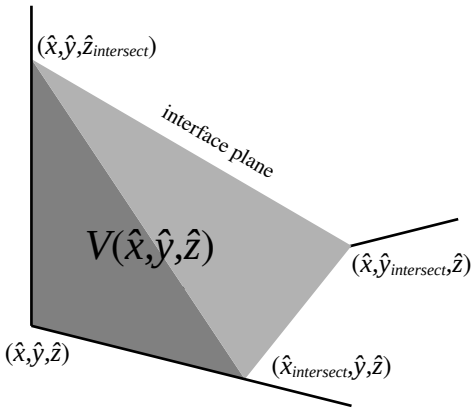


Figure 5.9: $V(\hat{x},\hat{y},\hat{z})$ returns the volume of the tetrahedron below the interface plane, to the right of $\hat{x}$, farther away than $\hat{y}$ and above $\hat{z}$.

If the point $(\hat{x},\hat{y},\hat{z})$ is above the interface plane, that is, $n_x\hat{x}+n_y\hat{y}+n_z\hat{z}+d>0$, the tetrahedron volume is zero. Otherwise, $\hat{x}_{intersect}$, $\hat{y}_{intersect}$ and $\hat{z}_{intersect}$ must be found to calculate the volume. Insert the known and unknown values into the plane equation:

$$n_x\hat{x}_{intersect}+n_y\hat{y}+n_z\hat{z}+d=0 \tag{5.50}$$
$$n_x\hat{x}+n_y\hat{y}_{intersect}+n_z\hat{z}+d=0 \tag{5.51}$$
$$n_x\hat{x}+n_y\hat{y}+n_z\hat{z}_{intersect}+d=0 \tag{5.52}$$

Solve for $\hat{x}_{intersect}$, $\hat{y}_{intersect}$ and $\hat{z}_{intersect}$:

$$\hat{x}_{intersect}=-\frac{n_y\hat{y}+n_z\hat{z}+d}{n_x} \tag{5.53}$$
$$\hat{y}_{intersect}=-\frac{n_x\hat{x}+n_z\hat{z}+d}{n_y} \tag{5.54}$$
$$\hat{z}_{intersect}=-\frac{n_x\hat{x}+n_y\hat{y}+d}{n_z} \tag{5.55}$$

Calculate the tetrahedron volume:

$$\frac{1}{6}(\hat{x}_{intersect}-\hat{x})(\hat{y}_{intersect}-\hat{y})(\hat{z}_{intersect}-\hat{z})=-\frac{(n_x\hat{x}+n_y\hat{y}+n_z\hat{z}+d)^3}{6n_xn_yn_z} \tag{5.56}$$

$V(\hat{x},\hat{y},\hat{z};n_x,n_y,n_z,d)$ is:

$$V(\hat{x},\hat{y},\hat{z})=\begin{cases} -\frac{(n_x\hat{x}+n_y\hat{y}+n_z\hat{z}+d)^3}{6n_xn_yn_z} & : \; n_x\hat{x}+n_y\hat{y}+n_z\hat{z}+d\le 0 \\ 0 & : \; n_x\hat{x}+n_y\hat{y}+n_z\hat{z}+d>0 \end{cases} \tag{5.57}$$

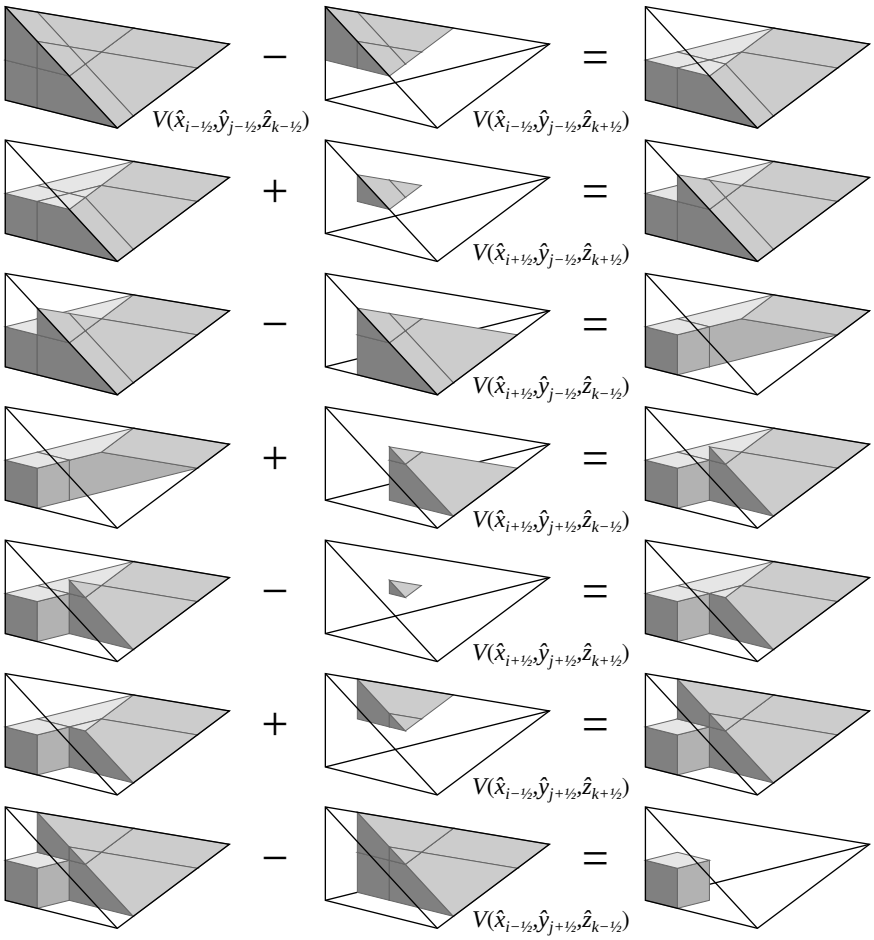


Figure 5.10: By adding and subtracting the right volumes, the volume of dark fluid inside the cell is found.

The volume of dark fluid inside the cell is (see figure 5.10):

$$\begin{aligned} f_{i,j,k}\Delta x\Delta y\Delta z \quad &= V(\hat{x}_{i-1/2}, \hat{y}_{j-1/2}, \hat{z}_{k-1/2}) - V(\hat{x}_{i+1/2}, \hat{y}_{j-1/2}, \hat{z}_{k-1/2}) \\ &-V(\hat{x}_{i-1/2}, \hat{y}_{j+1/2}, \hat{z}_{k-1/2}) + V(\hat{x}_{i+1/2}, \hat{y}_{j+1/2}, \hat{z}_{k-1/2}) \\ &-V(\hat{x}_{i-1/2}, \hat{y}_{j-1/2}, \hat{z}_{k+1/2}) + V(\hat{x}_{i+1/2}, \hat{y}_{j-1/2}, \hat{z}_{k+1/2}) \\ &+V(\hat{x}_{i-1/2}, \hat{y}_{j+1/2}, \hat{z}_{k+1/2}) - V(\hat{x}_{i+1/2}, \hat{y}_{j+1/2}, \hat{z}_{k+1/2}) \end{aligned} \tag{5.58}$$

Like in 2D, the calculations can become inaccurate when one of the normal components is close to zero. If one of the normal components is below some small value ϵ, set the component to exact zero and reduce the problem to 2D.

5.3.3 Reconstruction when the normal is known

The difference between PLIC reconstruction algorithms is how the interface normal is approximated. Normal estimation will be covered in the next section, but for now, assume that the normal is known. The reconstructed linear interface must be adjusted such that the volume fraction below the line or plane in a cell is equal to the cell's colour function value. Since the normal $\boldsymbol{n}$ is fixed, only d needs to be found.

2D: Let $f_{i,j}$ be the colour function value for the i,j-th cell. The volume fraction in a cell can be expressed as a function $\hat{f}$ of the interface's signed distance d from the cell centre. One has to solve the following equation with respect to d:

$$\hat{f}(d) = f_{i,j} \tag{5.59}$$

Because $\hat{f}(d)$ is a linear combination of piecewise second degree polynomial functions $A(\hat{x}, \hat{y}; n_x, n_y, d)$, $\hat{f}(d)$ itself is a piecewise second degree polynomial. Since $\hat{f}(d)$ calculates an area, it can be thought of as an integral of a function $\hat{f}'(d)$. Because $\hat{f}(d)$ is of second degree, $\hat{f}'(d)$ is a piecewise linear function as shown in figure 5.11. It is not too difficult to see that $\hat{f}(d)$ is monotonically decreasing. The polynomials are joined at d_1, d_2, d_3 and d_4 where the interface line intersects the cell corners, as shown in the figure. Note that d increases to the left in the figure. Because of symmetry, $d_1 = -d_4$, $d_2 = -d_3$ and $\hat{f}(d) = 1 - \hat{f}(-d)$.

The four d-values are found by inserting the cell's corner coordinates into the line equation and solving for d. Let $(\hat{x}, \hat{y})$ be one of the corners relative to the cell's centre:

$$d = -(n_x\hat{x} + n_y\hat{y}) \tag{5.60}$$

Sort the four d-values in descending order. The function $\hat{f}$ can be split into five polynomials, one for each interval $(-\infty, d_4]$, $[d_4, d_3]$, $[d_3, d_2]$, $[d_2, d_1]$ and $[d_1, \infty)$. Call the functions $\hat{f}_5$ to $\hat{f}_1$ respectively. By using the fact that $\hat{f}$ and its derivative are continuous, one gets the following information from which the five functions can be constructed:

- $\hat{f}_1(d_1) = \hat{f}_2(d_1) = 0$
- $\hat{f}_1'(d_1) = \hat{f}_2'(d_1)$
- $\hat{f}_2(d_2) = \hat{f}_3(d_2)$
- $\hat{f}_2'(d_2) = \hat{f}_3'(d_2)$
- $\hat{f}_3(d_3) = \hat{f}_4(d_3)$
- $\hat{f}_3'(d_3) = \hat{f}_4'(d_3)$
- $\hat{f}_4(d_4) = \hat{f}_5(d_4) = 1$

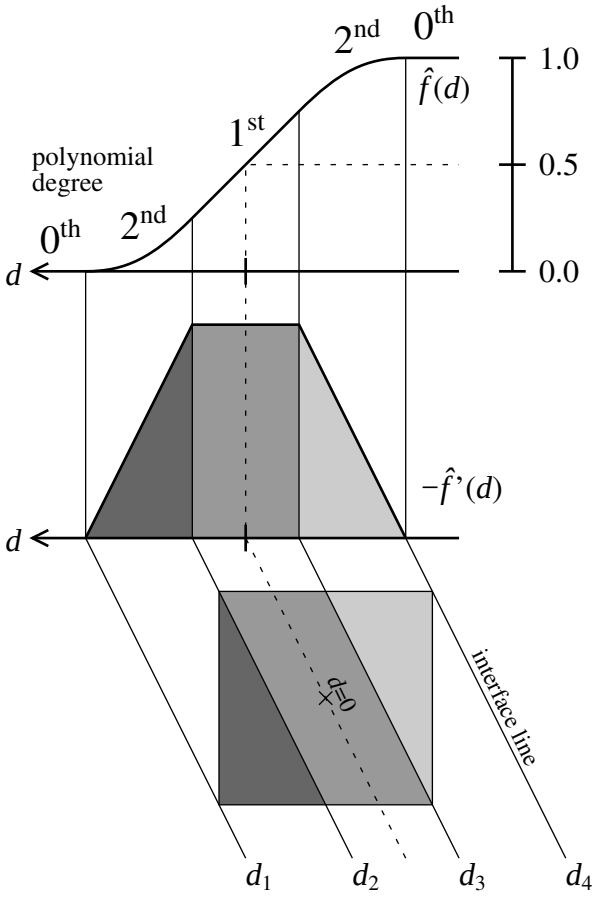


Figure 5.11: $\hat{f}(d)$ is a piecewise second degree polynomial.

- $\hat{f}_4'(d_4) = \hat{f}_5'(d_4)$
- $\hat{f}_1$ and $\hat{f}_5$ are constant.
- $\hat{f}_2$ and $\hat{f}_4$ are second degree polynomials.
- $\hat{f}_3$ is linear.

The functions can be constructed using clever observations, or by using a general method that I will explain here. Let:

- $\hat{f}_1(d) = a_1$
- $\hat{f}_2(d) = a_2 d^2 + b_2 d + c_2$
- $\hat{f}_3(d) = a_3 d + b_3$
- $\hat{f}_4(d) = a_4 d^2 + b_4 d + c_4$
- $\hat{f}_5(d) = a_5$

There are ten unknown coefficients, a, b and c with indices. Differentiate the functions:

- $\hat{f}_1'(d) = 0$
- $\hat{f}_2'(d) = 2a_2 d + b_2$
- $\hat{f}_3'(d) = a_3$
- $\hat{f}_4'(d) = 2a_4 d + b_4$
- $\hat{f}_5'(d) = 0$

Use the previous information to set up a linear equation system:

- $\hat{f}_1(d_1) = \hat{f}_2(d_1) = 0 \Rightarrow a_1 = 0 \wedge a_2 d_1^2 + b_2 d_1 + c_2 = 0$
- $\hat{f}_1'(d_1) = \hat{f}_2'(d_1) \Rightarrow 0 = 2a_2 d_1 + b_2$
- $\hat{f}_2(d_2) = \hat{f}_3(d_2) \Rightarrow a_2 d_2^2 + b_2 d_2 + c_2 = a_3 d_2 + b_3$
- $\hat{f}_2'(d_2) = \hat{f}_3'(d_2) \Rightarrow 2a_2 d_2 + b_2 = a_3$
- $\hat{f}_3(d_3) = \hat{f}_4(d_3) \Rightarrow a_3 d_3 + b_3 = a_4 d_3^2 + b_4 d_3 + c_4$
- $\hat{f}_3'(d_3) = \hat{f}_4'(d_3) \Rightarrow a_3 = 2a_4 d_3 + b_4$
- $\hat{f}_4(d_4) = \hat{f}_5(d_4) = 1 \Rightarrow a_4 d_4^2 + b_4 d_4 + c_4 = 1 \wedge a_5 = 1$
- $\hat{f}_4'(d_4) = \hat{f}_5'(d_4) \Rightarrow 2a_4 d_4 + b_4 = 0$

By solving the system and replacing d_4 with $-d_1$ and d_3 with $-d_2$, the function $\hat{f}$ can be expressed as [32]:

$$\hat{f}(d) = \begin{cases} \hat{f}_1(d) = 0 & d \geq d_1 \\ \hat{f}_2(d) = \frac{(d-d_1)^2}{2(d_1^2-d_2^2)} & d_1 > d > d_2 \\ \hat{f}_3(d) = \frac{1}{2} - \frac{d}{d_1+d_2} & d_2 \geq d \geq d_3 \\ \hat{f}_4(d) = 1 - \frac{(d+d_1)^2}{2(d_1^2-d_2^2)} & d_3 > d > d_4 \\ \hat{f}_5(d) = 1 & d_4 \geq d \end{cases} \tag{5.61}$$

Note that the inequality signs are chosen such that division by zero should not occur, even if $d_1 = d_2$.

Compare $f_{i,j}$ with $\hat{f}(d_1) = 0$, $\hat{f}(d_2)$, $\hat{f}(d_3)$ and $\hat{f}(d_4) = 1$ to find out in which interval the solution lies. One of the following three cases apply:

- $0 \leq f_{i,j} < \hat{f}(d_2)$: In this interval, $\hat{f}(d) = \hat{f}_2(d) = \frac{(d-d_1)^2}{2(d_1^2-d_2^2)}$. Solve for d:

$$d = d_1 \pm \sqrt{2(d_1^2 - d_2^2) f_{i,j}} \tag{5.62}$$

  Since d_1 is the highest possible value for d (any higher, and the interface does not intersect the cell at all), $\pm$ must be turned into $-$.

- $\hat{f}(d_3) < f_{i,j} \leq 1$: In this interval, the $\hat{f}(d) = \hat{f}_4(d) = 1 - \frac{(d+d_1)^2}{2(d_1^2-d_2^2)}$. Solve for d:

$$d = -d_1 \pm \sqrt{2(d_1^2 - d_2^2)(1 - f_{i,j})} \tag{5.63}$$

  Since $d_4 = -d_1$ is the lowest possible value for d, $\pm$ must be turned into $+$.

- $\hat{f}(d_2) \leq f_{i,j} \leq \hat{f}(d_3)$: In this interval, $\hat{f}(d) = \hat{f}_3(d) = \frac{1}{2} - \frac{d}{d_1+d_2}$. Solve for d:

$$d = (d_1 + d_2)\left(\frac{1}{2} - f_{i,j}\right) \tag{5.64}$$

A numerical method could also be used to solve the equation $\hat{f}(d) = f_{i,j}$, for instance the bisection method is guaranteed to succeed, but converges slowly. Rider and Kothe [24] suggest using Brent's method.

3D Let $f_{i,j,k}$ be the colour function value for the i, j, k-th cell. The volume fraction in a cell can be expressed as a function $\hat{f}$ of the interface's signed distance d from the cell centre. One has to solve the following equation with respect to d:

$$\hat{f}(d) = f_{i,j,k} \tag{5.65}$$

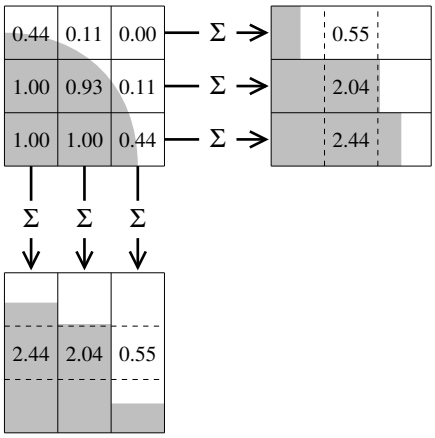


Figure 5.12: Histograms in the ELVIRA algorithm.

Because $\hat{f}(d)$ is a linear combination of piecewise third degree polynomial functions $V(\hat{x}, \hat{y}, \hat{z}; n_x, n_y, n_z, d)$, $\hat{f}(d)$ itself is a piecewise third degree polynomial. As in the 2D case, $\hat{f}(d)$ is monotonically decreasing. The polynomials are joined at d_1 to d_8 where the interface plane intersects the cell corners. Because of symmetry, $d_1 = -d_8$, $d_2 = -d_7$ etc. and $\hat{f}(d) = 1 - \hat{f}(-d)$. The equation can be solved numerically with a root-finding method, but as in 2D, it is possible to find the interval in which the solution lies, construct a polynomial for this interval, and solve the equation analytically.

5.3.4 Estimating the normal with ELVIRA

I chose to use the ELVIRA reconstruction algorithm found by Pilliod because it is reasonably fast and simple, and second order accurate according to [13]. It will reconstruct linear interfaces exactly. However, ELVIRA does not work well on too coarse grids and is only described for 2D.

In the ELVIRA algorithm, one finds six normal candidates, and then one picks the normal which gives the smallest error.

Consider a 3×3 block of cells about the i, j-th cell. Sum the colour function values in each row and column to create two histograms as shown in figure 5.12.

The purpose of summing the colour function values in each column is to create a discrete height function that approximates the interface. In effect, all the dark fluid in each column is moved to the bottom of the column. However, if the dark fluid was mostly at the top of the 3×3 block, moving the dark fluid to the bottom of each column would give a poor approximation to the interface. When most of the dark fluid is at the top,

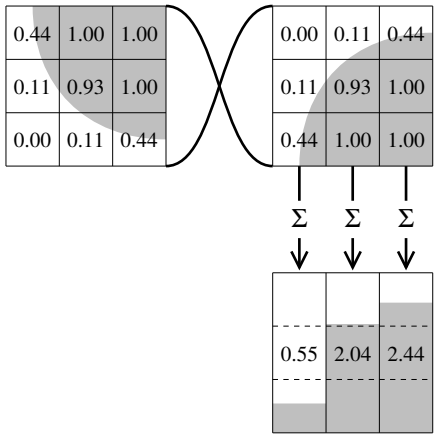


Figure 5.13: Histograms in the ELVIRA algorithm sometimes need to be mirrored.

I choose to implicitly[2] flip the 3×3 block upside down as shown in figure 5.13. Similarly for the rows as shown in figure 5.14. After I have found a normal, I flip the normal back to match the original 3×3 block. Another option is to rotate the 3×3 block until most of the dark fluid is in the lower left corner, calculate the normal and rotate the normal back.

To find out if most of the dark fluid is at the top or bottom, I simply compare the top row sum with the bottom row sum. If the top row sum is larger, then I assume that most of the dark fluid is at the top. Similarly, if the left column sum is larger than the right column sum, I assume that most of the dark fluid is to the left and vice versa.

Let C_1, C_2 and C_3 be column sums from left to right, and let R_1, R_2 and R_3 be the row sums from bottom to top. Assuming C_1, C_2 and C_3 are good approximations to the interface, we can estimate the slope a of the interface with a forward, backward and central difference (see figure 5.15):

$$a \approx \frac{\Delta y(C_2 - C_1)}{\Delta x}$$
$$a \approx \frac{\Delta y(C_3 - C_1)}{2\Delta x}$$
$$a \approx \frac{\Delta y(C_3 - C_2)}{\Delta x}$$

where Δx and Δy are the grid spacing.

[2]There is no need to actually move the colour function values around. The column sums are the same either way.

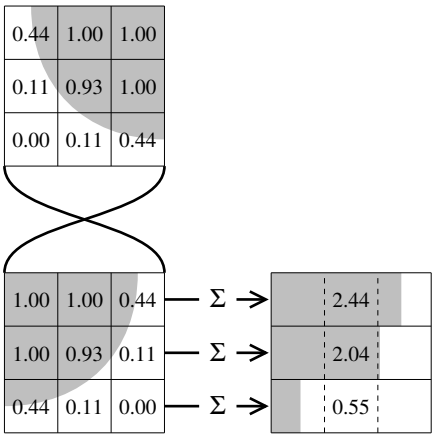


Figure 5.14: Histograms in the ELVIRA algorithm sometimes need to be mirrored.

The normalised normal corresponding to the slope a is (see figure 5.16):

$$\boldsymbol{n} = \frac{1}{\sqrt{1+a^2}} \begin{bmatrix} -a \\ 1 \end{bmatrix} \tag{5.66}$$

If the normal must be flipped upside down, just negate the y-component of the normal.

Similarly for the rows, the inverse slope[3] a^{-1} may be approximated:

$$a^{-1} \approx \frac{\Delta x(R_2 - R_1)}{\Delta y}$$
$$a^{-1} \approx \frac{\Delta x(R_3 - R_1)}{2\Delta y}$$
$$a^{-1} \approx \frac{\Delta x(R_3 - R_2)}{\Delta y}$$

The normalised normal corresponding to the inverse slope a^{-1} is:

$$\boldsymbol{n} = \frac{1}{\sqrt{1+(a^{-1})^2}} \begin{bmatrix} 1 \\ -a^{-1} \end{bmatrix} \tag{5.67}$$

If the normal must be flipped sideways, just negate the x-component of the normal.

Now there are six candidates for the normal in the centre cell. The error is calculated for each normal, then the normal resulting in the smallest error

[3]Since the slope generally is $\frac{dy}{dx}$, I call $\frac{dx}{dy}$ the inverse slope.

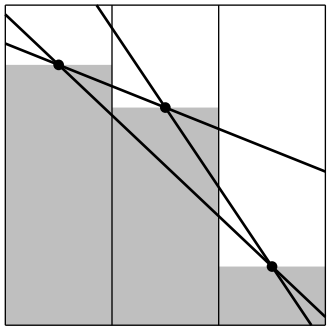


Figure 5.15: Forward, backward and central difference give three possible slopes.

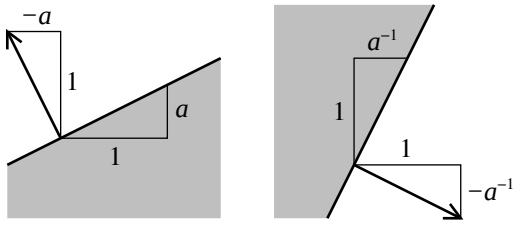


Figure 5.16: The normal corresponding to a given slope and inverse slope.

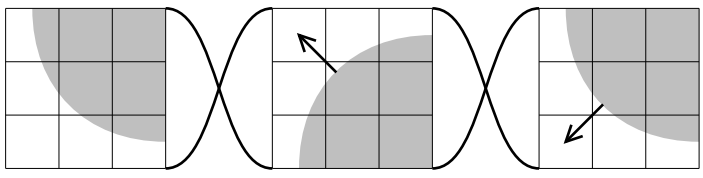


Figure 5.17: If most of the dark fluid is at the top of the 3×3 block, flip the block, find a normal, and flip the normal back.

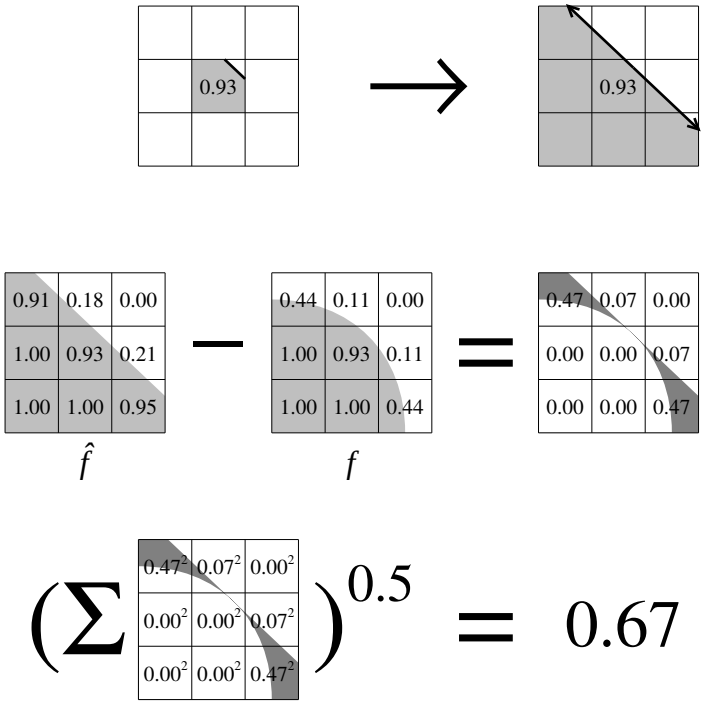


Figure 5.18: Extend the reconstructed interface and calculate the volume fraction for each cell based on the reconstructed interface. Compare with the colour function values and calculate the L^2-norm (Euclidean norm) of the difference.

is chosen. The error is calculated as follows: Given a normal $\boldsymbol{n}$, the interface is reconstructed for the i, j-th cell, which is the centre cell in the 3×3 block, as described in the section about PLIC reconstruction. Extend the linear interface to the edges of the entire 3×3 block and calculate the volume fraction $\hat{f}$ for each of the nine cells. Note that the signed distance must be calculated for each of the cells, as described in the last paragraph in section 5.3.1, so that each cell gets its own line equation relative to its centre. The error is (see figure 5.18):

$$E = \left(\sum_{k=i-1}^{i+1} \sum_{l=j-1}^{j+1} (\hat{f}_{k,l} - f_{k,l})^2 \right)^{\frac{1}{2}} \tag{5.68}$$

Use the chosen normal only to reconstruct the interface in the centre cell. Near the boundary, the 3×3 block will include cells that are outside the fluid domain. These cells, or ghost cells, must have values in accordance

with the boundary conditions, as described in section 5.2.

5.4 Advection

Advection of the colour function field can be described mathematically as:

$$\frac{\partial f}{\partial t} + \nabla \cdot (f\boldsymbol{u}) = 0 \tag{5.69}$$

where $\boldsymbol{u}$ is the velocity field and f the colour function field. I describe advection only in 2D, but advection in 3D is analogous. Discretise the equation with central difference in space and forward difference in time:

$$\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t} + \frac{(fu)_{i+1/2,j}^{n} - (fu)_{i-1/2,j}^{n}}{\Delta x} + \frac{(fv)_{i,j+1/2}^{n} - (fv)_{i,j-1/2}^{n}}{\Delta y} = 0 \tag{5.70}$$

$$f_{i,j}^{n+1} = f_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left[(fu)_{i+1/2,j}^{n} - (fu)_{i-1/2,j}^{n}\right] - \frac{\Delta t}{\Delta y}\left[(fv)_{i,j+1/2}^{n} - (fv)_{i,j-1/2}^{n}\right] \tag{5.71}$$

where $f_{i,j}^{n}$ is the colour function value for the i,j-th cell at the n-th time level, u is the horizontal component of the velocity and v the vertical component. Think of $f \cdot u$ and $f \cdot v$ as flux rate, that is, an amount of dark fluid passing through each of the cell's four faces per time.

To keep the interface sharp, the advection algorithm takes the reconstructed linear interface into account when calculating the face centred values of f ($f_{i+1/2,j}$, $f_{i-1/2,j}$, $f_{i,j+1/2}$ and $f_{i,j-1/2}$). Operator splitting seems to be the simplest approach[26, 32], though unsplit methods are presented in [24, 11]. The advection is split into a horizontal advection and a vertical advection. If the order of the splitting alternates for each time step, the advection algorithm is second order accurate in time, according to [26]. That is, if the colour function is advected horizontally, then vertically at one time level, it should be advected vertically, then horizontally at the next time level and vice versa. In 3D, the advection is split into three 1D advections where the order is interchanged each time level ($3! = 6$ different orders)[17]. Since the advection is performed in two steps, we first calculate some intermediate colour function f^*, then finally the colour function at the next time level f^{n+1}.

First, consider horizontal advection through a vertical face between two cells. The horizontal velocity component $u_{i+1/2,j}$ at the centre of the face is known. During the next time step Δt, the fluid at a distance of less than $|\Delta t \cdot u_{i+1/2,j}|$ upstream from the face will travel through the face as shown in figure 5.19. The hatched area in the figure is the *donating region*[32]. The amount of dark fluid inside the donating region can be calculated using the method described in the section about calculating volume fractions. Let

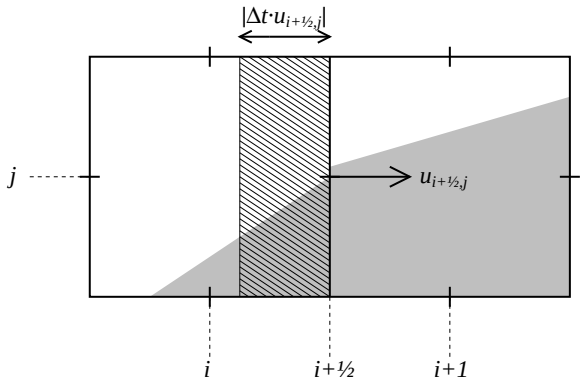


Figure 5.19: The fluid in the hatched area is advected from the left cell to the right during time step Δt.

$F_{i+1/2,j}$ be the volume fraction of dark fluid in this region. It is tempting to calculate the intermediate colour function as:

$$f^*i,j = f^n_{i,j} - \frac{\Delta t}{\Delta x}((F\cdot u)^n_{i+1/2,j} - (F\cdot u)^n_{i-1/2,j}) \tag{5.72}$$

However, in 1D, the fluid is not incompressible, and the amount of dark and light fluid does not necessarily add up to $\Delta x\Delta y$ any more. To find the intermediate colour function, which should be in the interval $[0,1]$, one must therefore divide the amount of dark fluid by the total amount of fluid[27]:

$$\begin{aligned} f^*i,j &= \frac{f^n_{i,j} - \left[\Delta t/\Delta x)((F\cdot u)^n_{i+1/2,j} - (F\cdot u)^n_{i-1/2,j}\right]}{1-(\Delta t/\Delta x)\left[(1\cdot u)^n_{i+1/2,j} - (1\cdot u)^n_{i-1/2,j}\right]} \\ &= \frac{f^n_{i,j} - \left[\Delta t/\Delta x)((F\cdot u)^n_{i+1/2,j} - (F\cdot u)^n_{i-1/2,j}\right]}{1-(\Delta t/\Delta x)\left[u^n_{i+1/2,j} - u^n_{i-1/2,j}\right]} \end{aligned} \tag{5.73}$$

Once the intermediate colour function f^* is found, the interface is reconstructed again from f^*. The vertical advection remains. Now, the flux through the top and bottom face of the cell must be found. Let $G_{i,j+1/2}$ be the volume fraction of dark fluid in the donating region $|\Delta t v_{i,j+1/2}|$ upstream from the face between the i,j-th and $i,j+1$-th cell. Undo the scaling of f^* to get back the actual amount of dark fluid (divided by the cell size) and add the change from vertical advection. Remember that

$(u_{i+1/2,j} - u_{i-1/2,j})/\Delta x + (v_{i,j+1/2} - v_{i,j-1/2})/\Delta y = 0$:

$$\begin{aligned} f_{i,j}^{n+1} &= f_{i,j}^{*}\left[1 - \frac{\Delta t}{\Delta x}(u_{i+1/2,j}^{n} - u_{i-1/2,j}^{n})\right] - \frac{\Delta t}{\Delta y}\left[(G\cdot v)_{i,j+1/2}^{n} - (G\cdot v)_{i,j-1/2}^{n}\right] \\ &= f_{i,j}^{*} - f_{i,j}^{*}\frac{\Delta t}{\Delta x}(u_{i+1/2,j}^{n} - u_{i-1/2,j}^{n}) - \frac{\Delta t}{\Delta y}\left[(G\cdot v)_{i,j+1/2}^{n} - (G\cdot v)_{i,j-1/2}^{n}\right] \\ &= f_{i,j}^{*} + f_{i,j}^{*}\frac{\Delta t}{\Delta y}(v_{i,j+1/2}^{n} - v_{i,j-1/2}^{n}) - \frac{\Delta t}{\Delta y}\left[(G\cdot v)_{i,j+1/2}^{n} - (G\cdot v)_{i,j-1/2}^{n}\right] \\ &= f_{i,j}^{*} + \frac{\Delta t}{\Delta y}\left(f_{i,j}^{*}(v_{i,j+1/2}^{n} - v_{i,j-1/2}^{n}) - \left[(G\cdot v)_{i,j+1/2}^{n} - (G\cdot v)_{i,j-1/2}^{n}\right]\right) \end{aligned} \tag{5.74}$$

By inserting equation (5.73) into equation 5.74, one can see that the combined equation is of the same form as the discretised advection equation (5.71). If one does not use operator splitting, but instead calculates both horizontal and vertical fluxes naively from the same colour function f^n, some fluid will be advected twice as shown in figure 5.20. This easily causes overshoots and undershoots, that is, the colour function value goes below zero or above one. Unsplit advection algorithms are usually more complex to avoid advecting fluid twice. According to [32], overshoots and undershoots can still occur using operator splitting. In such cases, I clamp the colour function values to the range $[0, 1]$. Thus mass will generally not be conserved exactly, but the errors are relatively small. In [32], the errors are said to be in the order of 10^{-4} in general, which agrees with my experience.

5.4.1 Time-step restriction

There are certain apparent conditions that need to be fulfilled if the advection is to be numerically stable:

- A donating region must not be larger than one cell.
- Two donating regions must not overlap.

The first condition can be fulfilled by ensuring that:

$$|\Delta t \cdot u| \leq \Delta x \tag{5.75}$$
$$|\Delta t \cdot v| \leq \Delta y \tag{5.76}$$

Two donating regions might overlap if the velocity at the centre of two opposing faces points away from the cell, as shown in figure 5.21. The overlap can be avoided by ensuring that[13]:

$$\Delta t \cdot u_{i+1/2,j} - \Delta t \cdot u_{i-1/2,j} < \Delta x \tag{5.77}$$
$$\Delta t \cdot v_{i,j+1/2} - \Delta t \cdot v_{i,j-1/2} < \Delta y \tag{5.78}$$

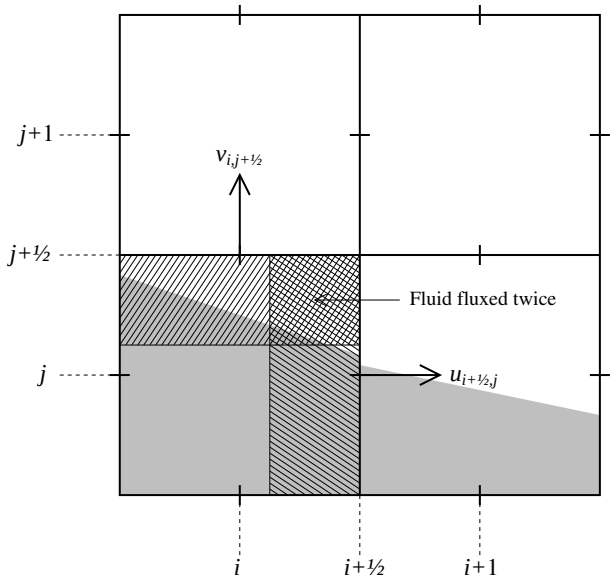


Figure 5.20: In a naive advection implementation, the fluid in the cross-hatched region is fluxed through both the top face and the right face.

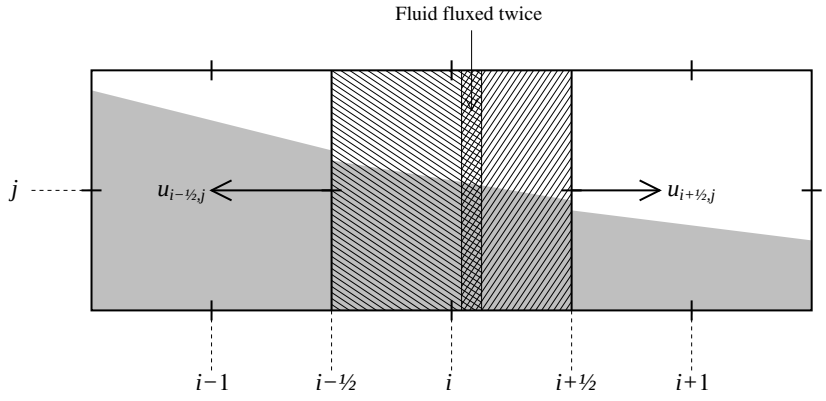


Figure 5.21: Overlapping donating regions must be avoided.

where the strict inequality will prevent division by zero in equation (5.73).

Combine all the restrictions:

$$\Delta t < \min\left(\frac{\Delta x}{|u_{i+1/2,j} - u_{i-1/2,j}|_{max}}, \frac{\Delta y}{|v_{i,j+1/2} - v_{i,j-1/2}|_{max}}, \frac{\Delta x}{|u|_{max}}, \frac{\Delta y}{|v|_{max}}\right) \tag{5.79}$$

Since $|u_{i+1/2,j} - u_{i-1/2,j}| \leq |u_{i+1/2,j}| + |u_{i-1/2,j}| \leq 2|u|_{max}$, it is possible to use a stricter, but simpler restriction[13]:

$$\Delta t < \min\left(\frac{\Delta x}{2|u|_{max}}, \frac{\Delta y}{2|v|_{max}}\right) \tag{5.80}$$

which I used in my implementation.

5.5 Surface tension

The surface tension force is a force which acts on the interface between the two fluids. The surface tension force per area can be expressed as[2, 19]:

$$\boldsymbol{F}_{sa} = \sigma\kappa\boldsymbol{n} \tag{5.81}$$

where σ is the surface tension coefficient, κ is the curvature of the dark fluid (positive if the centre of curvature is in the dark fluid) and $\boldsymbol{n}$ is the interface normal pointing into the dark fluid. In my implementation, I assume that the surface tension coefficient σ is constant. The curvature is defined as (2D and 3D):

$$\kappa = \frac{1}{R} \tag{5.82}$$

$$\kappa = \kappa_1 + \kappa_2 = \frac{1}{R_1} + \frac{1}{R_2} \tag{5.83}$$

where R is the radius of the circle which best approximates the interface locally in 2D, κ_1 and κ_2 are the principal curvatures in 3D, and R_1 and R_2 are their respective radii[4]. Define a curve as the intersection between the interface and a plane. Let the plane contain a point $\boldsymbol{p}$ on the interface and be parallel to the normal $\boldsymbol{n}$ there. The curve is two-dimensional (lies in the plane) and has an associated curvature at $\boldsymbol{p}$. The curvature varies with the choice of the plane. The principal curvatures at the point $\boldsymbol{p}$ are defined as the maximum and minimum curvature.

The surface tension force must be included on the right hand side of the momentum equation. Since the nodes are fixed in space, and does not follow the interface, the surface tension force must be distributed among the

[4]In 3D, $(\kappa_1 + \kappa_2)/2$ is called mean curvature.

nodes closest to the interface. The surface tension force can be expressed as a volume force:

$$\boldsymbol{F}_{sv} = \sigma\kappa\nabla f \tag{5.84}$$

Alternatively, the surface force can be distributed among several cells[2]:

$$\boldsymbol{F}_{sv} = \sigma\kappa\nabla \tilde{f} \tag{5.85}$$

where $\tilde{f}$ is a smoothed version of the colour function f. Note that in some articles like [23, 7], the volume force is $-\sigma\kappa\nabla f$. The sign depends on the definition of the positive curvature. The complete momentum equation becomes:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla p + \nabla\cdot\left[\mu\left(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T\right)\right] + \rho\boldsymbol{g} + \sigma\kappa\nabla f \tag{5.86}$$

There are drawbacks with both methods. Discontinuities can cause trouble with stability, and differentiating discontinuous functions is questionable. It might therefore be a good idea to smooth the colour function and surface tension a bit. But the surface tension is only defined at the interface, and applying some fictitious surface tension anywhere else can introduce errors. I tested both methods, and to me, it seems like the sharp surface tension gives the most accurate results, mostly because the curvature cannot be well approximated away from the interface. I therefore ended up using the sharp surface tension implementation.

A common test for surface tension implementation is the perfectly circular drop in zero gravity. Surface tension will try to pull the drop together, and the pressure difference between the inside and outside of the drop will try to pull the drop apart. Because of symmetry, the two forces should be equal in opposite directions, and the drop should stay in equilibrium. Since the velocity and gravity are zero, the momentum equation becomes:

$$0 = -\nabla p + \sigma\kappa\nabla f \tag{5.87}$$

Since the pressure and surface tension should balance, ∇p and ∇f should be discretised the same way:

$$\left(\frac{\partial f}{\partial x}\right)_{i+1/2,j} \approx \frac{f_{i+1,j} - f_{i,j}}{\Delta x} \tag{5.88}$$

$$\left(\frac{\partial f}{\partial y}\right)_{i,j+1/2} \approx \frac{f_{i,j+1} - f_{i,j}}{\Delta y} \tag{5.89}$$

Now, the curvature must somehow be approximated. I implemented two methods: CSF and DAC. In addition to these two, I also implemented a modified version of the DAC method to give more accurate results for circles.

5.5.1 Continuum surface force

The CSF method was introduced in an article by Brackbill, Kothe and Zemach[2]. The method describes both the curvature estimation and the smoothing of the surface tension over a few cell widths near the interface. However, I only use the curvature estimation, since I find that smoothing of the surface tension reduces the accuracy.

The curvature is calculated from a smoothed colour function field $\tilde{f}$. The smoothing is the same as described in the section 5.1. In my tests, too much smoothing caused folds to appear on the interface. I suspect that the folds were so small that they disappeared in the smoothed colour function field $\tilde{f}$. Since the surface force is calculated from the smoothed colour function field, no surface force was generated to counteract the forming of the folds. I guess that the folds appeared in the first place due to inaccurate curvature estimation. A high surface tension coefficient may also have been a factor. However, as long as the smoothing stencil was 3×3 in size, I experienced no folds on the interface.

Let $\boldsymbol{n} = \nabla \tilde{f}$, a normal vector which is not normalised, pointing into the dark fluid. According to the [2], the curvature can be expressed as:

$$\kappa = -\nabla \cdot \frac{\boldsymbol{n}}{|\boldsymbol{n}|} \tag{5.90}$$

Two discretisations of the above expression are presented in the article: "MAC-like scheme" and "ALE-like scheme". I implemented the ALE-like scheme because it gives more accurate results than the MAC-like scheme[2].

Before discretising, the expression is rewritten using the quotient rule:

$$\begin{aligned}
\kappa &= -\nabla \cdot \frac{\boldsymbol{n}}{|\boldsymbol{n}|} \\
&= -\left(\frac{\partial}{\partial x} \frac{n_x}{|\boldsymbol{n}|} + \frac{\partial}{\partial y} \frac{n_y}{|\boldsymbol{n}|} \right) \\
&= -\left(\frac{1}{|\boldsymbol{n}|^2} \left(|\boldsymbol{n}| \frac{\partial n_x}{\partial x} - n_x \frac{\partial |\boldsymbol{n}|}{\partial x} \right) + \frac{1}{|\boldsymbol{n}|^2} \left(|\boldsymbol{n}| \frac{\partial n_y}{\partial y} - n_y \frac{\partial |\boldsymbol{n}|}{\partial y} \right) \right) \\
&= -\frac{1}{|\boldsymbol{n}|^2} \left(|\boldsymbol{n}| \frac{\partial n_x}{\partial x} - n_x \frac{\partial |\boldsymbol{n}|}{\partial x} + |\boldsymbol{n}| \frac{\partial n_y}{\partial y} - n_y \frac{\partial |\boldsymbol{n}|}{\partial y} \right) \\
&= -\frac{1}{|\boldsymbol{n}|^2} \left(|\boldsymbol{n}| \left(\frac{\partial n_x}{\partial x} + \frac{\partial n_y}{\partial y} \right) - \left(n_x \frac{\partial |\boldsymbol{n}|}{\partial x} + n_y \frac{\partial |\boldsymbol{n}|}{\partial y} \right) \right) \\
&= -\frac{1}{|\boldsymbol{n}|^2} \left(|\boldsymbol{n}| \nabla \cdot \boldsymbol{n} - \boldsymbol{n} \cdot \nabla |\boldsymbol{n}| \right) \\
&= \frac{1}{|\boldsymbol{n}|} \left[\left(\frac{\boldsymbol{n}}{|\boldsymbol{n}|} \cdot \nabla \right) |\boldsymbol{n}| - (\nabla \cdot \boldsymbol{n}) \right]
\end{aligned} \tag{5.91}$$

First, the vertex-centred[5] normals $\boldsymbol{n}_{i+1/2,j+1/2}$ are found by averaging two

[5] located in cell corners

central differences:

$$\begin{aligned} n_{xi+1/2,j+1/2} &= \frac{1}{2}\left[\left(\frac{\partial \tilde{f}}{\partial x}\right)_{i+1/2,j} + \left(\frac{\partial \tilde{f}}{\partial x}\right)_{i+1/2,j+1}\right] \\ &= \frac{1}{2}\left[\frac{\tilde{f}_{i+1,j} - \tilde{f}_{i,j}}{\Delta x} + \frac{\tilde{f}_{i+1,j+1} - \tilde{f}_{i,j+1}}{\Delta x}\right] \\ &= \frac{\tilde{f}_{i+1,j} + \tilde{f}_{i+1,j+1} - \tilde{f}_{i,j} - \tilde{f}_{i,j+1}}{2\Delta x} \end{aligned} \tag{5.92}$$

$$n_{y_{i+1/2,j+1/2}} = \frac{\tilde{f}_{i,j+1} + \tilde{f}_{i+1,j+1} - \tilde{f}_{i,j} - \tilde{f}_{i+1,j}}{2\Delta y} \tag{5.93}$$

Cell-centred normals $\boldsymbol{n}_{i,j}$ are found by averaging the four surrounding vertex-centred normals:

$$\boldsymbol{n}_{i,j} = \frac{1}{4}\left(\boldsymbol{n}_{i+1/2,j+1/2} + \boldsymbol{n}_{i+1/2,j-1/2} + \boldsymbol{n}_{i-1/2,j+1/2} + \boldsymbol{n}_{i-1/2,j-1/2}\right) \tag{5.94}$$

Let the magnitude of the normal be $|\boldsymbol{n}| = \sqrt{n_x^2 + n_y^2}$. Like for normals, the magnitude is defined in cell corners and centres.

$\left(\frac{\boldsymbol{n}}{|\boldsymbol{n}|} \cdot \nabla\right)|\boldsymbol{n}|$ is discretised as follows:

$$\begin{aligned} \left[\left(\frac{\boldsymbol{n}}{|\boldsymbol{n}|} \cdot \nabla\right)|\boldsymbol{n}|\right]_{i,j} &= \left(\frac{n_x}{|\boldsymbol{n}|}\right)_{i,j}\left(\frac{\partial |\boldsymbol{n}|}{\partial x}\right) + \left(\frac{n_y}{|\boldsymbol{n}|}\right)_{i,j}\left(\frac{\partial |\boldsymbol{n}|}{\partial y}\right) \\ = \frac{n_{xi,j}}{|\boldsymbol{n}_{i,j}|}&\left(\frac{|\boldsymbol{n}|_{i+1/2,j+1/2} - |\boldsymbol{n}|_{i-1/2,j+1/2} + |\boldsymbol{n}|_{i+1/2,j-1/2} - |\boldsymbol{n}|_{i-1/2,j-1/2}}{2\Delta x}\right) \\ + \frac{n_{y_{i,j}}}{|\boldsymbol{n}_{i,j}|}&\left(\frac{|\boldsymbol{n}|_{i+1/2,j+1/2} - |\boldsymbol{n}|_{i+1/2,j-1/2} + |\boldsymbol{n}|_{i-1/2,j+1/2} - |\boldsymbol{n}|_{i-1/2,j-1/2}}{2\Delta y}\right) \end{aligned} \tag{5.95}$$

$\nabla \cdot \boldsymbol{n}$ is discretised as follows:

$$\begin{aligned} (\nabla \cdot \boldsymbol{n})_{i,j} &= \left(\frac{\partial n_x}{\partial x}\right)_{i,j} + \left(\frac{\partial n_y}{\partial y}\right)_{i,j} \\ &= \frac{n_{xi+1/2,j+1/2} + n_{xi+1/2,j-1/2} - n_{xi-1/2,j+1/2} - n_{xi-1/2,j-1/2}}{2\Delta x} \\ &+ \frac{n_{y_{i+1/2,j+1/2}} + n_{y_{i-1/2,j+1/2}} - n_{y_{i+1/2,j-1/2}} - n_{y_{i-1/2,j-1/2}}}{2\Delta y} \end{aligned} \tag{5.96}$$

The curvature can thus be calculated at cell centres in the vicinity of the interface. Since a staggered grid is used, the surface tension must be applied at face centres. The face-centred curvature is found by averaging two cell-centred curvatures:

$$\kappa_{i+1/2,j} = \frac{1}{2}(\kappa_{i,j} + \kappa_{i+1,j}) \tag{5.97}$$

5.5.2 Direction averaged curvature

The DAC method is described in [17], but a very similar method is also described in [27]. I choose to interpret the methods in a way that makes most sense to me. The method is second order accurate[17, 27].

In most cases, it is sufficient to calculate the curvature for cells containing the interface, but there are rare cases where the interface may pass exactly on the face between two cells. I therefore calculate the curvature for any cell fulfilling the following condition:

$$0 < f_{i,j} < 1 \vee |f_{i,j} - f_{i\pm1,j}| = 1 \vee |f_{i,j} - f_{i,j\pm1}| = 1 \tag{5.98}$$

Before calculating the curvature, the interface is reconstructed for all cells containing or touching the interface. The normal is then used to find the main orientation of the interface. There are four cases that are treated separately:

1. $n_y \geq |n_x|$: The normal is mainly pointing upwards.
2. $n_y \leq -|n_x|$: The normal is mainly pointing downwards.
3. $n_x > |n_y|$: The normal is mainly pointing right.
4. $n_x < -|n_y|$: The normal is mainly pointing left.

Consider case 1. The other cases are treated analogously. Look at a block of cells three cells wide and seven cells tall. As in the ELVIRA method, sum colour function values f in each column. Start the summation from the middle row. Sum upwards until one of the following cells are found:

- A cell containing only light fluid.
- An obstacle cell.
- A cell outside the computational domain.

Now, continue summing downwards from the middle row, but instead of adding colour function value f, add $f - 1$. Stop the summation when one of the following cells are found:

- A cell containing only dark fluid.
- An obstacle cell.
- A cell outside the computational domain.

This results in a histogram with its baseline at the bottom face of the centre cell, as shown in figure 5.22. Let the column sums be C_1, C_2 and

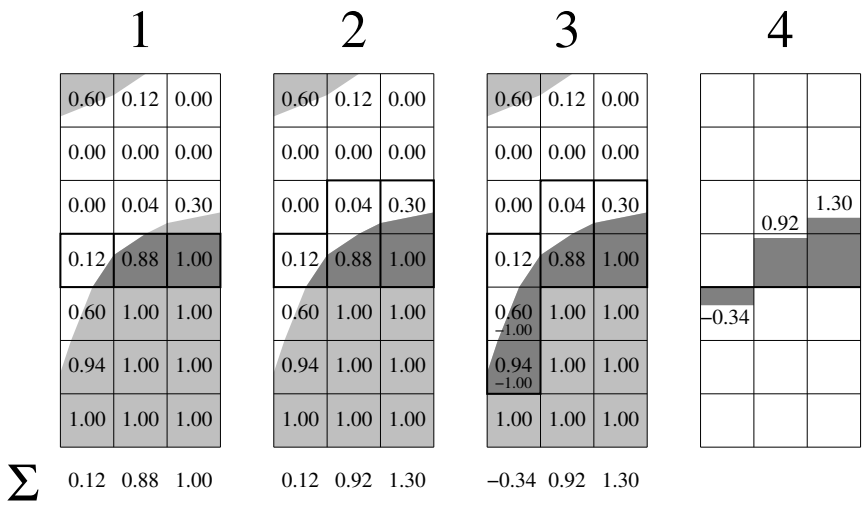


Figure 5.22: 1: Start the summation from the middle row. 2: Sum upwards as long as the cells contain dark fluid. 3: Sum downwards as long as the cells contain light fluid, but subtract one for each cell added. 4: The sums result in a histogram which approximates the interface.

C_3. The sums can be regarded as integrals of a height function $h(x)$ which defines the interface locally:

$$\Delta x \Delta y C_1 = \int_{x_{i-3/2}}^{x_{i-1/2}} h(s)ds \tag{5.99}$$

$$\Delta x \Delta y C_2 = \int_{x_{i-1/2}}^{x_{i+1/2}} h(s)ds \tag{5.100}$$

$$\Delta x \Delta y C_3 = \int_{x_{i+1/2}}^{x_{i+3/2}} h(s)ds \tag{5.101}$$

For a function $h(x)$, the curvature is:

$$\kappa = -\frac{h''(x)}{\left[1 + (h'(x))^2\right]^{3/2}} \tag{5.102}$$

Both $h''(x_i)$ and $h'(x_i)$ can be approximated with the column sums:

$$h''(x_i) \approx \frac{\Delta y(C_1 - 2C_2 + C_1)}{\Delta x^2} \tag{5.103}$$

$$h'(x_i) \approx \frac{\Delta y(C_3 - C_1)}{2\Delta x} \tag{5.104}$$

The method can be extended to 3D. Assume that the normal is pointing mainly upwards, that is $n_z > |n_x| \wedge n_z > |n_y|$. Consider a block of cells three

cells wide and deep, and seven cells tall. As in 2D, calculate the nine column sums $C_{1,1}$ to $C_{3,3}$. The sums are integrals of a height function $h(x, y)$ which approximates the interface locally. For instance:

$$C_{1,1} = \int_{x_{i-3/2}}^{x_{i-1/2}} \int_{y_{j-3/2}}^{y_{j-1/2}} h(s,t)dt\ ds \tag{5.105}$$

The curvature for $h(x, y)$ is:

$$\kappa = -\frac{\left(1 + \left(\frac{\partial h}{\partial x}\right)^2\right)\frac{\partial^2 h}{\partial y^2} - 2\frac{\partial h}{\partial x}\frac{\partial h}{\partial y}\frac{\partial^2 h}{\partial x \partial y} + \left(1 + \left(\frac{\partial h}{\partial y}\right)^2\right)\frac{\partial^2 h}{\partial x^2}}{\left(1 + \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2\right)^{3/2}} \tag{5.106}$$

The derivatives can be approximated with central differences:

$$\frac{\partial h}{\partial x} \approx \frac{\Delta z(C_{3,2} - C_{1,2})}{2\Delta x} \tag{5.107}$$

$$\frac{\partial h}{\partial y} \approx \frac{\Delta z(C_{2,3} - C_{2,1})}{2\Delta y} \tag{5.108}$$

$$\frac{\partial^2 h}{\partial x^2} \approx \frac{\Delta z(C_{1,2} - 2C_{2,2} + C_{3,2})}{\Delta x^2} \tag{5.109}$$

$$\frac{\partial^2 h}{\partial y^2} \approx \frac{\Delta z(C_{2,1} - 2C_{2,2} + C_{2,3})}{\Delta y^2} \tag{5.110}$$

$$\frac{\partial^2 h}{\partial x \partial y} \approx \frac{\Delta z(C_{3,1} + C_{1,3} - C_{1,1} - C_{3,3})}{4\Delta x \Delta y} \tag{5.111}$$

The curvature is defined at the centre of cells containing the interface. Because a staggered grid is used, the surface tension is applied at face centres. To find the curvature at the face centres, some kind of average should be used. On the face between the i, j-th and $i+1, j$-th cells, we can have at least one of the following cases:

1. $f_{i,j} = f_{i+1,j}$: The surface tension becomes zero because $\frac{\partial f}{\partial x}$ evaluates to zero, so the curvature does not matter.

2. $0 < f_{i,j}, f_{i+1,j} < 1$: The curvature is defined in both cells, so any average can be used.

3. $0 < f_{i,j} < 1 \wedge f_{i+1,j} \in \{0, 1\}$ or vice versa: The curvature is only defined in one of the cells, so only this curvature should be used.

4. $|f_{i,j} - f_{i+1,j}| = 1$: The curvature is defined in both cells, so any average can be used.

Since there are cases where the curvature is only defined on one side of a face, some kind of weighted average can be used:

$$\kappa_{i+1/2,j} = \frac{w_{i,j}\kappa_{i,j} + w_{i+1,j}\kappa_{i+1,j}}{w_{i,j} + w_{i+1,j}} \tag{5.112}$$

It is clear that the second and third case can be covered by requiring:

$$w_{i,j} \begin{cases} = 0 & : \ f_{i,j} \in \{0, 1\} \\ > 0 & : \ f_{i,j} \in (0, 1) \end{cases} \tag{5.113}$$

In [23], the following weighting was used because the curvature approximation was expected to be more accurate in cells where f is not close to zero or one:

$$w_{i,j} = f_{i,j}(1 - f_{i,j}) \tag{5.114}$$

However, case one may, and case four will cause division by zero. Instead of detecting such cases and treating them specially, I chose to make a small hack:

$$w_{i,j} = f_{i,j}(1 - f_{i,j}) + \epsilon \tag{5.115}$$

where ϵ is very small, for instance 10^{-16}. Such a small value will be insignificant in case two and three, division by zero is avoided in case one, and a proper average will be defined for case four. This solution is also vectorisation friendly.

5.5.3 Direction averaged curvature with refinement

Spurious currents are a common problem in many surface tension implementations. Spurious currents are non-physical, vortical flows near an interface. The main reason spurious currents appear, is imbalance between the pressure and the surface tension, for instance due to inaccurate calculation of the curvature[7]. Many articles about how spurious currents can be reduced or how to estimate the curvature better have been published[18, 23, 20, 30, 22].

In an attempt to get rid of spurious currents, I modified the DAC method. In the DAC method, the curvature will not be calculated exactly for a circular interface. Thus, the net force on the interface of a static circular drop in zero gravity will not add up to zero as it should. I therefore suggest modifying the method such that the curvature is calculated exactly for circular interfaces.

I calculate the three column sums C_1, C_2 and C_3 as in the DAC method. I then try to find a circular interface which will result in these sums. Let the height function be a semi-circle:

$$h(x; c_x, c_y, r) = c_y + \sqrt{\max(0, r^2 - (x - c_x)^2)} \tag{5.116}$$

where (c_x, c_y) is the circle centre and r is the circle radius.

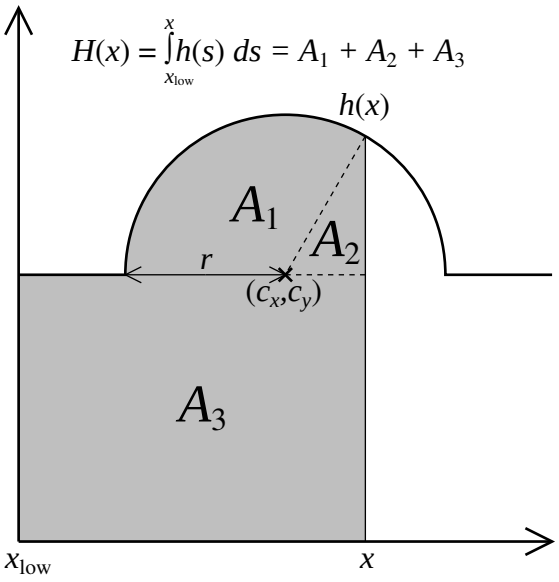


Figure 5.23: The integral of $h(x)$ is found geometrically.

The integral of $h(x; c_x, c_y, r)$ can be found geometrically (see figure 5.23):

$$\begin{aligned} H(x; c_x, c_y, r) &= \int_{x_{low}}^{x} h(s; c_x, c_y, r) ds \\ &= \underbrace{\frac{r^2}{2} \arccos\left[\text{clamp}\left(-\frac{x - c_x}{r}, -1, 1\right)\right]}_{A_1} \\ &+ \underbrace{\frac{1}{2}(x - c_x)\sqrt{\max(0, r^2 - (x - c_x)^2)}}_{A_2} \\ &+ \underbrace{(x - x_{low})c_y}_{A_3} \end{aligned} \tag{5.117}$$

where $\text{clamp}(x, a, b) = \max(a, \min(b, x))$ and $x_{low} < c_x - r$. The exact value of x_{low} is unimportant, since it will be eliminated in the final expressions. The areas A_1, A_2 and A_3 are shown in the figure.

The following equations must be solved with respect to c_x, c_y and r:

$$\begin{aligned} \Delta x \Delta y C_1 &= H(x_{i-1/2}; c_x, c_y, r) - H(x_{i-3/2}; c_x, c_y, r) \\ \Delta x \Delta y C_2 &= H(x_{i+1/2}; c_x, c_y, r) - H(x_{i-1/2}; c_x, c_y, r) \\ \Delta x \Delta y C_3 &= H(x_{i+3/2}; c_x, c_y, r) - H(x_{i+1/2}; c_x, c_y, r) \end{aligned} \tag{5.118}$$

I chose to solve the equations using Newton's method for multiple variables. To solve an equation $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, the following iteration is used:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - {\boldsymbol{J_f}}^{-1}(\boldsymbol{x}_n)\boldsymbol{f}(\boldsymbol{x}_n) \tag{5.119}$$

where $\boldsymbol{x}$ is the unknown vector, $\boldsymbol{f}(\boldsymbol{x})$ is a vector function and $\boldsymbol{J_f}(\boldsymbol{x})$ is the Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$.

In equation (5.118), the unknown vector is $[c_x, c_y, r]^T$ and the vector function is:

$$\boldsymbol{f}(c_x, c_y, r) = \begin{bmatrix} H(x_{i-1/2}; c_x, c_y, r) - H(x_{i-3/2}; c_x, c_y, r) - \Delta x \Delta y C_1 \\ H(x_{i+1/2}; c_x, c_y, r) - H(x_{i-1/2}; c_x, c_y, r) - \Delta x \Delta y C_2 \\ H(x_{i+3/2}; c_x, c_y, r) - H(x_{i+1/2}; c_x, c_y, r) - \Delta x \Delta y C_3 \end{bmatrix} \tag{5.120}$$

To find the Jacobian matrix, the derivatives of $H(x; c_x, c_y, r)$ are needed:

$$\frac{\partial H}{\partial c_x} = -\sqrt{\max(0, r^2 - (x - c_x)^2)} \tag{5.121}$$

$$\frac{\partial H}{\partial c_y} = x - x_{low} \tag{5.122}$$

$$\frac{\partial H}{\partial r} = r \arccos\left[\text{clamp}\left(-\frac{x - c_x}{r}, -1, 1\right)\right] \tag{5.123}$$

The Jacobian matrix is:

$$\boldsymbol{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial c_x} & \frac{\partial f_1}{\partial c_y} & \frac{\partial f_1}{\partial r} \\ \frac{\partial f_2}{\partial c_x} & \frac{\partial f_2}{\partial c_y} & \frac{\partial f_2}{\partial r} \\ \frac{\partial f_3}{\partial c_x} & \frac{\partial f_3}{\partial c_y} & \frac{\partial f_3}{\partial r} \end{bmatrix} \tag{5.124}$$

where:

$$\begin{aligned} \frac{\partial f_1}{\partial c_x} &= \frac{\partial}{\partial c_x}(H(x_{i-1/2}; c_x, c_y, r) - H(x_{i-3/2}; c_x, c_y, r) - \Delta x \Delta y C_1) \\ &= -\sqrt{\max(0, r^2 - (x_{i-1/2} - c_x)^2)} + \sqrt{\max(0, r^2 - (x_{i-3/2} - c_x)^2)} \end{aligned} \tag{5.125}$$

$$\begin{aligned} \frac{\partial f_1}{\partial c_y} &= \frac{\partial}{\partial c_y}(H(x_{i-1/2}; c_x, c_y, r) - H(x_{i-3/2}; c_x, c_y, r) - \Delta x \Delta y C_1) \\ &= x_{i-1/2} - x_{i-3/2} \end{aligned} \tag{5.126}$$

$$\begin{aligned} \frac{\partial f_1}{\partial r} &= \frac{\partial}{\partial r}(H(x_{i-1/2}; c_x, c_y, r) - H(x_{i-3/2}; c_x, c_y, r) - \Delta x \Delta y C_1) \\ &= r \arccos\left[\text{clamp}\left(-\frac{x_{i-1/2} - c_x}{r}, -1, 1\right)\right] \\ &\quad - r \arccos\left[\text{clamp}\left(-\frac{x_{i-3/2} - c_x}{r}, -1, 1\right)\right] \end{aligned} \tag{5.127}$$

etc.

Since the Jacobian matrix is only 3×3, it can be inverted directly within reasonable time:

$$\boldsymbol{J_f}^{-1} = \frac{1}{\det(\boldsymbol{J_f})} \begin{bmatrix} \frac{\partial f_2}{\partial c_y}\frac{\partial f_3}{\partial r} - \frac{\partial f_2}{\partial r}\frac{\partial f_3}{\partial c_y} & \frac{\partial f_3}{\partial c_y}\frac{\partial f_1}{\partial r} - \frac{\partial f_3}{\partial r}\frac{\partial f_1}{\partial c_y} & \frac{\partial f_1}{\partial c_y}\frac{\partial f_2}{\partial r} - \frac{\partial f_1}{\partial r}\frac{\partial f_2}{\partial c_y} \\ \frac{\partial f_2}{\partial r}\frac{\partial f_3}{\partial c_x} - \frac{\partial f_2}{\partial c_x}\frac{\partial f_3}{\partial r} & \frac{\partial f_3}{\partial r}\frac{\partial f_1}{\partial c_x} - \frac{\partial f_3}{\partial c_x}\frac{\partial f_1}{\partial r} & \frac{\partial f_1}{\partial r}\frac{\partial f_2}{\partial c_x} - \frac{\partial f_1}{\partial c_x}\frac{\partial f_2}{\partial r} \\ \frac{\partial f_2}{\partial c_x}\frac{\partial f_3}{\partial c_y} - \frac{\partial f_2}{\partial c_y}\frac{\partial f_3}{\partial c_x} & \frac{\partial f_3}{\partial c_x}\frac{\partial f_1}{\partial c_y} - \frac{\partial f_3}{\partial c_y}\frac{\partial f_1}{\partial c_x} & \frac{\partial f_1}{\partial c_x}\frac{\partial f_2}{\partial c_y} - \frac{\partial f_1}{\partial c_y}\frac{\partial f_2}{\partial c_x} \end{bmatrix} \tag{5.128}$$

where:

$$\begin{aligned} \det(\boldsymbol{J_f}) = &\frac{\partial f_1}{\partial c_x}\frac{\partial f_2}{\partial c_y}\frac{\partial f_3}{\partial r} - \frac{\partial f_1}{\partial r}\frac{\partial f_2}{\partial c_y}\frac{\partial f_3}{\partial c_x} \\ &+ \frac{\partial f_1}{\partial c_y}\frac{\partial f_2}{\partial r}\frac{\partial f_3}{\partial c_x} - \frac{\partial f_1}{\partial c_x}\frac{\partial f_2}{\partial r}\frac{\partial f_3}{\partial c_y} \\ &+ \frac{\partial f_1}{\partial r}\frac{\partial f_2}{\partial c_x}\frac{\partial f_3}{\partial c_y} - \frac{\partial f_1}{\partial c_y}\frac{\partial f_2}{\partial c_x}\frac{\partial f_3}{\partial r} \end{aligned} \tag{5.129}$$

I first calculate the curvature with the usual DAC method. If Newton's method diverges, I fall back to this curvature value. If the interface is nearly flat, I assume that the original DAC method is quite accurate, while my method will become less accurate with low curvatures. I therefore only apply my method if the absolute curvature is above some small value. I used $1/100$ in my implementation, but I believe it can be set much lower.

My method requires a convex dark fluid area, so if the curvature calculated with DAC is negative (concave dark fluid area), I swap the dark and light fluids and negate the calculated curvature value before returning it. The dark and light fluids are swapped by negating the column sums:

$$C_1 \leftarrow -C_1 \tag{5.130}$$

$$C_2 \leftarrow -C_2 \tag{5.131}$$

$$C_3 \leftarrow -C_3 \tag{5.132}$$

Newton's method requires an initial guess for c_x, c_y and r. I approximate the interface with a parabola $\hat{h}(x)$ which pass through the points $(x_{i-1}, \Delta y C_1)$, $(x_i, \Delta y C_2)$ and $(x_{i+1}, \Delta y C_3)$. For the initial guess, I use the circle which best approximates the parabola $\hat{h}(x)$ at the point $(x_i, \Delta y C_2)$. This is done as follows:

- Calculate the slope of the parabola at x_i:

$$a = \hat{h}'(x_i) = \frac{\Delta y(C_3 - C_1)}{2\Delta x} \tag{5.133}$$

- Calculate the downward pointing normal $\boldsymbol{n}$ at x_i:

$$n_x = \frac{a}{\sqrt{1 + a^2}} \tag{5.134}$$

$$n_y = -\frac{1}{\sqrt{1 + a^2}} \tag{5.135}$$

- Calculate the curvature of the parabola at x_i:

$$\hat{h}''(x_i) = \frac{\Delta y(C_1 - 2C_2 + C_3)}{\Delta x^2} \tag{5.136}$$

$$\kappa = -\frac{\hat{h}''(x_i)}{\left[1 + \left(\hat{h}'(x_i)\right)^2\right]^{3/2}} \tag{5.137}$$

- Calculate the corresponding radius at x_i:

$$r = \frac{1}{\kappa} \tag{5.138}$$

- Calculate the centre of the initial circle:

$$c_x = x_i + r \cdot n_x \tag{5.139}$$

$$c_y = \Delta y C_2 + r \cdot n_y \tag{5.140}$$

For each iteration of Newton's method, I force the current values of c_x, c_y and r to represent a circle spanning at least $[x_{i-3/2}, x_{i+3/2}]$. If the circle becomes smaller than this, the Jacobian matrix may become singular. In practice, this size limit is not a problem, since other factors will cause greater inaccuracies. For instance, at such small radii, the column sums will not be accurate representations of the interface.

If the low end of the circle $c_x - r$ is greater than the low end $x_{i-3/2}$ of the first column, the circle is resized and repositioned:

$$c_{xnew} \leftarrow \frac{1}{2}(c_{xold} + r_{old} + x_{i-3/2}) \tag{5.141}$$

$$r_{new} \leftarrow \frac{1}{2}(c_{xold} + r_{old} - x_{i-3/2}) \tag{5.142}$$

Similarly, if the high end of the circle $c_x + r$ is less than the high end $x_{i+3/2}$ of the last column, the circle is resized and repositioned:

$$c_{xnew} \leftarrow \frac{1}{2}(x_{i+3/2} + c_{xold} - r_{old}) \tag{5.143}$$

$$r_{new} \leftarrow \frac{1}{2}(x_{i+3/2} - c_{xold} + r_{old}) \tag{5.144}$$

The residual can be calculated as $|\boldsymbol{f}(c_x, c_y, r)|$. The iteration is stopped after a fixed number of iterations or when the residual falls below some small value. The curvature is then $\kappa = 1/r$. If the residual is larger than it was at the beginning, I assume that Newton's method diverged, and I fall back to the original DAC curvature value.

The face centred curvatures are weighted averages of the two bordering cells as in the DAC method.

The method is not readily extendible to 3D.

5.5.4 Time-step restriction

Capillary waves are small surface waves where the dynamics is dominated by the surface tension. The time step Δt must be small enough to resolve the propagation of such waves[7, 2]:

$$\Delta t < \sqrt{\frac{(\rho_L + \rho_D)\min(\Delta x, \Delta y)^3}{4\pi\sigma}} \tag{5.145}$$

where ρ_L and ρ_D are the density of the light and dark fluid respectively, Δx and Δy are the grid spacing and σ is the surface tension coefficient.

Chapter 6

Verification

I have run the following tests to verify the code:

- Channel flow
- Pressure driven channel flow
- (Lid driven) square cavity
- Backward facing step
- Rising bubble
- Falling droplet
- Static drop
- Rayleigh-Taylor instability
- Pressure Poisson equation
- Zalesak's rotating disc
- Profiling

The coefficients used in tests are usually dimensionless, that is, without units like metres or seconds. The non-dimensionalised momentum equation is[19]:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \frac{1}{\rho Re} \nabla \cdot \left(\mu \left((\nabla \boldsymbol{u}) + (\nabla \boldsymbol{u})^T \right) \right) + \frac{\boldsymbol{e}_g}{Fr^2} + \frac{\boldsymbol{F}_{sv}}{\sigma \rho We} \quad (6.1)$$

where

- ρ is the dimensionless density.
- Re is the Reynolds number.

- μ is the dimensionless viscosity.
- Fr is the Froude number.
- $\boldsymbol{e}_g$ is the unit vector in the direction of gravity.
- We is the Weber number.
- $\boldsymbol{F}_{sv}$ is the surface tension force per volume.

However, my implementation has the following momentum equation which requires SI-units:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \frac{1}{\rho} \nabla \cdot \left[\mu \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T \right) \right] + \boldsymbol{g} + \frac{\boldsymbol{F}_{sv}}{\rho} \tag{6.2}$$

The two equations become equivalent if the coefficients with SI-units are replaced with the dimensionless coefficients listed in table 6.1.

6.1 Channel flow

The channel flow test was set up using the following parameters:

- *Domain*: $[0, 3] \times [0, 1]$
- *Fluid properties*: $\rho = 1$, $\mu = 1$
- *North and south boundary conditions*: $u = 0$, $v = 0$
- *West boundary conditions*: $u = 1$, $v = 0$
- *East boundary conditions*: $\frac{\partial u}{\partial x} = 0$, $\frac{\partial v}{\partial x} = 0$

The inlet velocity profile is flat ($u = 1$). It is expected that the flow will develop a parabolic velocity profile ($u(y) = 6y(1 - y)$) farther downstream. The maximum outlet velocity is expected to be 1.5 for a fully developed flow. The test was run until convergence ($t = 0.25$). The maximum outlet velocity for different grid resolutions is presented in table 6.2. The overall accuracy of the of the discretisation in space is of second order, but since it is only first order accurate near the boundary, we cannot expect exact results. As shown in table 6.2, the order of accuracy is about 2 ($O(\Delta y^2)$), as expected.

6.2 Pressure driven flow

The pressure driven flow test was set up using the following parameters:

- *Domain*: $[0, 3] \times [0, 1]$

- *Fluid properties*: $\rho = 1$, $\mu = 1$
- *North and south boundary conditions*: $u = 0$, $v = 0$
- *West boundary conditions*: $p = 24$, $\frac{\partial v}{\partial x} = 0$
- *East boundary conditions*: $p = 0$, $\frac{\partial v}{\partial x} = 0$

The test was run until convergence ($t = 2.0$). For a steady state solution, there is no acceleration. The momentum equation can therefore be expressed as:

$$0 = -\nabla p + \mu \nabla^2 \boldsymbol{u} \tag{6.3}$$

Because of symmetry, one can expect

$$\frac{\Delta p}{L} = \mu \frac{\partial^2 u}{\partial y^2} \tag{6.4}$$

where Δp is the pressure difference and L is the distance between the east and west boundary. Since the velocity is zero on the walls, the following expression can be derived for u:

$$u(y) = -\frac{\Delta p}{2L\mu} y(1-y) = 4y(1-y) \tag{6.5}$$

At the centre, the horizontal velocity component is thus expected to be $u(0.5) = 1$. As shown in table 6.3, the order of accuracy is about 2 ($O(\Delta y^2)$), as expected.

6.3 Square cavity

The square cavity test was set up using the following parameters:

- *Domain*: $[0, 1] \times [0, 1]$
- *North boundary conditions*: $u = -1$, $v = 0$
- *South boundary conditions*: $u = 0$, $v = 0$
- *East and west boundary conditions*: $u = 0$, $v = 0$

I ran two tests with $Re = 1000$ ($\rho = 1$, $\mu = 1/1000$) and $Re = 5000$ ($\rho = 1$, $\mu = 1/5000$) and let them converge. In table 6.4, the maximum stream function value (ϕ_{max}) is presented with the corresponding vorticity (ω) and position (x, y). Similarly, in table 6.5, the minimum stream function value (ϕ_{min}) is presented.

Assuming that the results for the 1024×1024 grid in [3] are the most accurate, tables 6.4 and 6.5 show that my results for the 128×128 grid are

With SI-units	Dimensionless
ρ	ρ
μ	$\frac{\mu}{Re}$
$\boldsymbol{g}$	$\frac{\boldsymbol{e}_g}{Fr^2}$
σ	$\frac{1}{We}$

Table 6.1: When using dimensionless coefficients, replace the coefficients in the left column with the expressions in the right.

Grid resolution	u_{max}	$\lvert 1.5 - u_{max}\rvert$	Order
18×13	1.49123	0.008772	-
27×19	1.49587	0.004132	1.9837
36×25	1.49761	0.002392	1.9871
54×37	1.49891	0.001094	1.9902
75×51	1.49942	0.000576	1.9923
111×75	1.49973	0.000267	1.9926
150×101	1.49985	0.000147	1.9944

Table 6.2: The maximum outlet velocity, error and empirical order of accuracy relative to the 18×13 grid.

Grid resolution	u_{max}	$\lvert 1.0 - u_{max}\rvert$	Order
18×13	1.00592	0.005917	-
27×19	1.00277	0.002770	2.0000
36×25	1.00160	0.001600	2.0000
54×37	1.00073	0.000730	2.0006
75×51	1.00038	0.000384	2.0085
111×75	1.00018	0.000178	1.9929
150×101	1.00010	0.000098	1.9903

Table 6.3: The maximum outlet velocity, error and empirical order of accuracy relative to the 18×13 grid.

Scheme	Grid	Re	ϕ_{max}	ω	x	y
[3]	1024×1024	1000	0.11892	2.0674	0.46875	0.56543
[3]	128×128	1000	0.11786	2.0508	0.46875	0.56250
present	128×128	1000	0.11751	2.0458	0.46875	0.56250
[3]	1024×1024	5000	0.12193	1.9322	0.48535	0.53516
[3]	128×128	5000	0.11731	1.8595	0.48438	0.53906
present	128×128	5000	0.11506	1.8332	0.48438	0.53906

Table 6.4: Position of maximum stream function value. The values were compared to table 2 and 9 in section 5 in [3].

less accurate than the ones in [3]. Nevertheless, the errors are acceptable and does not indicate any problems with my solver.

I plotted the development of the stream lines over time for $Re = 1000$, and the stream lines and vorticity contours of the steady state solutions for various Re so that the figures can be compared with the ones in [9]. The following contour values were used for the stream function:

```
stream_contours = [-0.1, -0.08, -0.06, -0.04, -0.02,
    -0.01, -3e-3, -1e-3, -3e-4, -1e-4, -3e-5, -1e-5,
    -3e-6, -1e-6, -1e-7, -1e-8, -1e-9, -1e-10, 0.0,
    1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 3e-6, 1e-5, 3e-5,
    1e-4, 3e-4, 1e-3, 3e-3, 0.01, 0.03, 0.05, 0.07,
    0.09, 0.1, 0.11, 0.115, 0.1175]
```

The following contour values were used for the vorticity:

```
vorticity_contours = [-40.0, -35.0, -30.0, -25.0,
    -20.0, -15.0, -10.0, -8.0, -6.0, -4.0, -3.0, -2.0,
    -1.0, -0.5, -0.2, 0.2, 0.5, 1.0, 2.0, 3.0, 4.0,
    6.0, 8.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0]
```

6.4 Backward facing step

The backward facing step test was set up using the following parameters:

- *Domain*: $[0, 15] \times [0, 1]$
- *North and south boundary conditions*: $u = 0$, $v = 0$
- *West boundary conditions*: $u(y) = \max(0, 24(1 - y)(y - 1/2))$, $v = 0$
- *East boundary conditions*: $\frac{\partial u}{\partial x} = 0$, $\frac{\partial v}{\partial x} = 0$

No-slip boundary conditions are imposed on the north, south and lower half of the west boundary. The upper half of the west boundary is an inlet with parabolic velocity profile. Outlet boundary conditions are imposed on the east boundary.

I ran the test with Re in the range 100 to 600 in steps of 100 ($\rho = 1$, $\mu = 1/Re$). I let the test converge before I found the attachment and detachment lengths (see figure 6.9). The streamlines are shown in figure 6.8. The plots have been stretched vertically. The following contour values were used for the stream function:

```
stream_contours = \
    [0.003 * i for i in xrange(-50, 0)] + \
    [-0.001, -0.0003, -0.0001, 0.0, 0.0001, 0.0003,
```

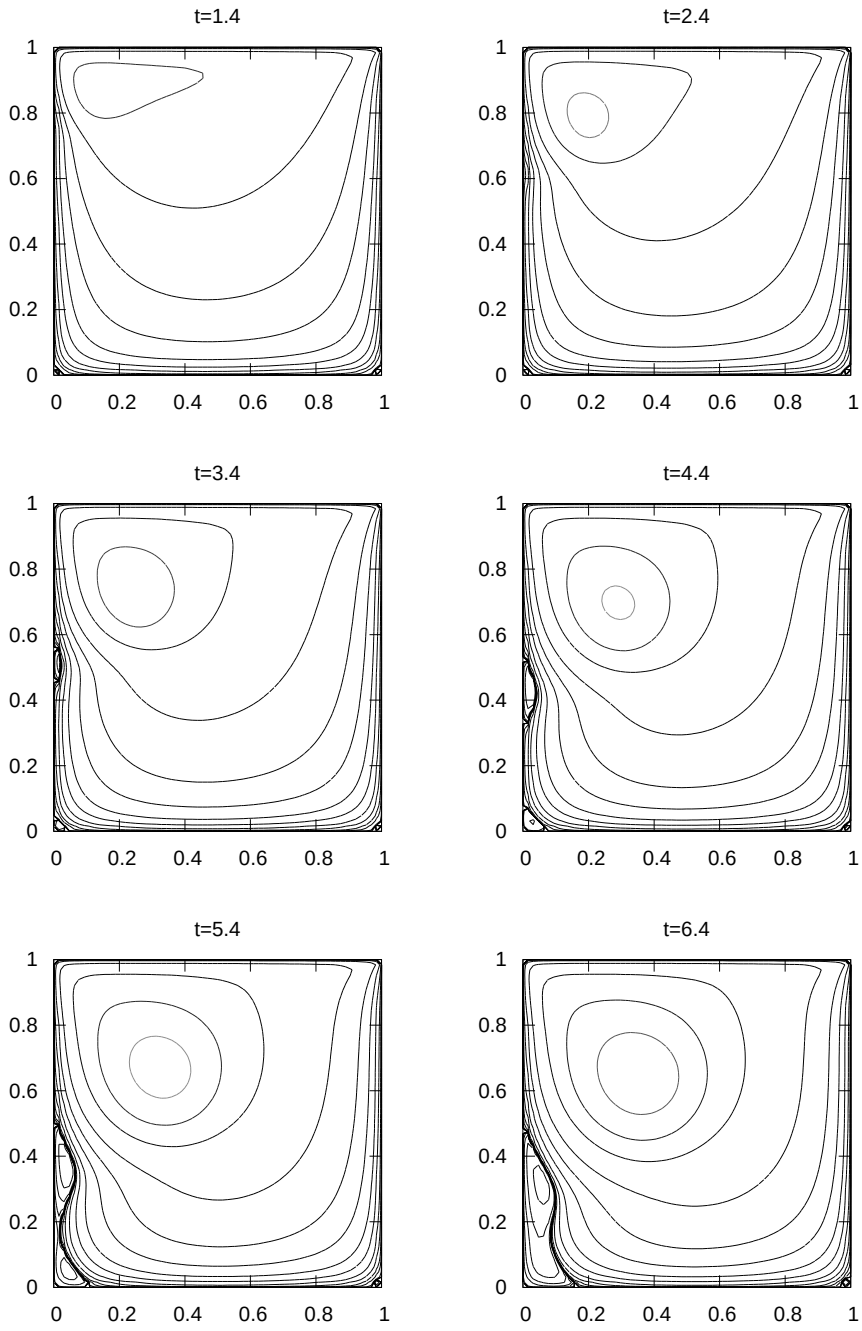


Figure 6.1: Streamlines in square cavity test for $Re = 1000$, 64×64 cells.

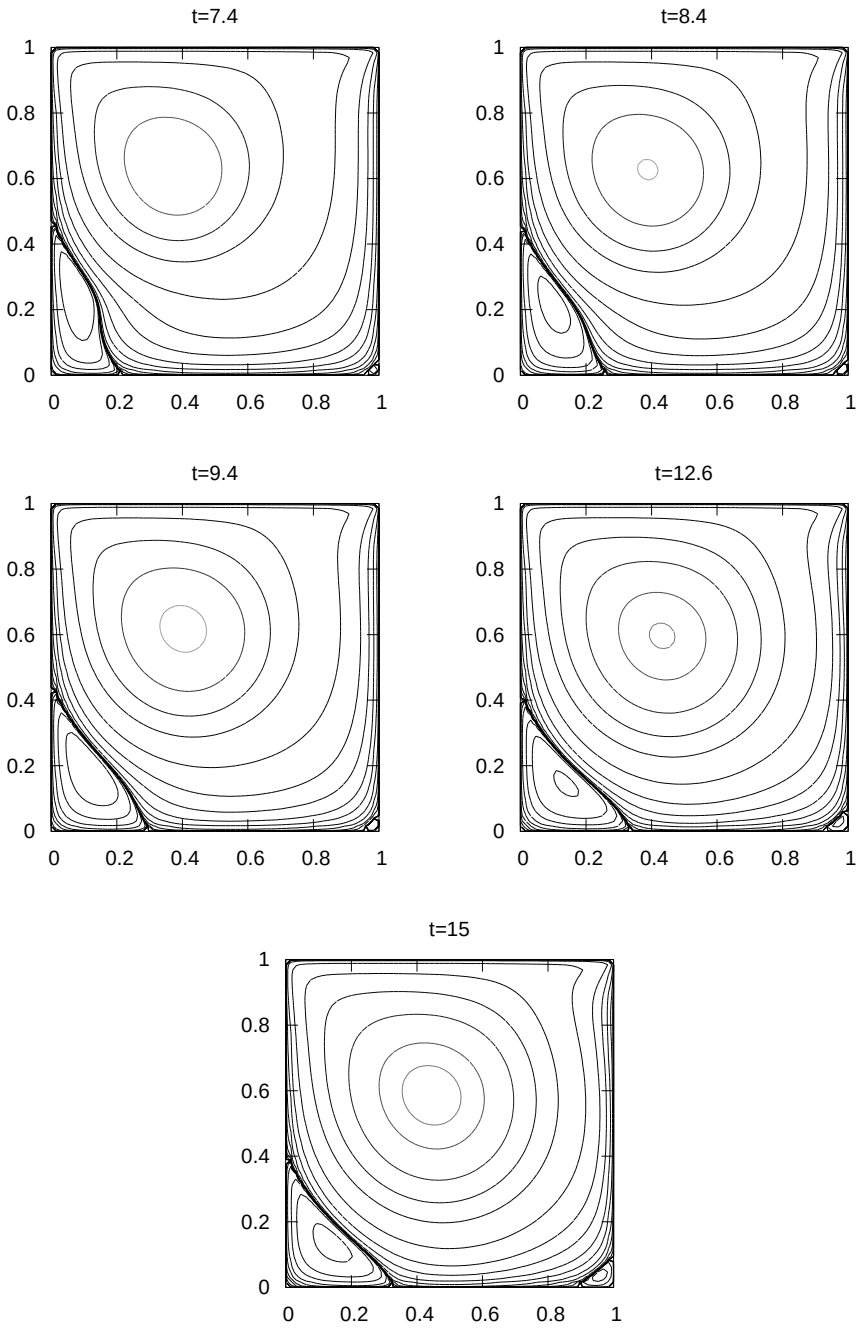
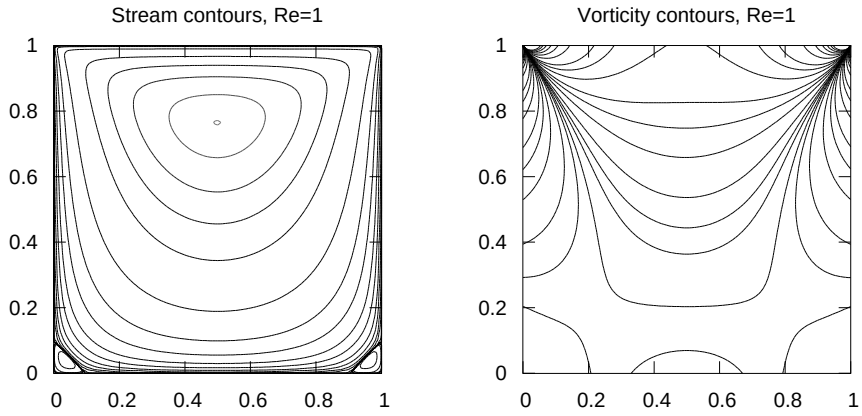


Figure 6.2: Streamlines in square cavity test for $Re = 1000$, 64×64 cells.

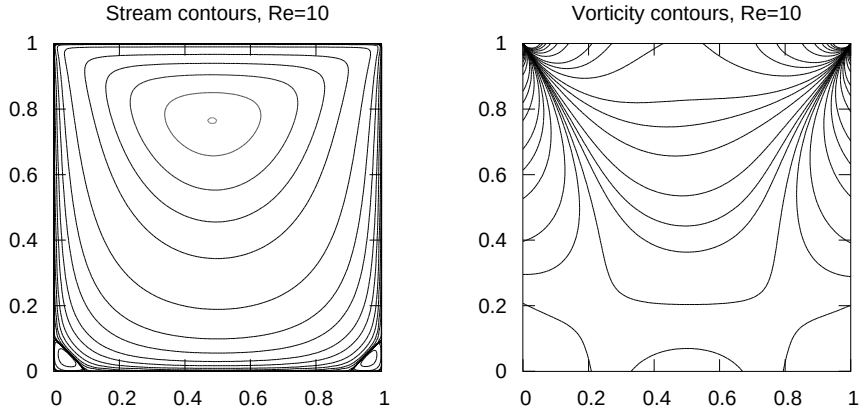
Scheme	Grid	Re	$\frac{\phi_{min}}{10^{-3}}$	ω	x	y
[3]	1024×1024	1000	-1.7292	-1.1120	0.13574	0.11230
[3]	128×128	1000	-1.7003	-1.1304	0.14063	0.10938
present	128×128	1000	-1.7689	-1.1400	0.14063	0.10938
[3]	1024×1024	5000	-3.0694	-2.7245	0.19434	0.073242
[3]	128×128	5000	-2.9313	-2.7718	0.19531	0.070313
present	128×128	5000	-3.4447	-2.9971	0.20313	0.070313

Table 6.5: Position of minimum stream function value. The values were compared to table 3 and 10 in section 5 in [3].



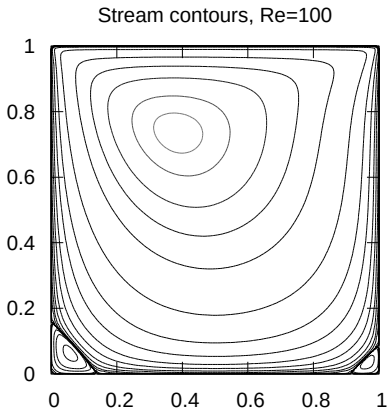
(a) Streamlines (b) Vorticity

Figure 6.3: Square cavity test for $Re = 1$, 128×128 cells.

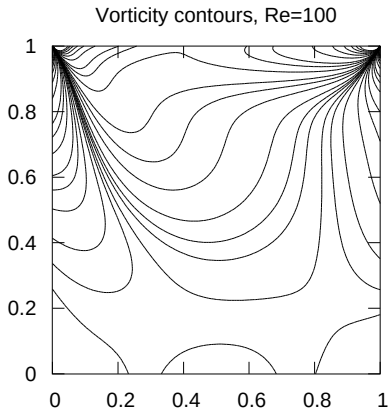


(a) Streamlines (b) Vorticity

Figure 6.4: Square cavity test for $Re = 10$, 128×128 cells.

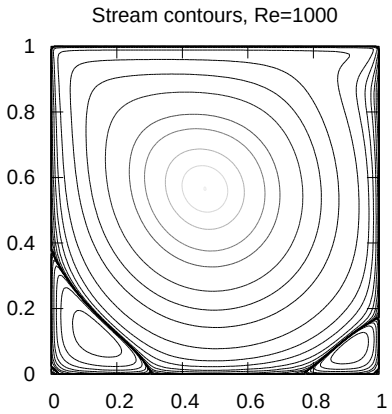


(a) Streamlines

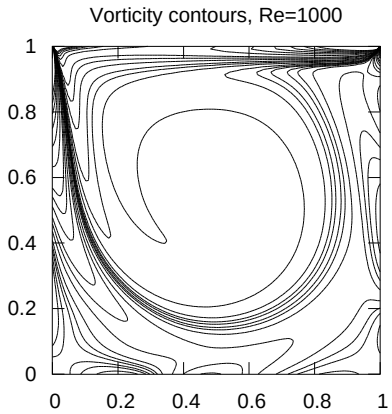


(b) Vorticity

Figure 6.5: Square cavity test for $Re = 100$, 128×128 cells.



(a) Streamlines



(b) Vorticity

Figure 6.6: Square cavity test for $Re = 1000$, 128×128 cells.

```
    0.001, 0.003] + \
    [0.02 * i for i in xrange(1, 25)] + \
    [0.497, 0.499, 0.4997, 0.4999, 0.5, 0.5001] + \
    [0.5 + 0.0003 * i for i in xrange(1, 50)]
```

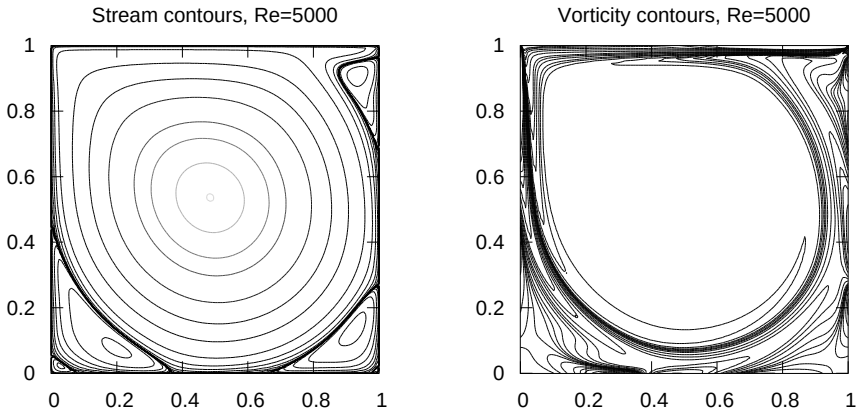
The numbers from [1] listed in table 6.6 were read from a graph and may therefore be a bit inaccurate. In the table, the attachment and detachment lengths are divided by the step height h, which is one half. I found the attachment and detachment lengths by iterating over the upper and lower row of cells in the domain and comparing the u-node values on the west and east side of each cell. If the u-node value on the west side of a cell has a different sign than the one on the east side, the centre of the cell is assumed to be an attachment or detachment point (see figure 6.10 and 6.11). This method is not very accurate. The difference between my solutions and the ones in [1] is a little over 3% for some of the values.

6.5 Rising bubble

I used the same setup for the rising bubble test as in [19]. Since I have not non-dimensionalised Navier-Stokes equations, I had to convert the unitless numbers Re, Fr and We to the equivalent ρ, μ and σ:

- *Domain*: $[0, 2] \times [0, 4]$
- *Boundary conditions*: $u = 0$, $v = 0$
- *Liquid properties*: $\rho = 1$, $\mu = 0.002$
- *Gas properties*: $\rho = 0.0013$, $\mu = 3.2 \cdot 10^{-5}$
- *Surface tension*: $\sigma = 1.46$
- *Gravity*: $\boldsymbol{g} = [0, -5]^T$
- *Curvature estimation method*: DAC with curvature refinement

The bubble is centred at $(1, 1)$ and has radius 0.5. I ran the simulation with grid sizes 25×50, 50×100, 100×200 and 200×400. At $t = 0.5$, I compared the bubble shapes in my simulation (see figures 6.12) with the ones in [19]. All the bubbles in my plots have the same shape, while the bubbles in [19] are more circular at lower resolutions. Only the bubble at the highest resolution (200×400) in [19] looks like the bubbles in my simulation. This is probably because the heaviside function is smeared out in the level-set method described in [19]. As in [19], the velocity of the bubble's mass centre converges to slightly less than 0.9 at $t = 0.5$ as shown in figure 6.13. The graphs in the figure are jagged probably because the calculation of the mass centre is a bit inaccurate.



(a) Streamlines (b) Vorticity

Figure 6.7: Square cavity test for $Re = 5000$, 128×128 cells.

Scheme	Grid	Re	x_1/h	x_2/h	x_3/h
[1]	300×100	100	3.2	-	-
[1]	300×100	200	5.3	-	-
[1]	300×100	300	7.0	-	-
[1]	300×100	400	8.6	8.2	10.2
[1]	300×100	500	9.7	8.5	13.4
[1]	300×100	600	10.7	8.8	16.0
present	300×100	100	3.25	-	-
present	300×100	200	5.35	-	-
present	300×100	300	7.15	-	-
present	300×100	400	8.65	7.95	10.45
present	300×100	500	9.85	8.25	13.55
present	300×100	600	10.75	8.75	16.25

Table 6.6: Backward facing step attachment and detachment lengths.

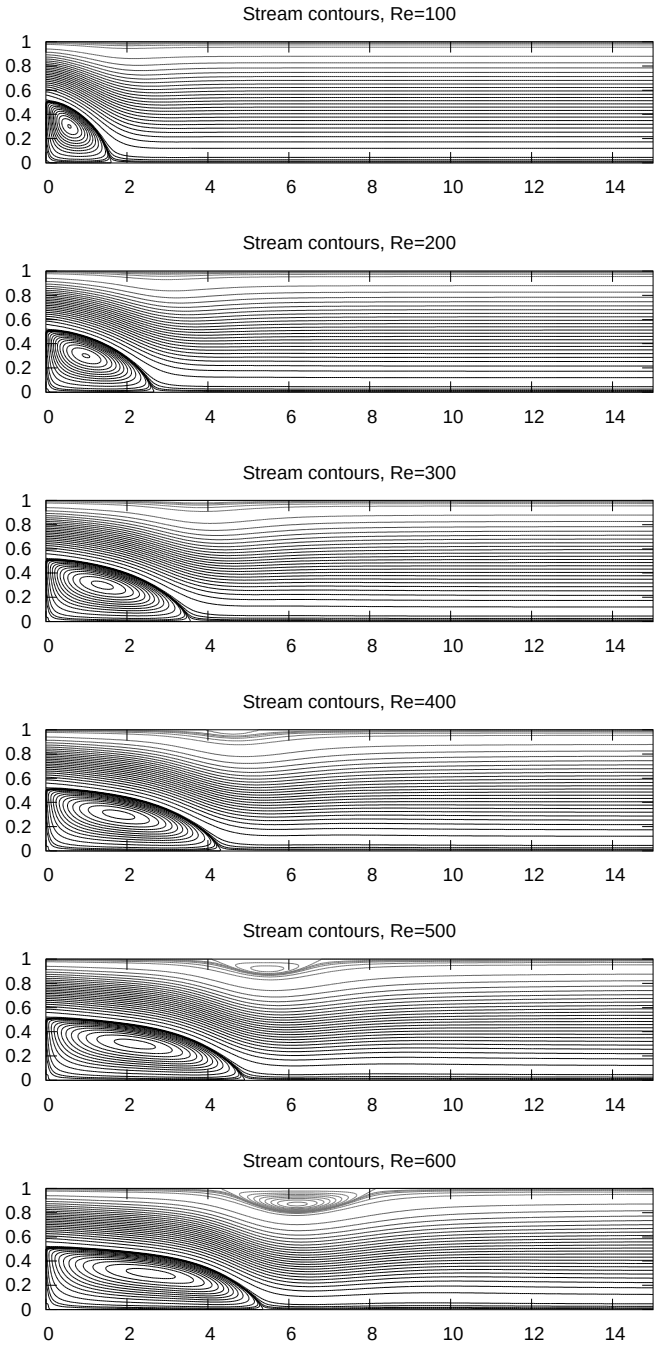


Figure 6.8: Streamlines in the backward facing step test.

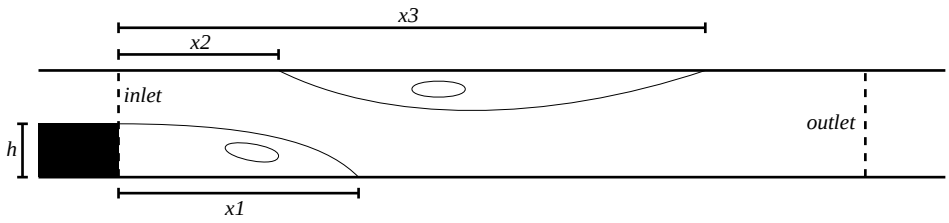


Figure 6.9: Backward facing step geometry. x_1 is the distance from the step to the lower attachment point. x_2 and x_3 are the distances from the step to the upper detachment and attachment point respectively.

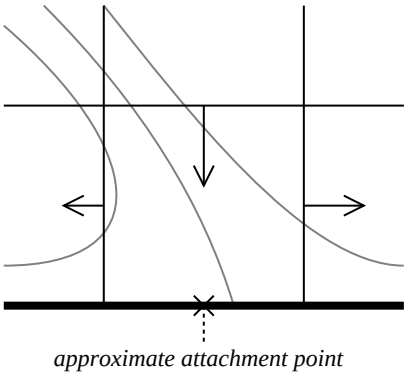


Figure 6.10: Attachment point

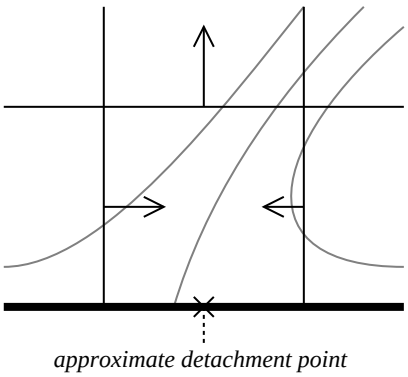
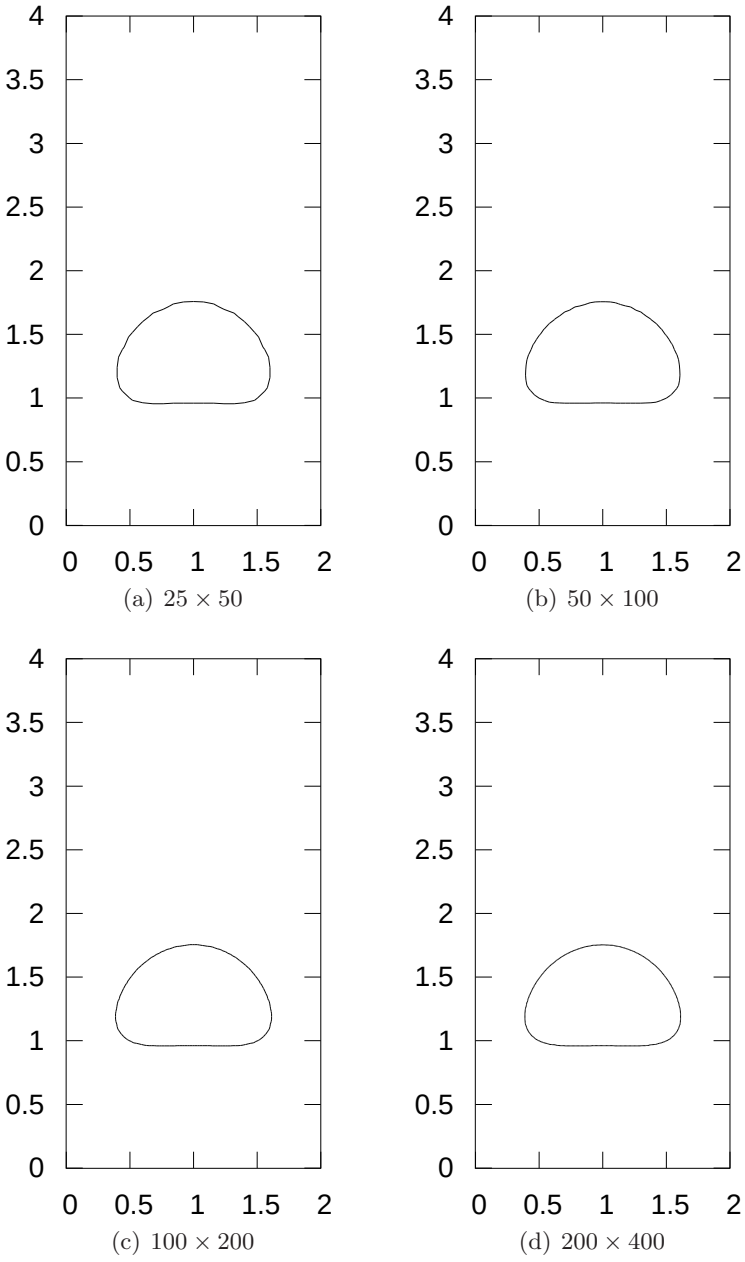


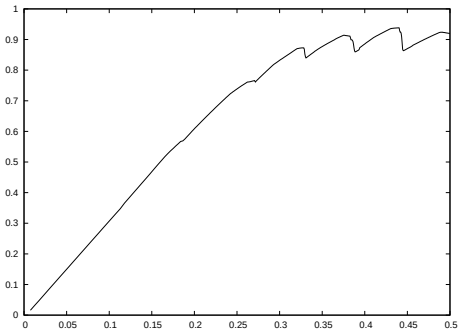
Figure 6.11: Detachment point



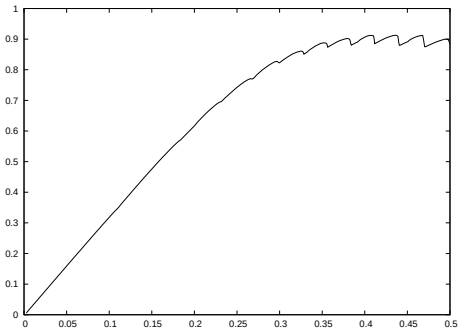
(a) 25×50 (b) 50×100

(c) 100×200 (d) 200×400

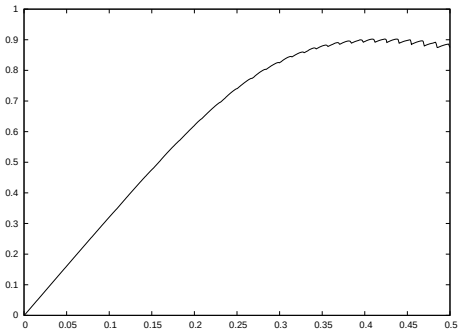
Figure 6.12: Rising bubble at $t = 0.5$ for different grid resolutions.



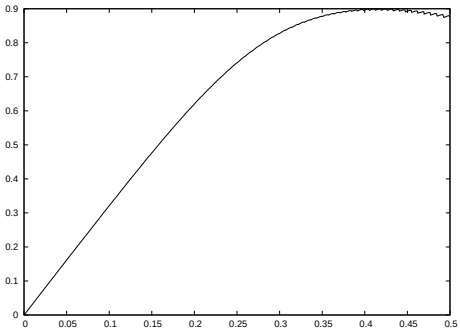
(a) 25×50



(b) 50×100



(c) 100×200



(d) 200×400

Figure 6.13: The speed of the rising bubbles' mass centre over time for different grid resolutions.

6.6 Falling droplet

For the falling droplet test, I used the same setup as in [19]. Since I have not non-dimensionalised Navier-Stokes equations, I had to convert the unitless numbers Re, Fr and We to the equivalent ρ, μ and σ:

- *Domain*: $[0, 6] \times [0, 6]$
- *Grid*: 100×100
- *Boundary conditions*: $u = 0$, $v = 0$
- *Liquid properties*: $\rho = 1$, $\mu = 0.1$
- *Gas properties*: $\rho = 0.0013$, $\mu = 0.0016$
- *Surface tension*: $\sigma = 730$
- *Gravity*: $\boldsymbol{g} = [0, -100]^T$
- *Curvature estimation method*: DAC with curvature refinement

A 2 units deep liquid body is placed at the bottom the domain. A drop of liquid with radius 0.3 is centred at $(3, 4.5)$. The rest of the domain is filled with gas. The interface at different time levels is shown in figures 6.14 and 6.15.

When comparing the figures with [19], one can see that the droplet hits the surface very differently in my simulation. In [19], where the level-set method is used, the droplet attracts the surface and creates a bump on the surface right before hitting it. Thus, the droplet and surface come in contact with each other earlier than what is physically correct. The situation is the opposite in my simulation. It looks like the droplet has difficulties pushing away the gas between the surface and the droplet. When the droplet touches the surface, the interface looks rather jagged and unnatural, possibly because the histograms used in the ELVIRA algorithm are poor approximations of the interface in this case. The wave propagation afterwards looks a bit delayed compared to [19].

6.7 Static drop

The static drop test is commonly used to verify surface tension implementations. According to Navier-Stokes equations, the surface tension and pressure should be in perfect balance, and the drop should be in equilibrium. However, numerical inaccuracies often lead to non-physical currents near the drop surface. The static drop test is used to measure and visualise the spurious currents.

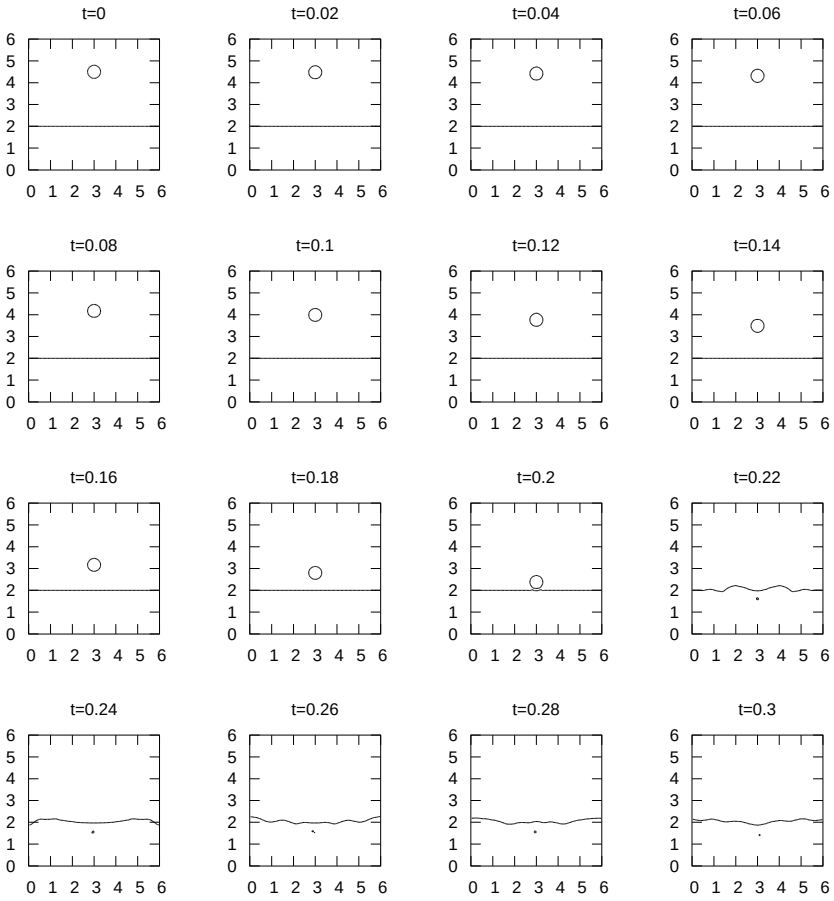


Figure 6.14: Falling droplet at different time levels.

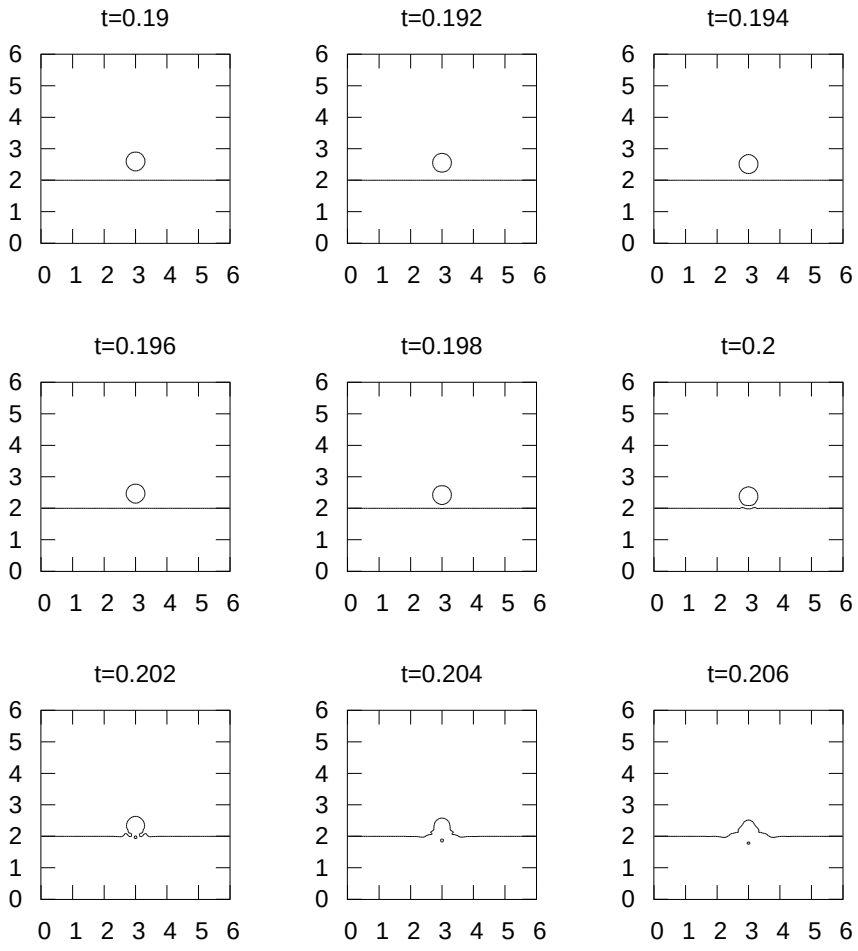


Figure 6.15: Falling droplet at different time levels, close to the surface.

I chose to set up the test almost as in [23, 16]. Since I simulate two-dimensional drops, while three-dimensional drops are simulated in the articles, the results are not directly comparable.

I set up a circular drop of liquid in zero gravity in the middle of a closed box with gas. I used the following parameters:

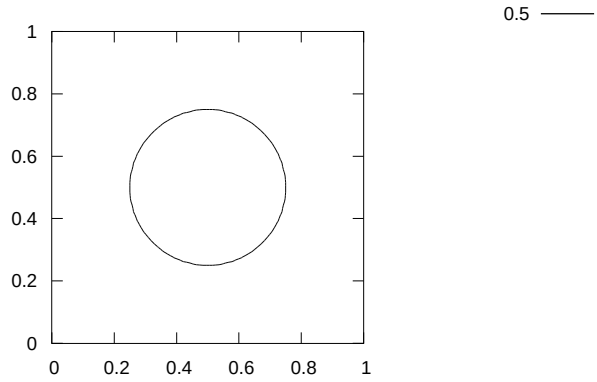
- *Domain*: $[0, 1] \times [0, 1]$
- *Grid*: 96×96
- *Drop radius*: $r = 0.25$
- *Boundary conditions*: $u = 0$, $v = 0$
- *Liquid properties*: $\mu = 1$, $\rho = 4$
- *Gas properties*: $\mu = 1$, $\rho = 4$
- *Surface tension*: $\sigma = 0.357$
- *Time step*: $\Delta t = 10^{-5}$

I let the simulation run until $t = 200\Delta t$ for each of the three curvature estimation methods CSF, DAC and DAC with curvature refinement. The pressure jump at the drop surface is expected to be $\sigma/r = 1.428$. The results are shown in tables 6.7 and 6.8. The table shows that CSF is far less accurate than DAC, which in turn is less accurate than DAC with curvature refinement. The spurious currents become negligible when using DAC with curvature refinement. The interface, pressure, curvature and velocity fields are shown in figures 6.16 to 6.21. Since the curvature is only defined for interface cells, the curvature in figure 6.18 is set to the expected value -4 everywhere else to avoid disturbing contour lines. Note that the arrow lengths are not comparable between the figures 6.19, 6.20 and 6.21.

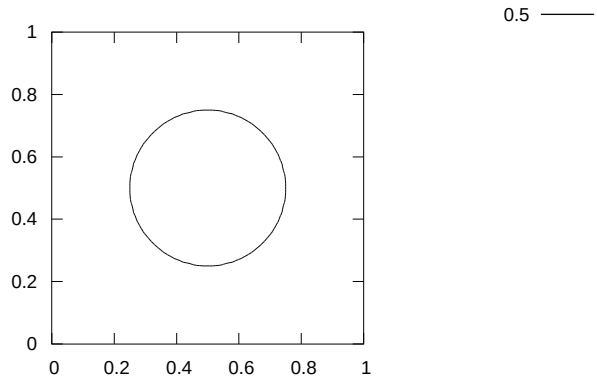
6.8 Rayleigh-Taylor instability

I set up the Rayleigh-Taylor instability test as described in [20], but with higher grid resolution:

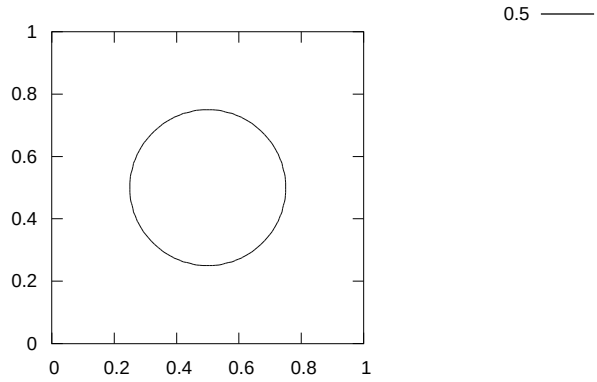
- *Domain*: $[0, 1] \times [0, 4]$
- *Grid*: 128×512
- *North and south boundary conditions*: $u = 0$, $v = 0$
- *East and west boundary conditions*: $u = 0$, $\frac{\partial v}{\partial x} = 0$
- *Upper fluid properties*: $\mu = 0.00313$, $\rho = 1.225$



(a) CSF

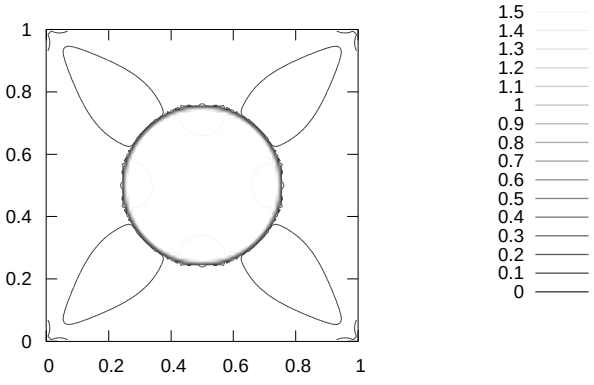


(b) DAC

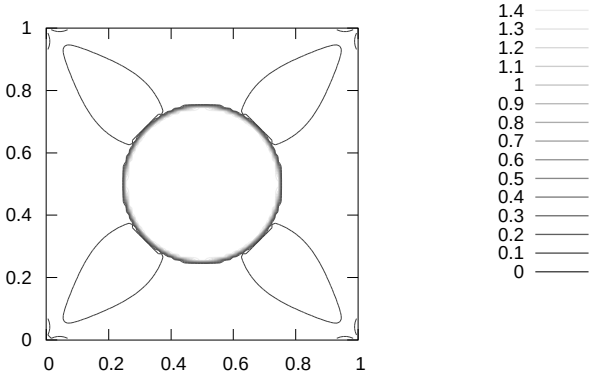


(c) DAC w/refinement

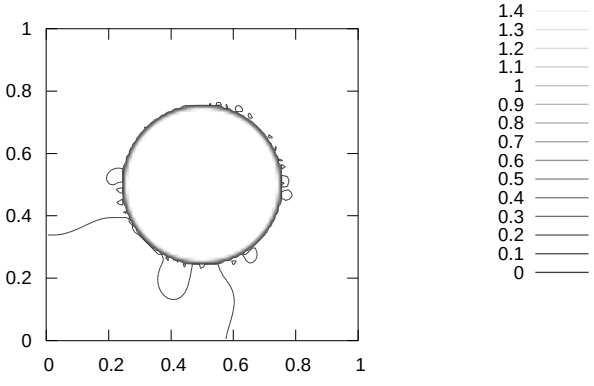
Figure 6.16: Static drop interface after 200 time steps.



(a) CSF

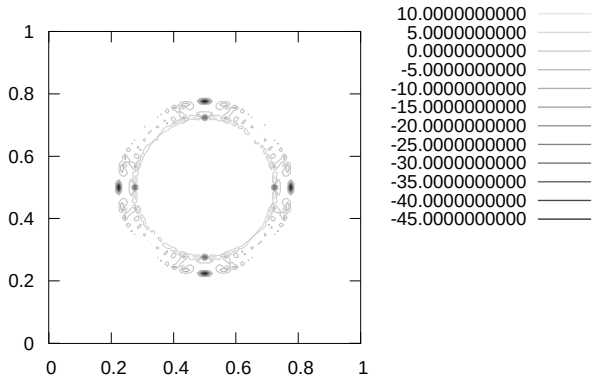


(b) DAC

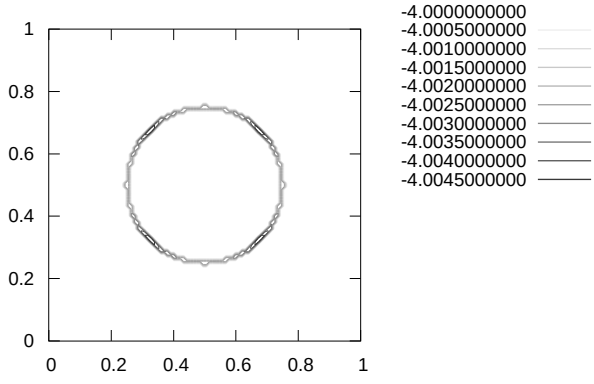


(c) DAC w/refinement

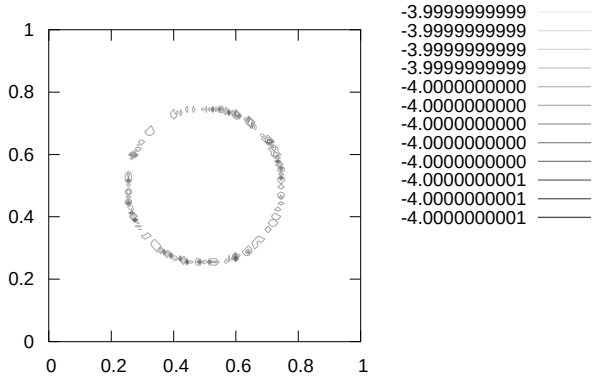
Figure 6.17: Static drop pressure after 200 time steps.



(a) CSF



(b) DAC



(c) DAC w/refinement

Figure 6.18: Static drop curvature after 200 time steps.

Method	Max. vel. at $t = \Delta t$	Max. vel. at $t = 200\Delta t$
CSF	$5.29359 \cdot 10^{-4}$	$1.33812 \cdot 10^{-3}$
DAC	$1.61074 \cdot 10^{-7}$	$1.43286 \cdot 10^{-6}$
DAC w/refinement	$3.55932 \cdot 10^{-14}$	$6.13064 \cdot 10^{-14}$

Table 6.7: Maximum velocity in the static drop test.

Method	Pressure at $t = 200\Delta t$	Rel. error
CSF	1.51251	5.92%
DAC	1.42920	0.84%
DAC w/refinement	1.42800	0.00%

Table 6.8: Pressure difference and pressure error in the static drop test.

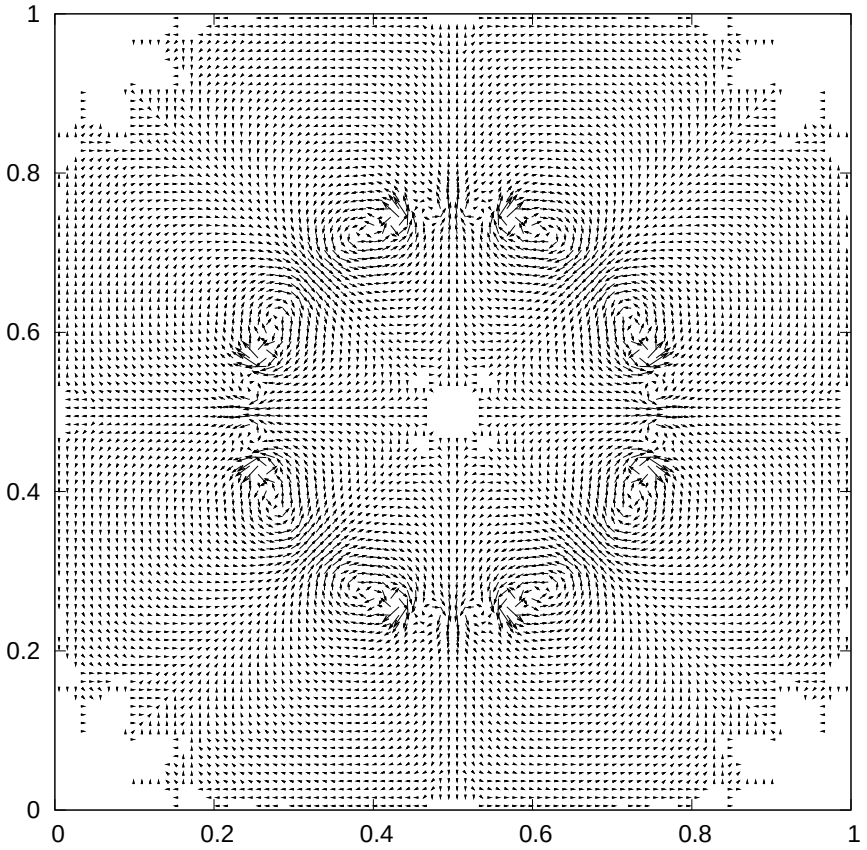


Figure 6.19: Static drop velocity after 200 time steps for CSF.

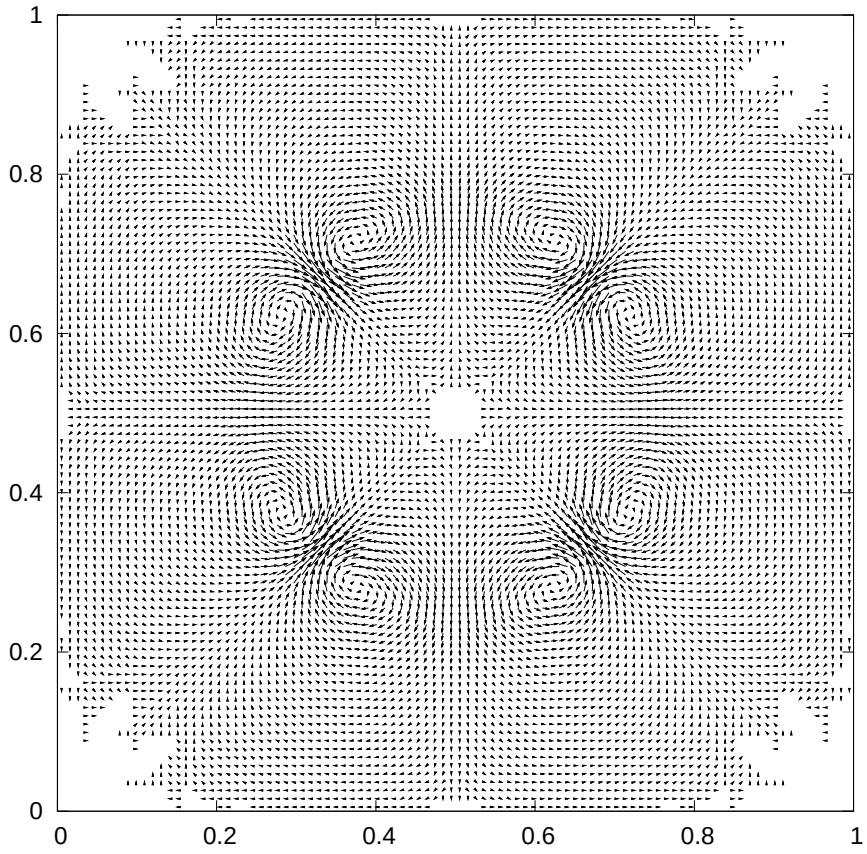


Figure 6.20: Static drop velocity after 200 time steps for DAC.

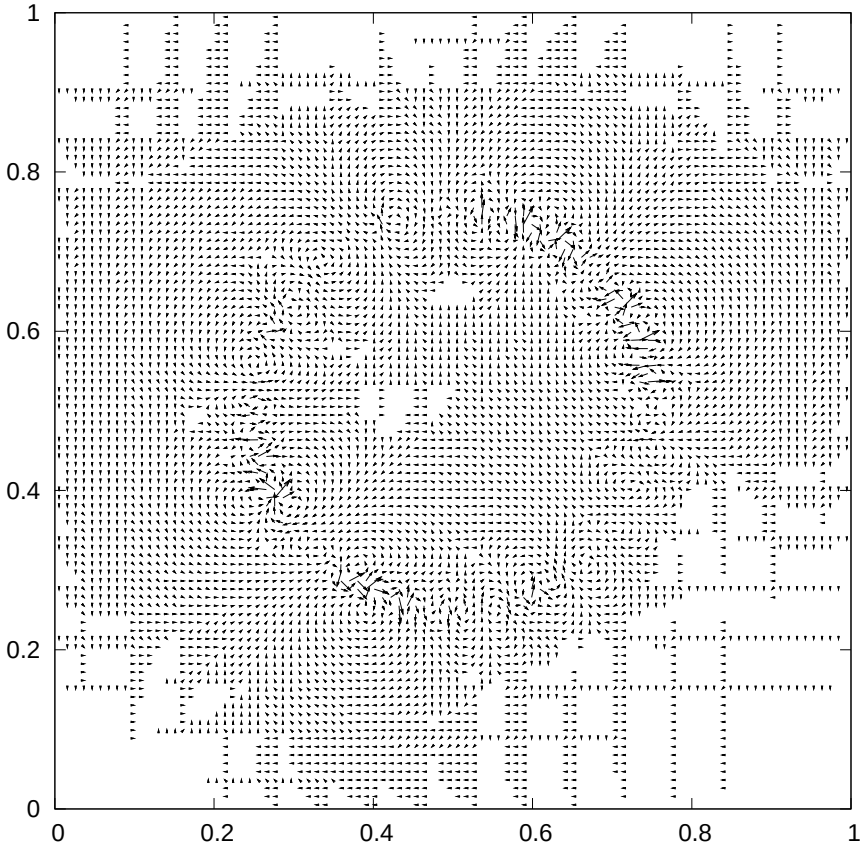


Figure 6.21: Static drop velocity after 200 time steps for DAC with refinement.

- *Lower fluid properties*: $\mu = 0.00313$, $\rho = 0.1694$
- *Interface function*: $h(x) = 2 + 0.05\cos(2\pi x)$
- *Gravity*: $\boldsymbol{g} = [0, -9.81]^T$

The boundary conditions and the gravity were not specifically given in [20], so I used parameters that seemed reasonable.

I plotted the interface at $t = 0$, $t = 0.7$, $t = 0.8$ and $t = 0.9$ (see figure 6.22) and compared with the plots in [20]. The plots look as expected.

6.9 Pressure Poisson equation

I set up a test to see if the PPE was solved properly. Since the `poisson` module was made to be independent of the number of dimensions, I set up a 3D problem in Python:

```
n = (20, 30, 40)
f = array(range(n[0] * n[1] * n[2]), float)
f.shape = n
delta = (0.001, 0.002, 0.003)
mask = ones(n, int)
remove_singularity(mask)
f[1:-1,1:-1,1:-1] -= f[1:-1,1:-1,1:-1].mean()
rho = f.copy() % 13 + 1
phi = zeros(n,float)
poisson(phi, f, delta, mask, rho)
```

The L^2-norm of the residual ($5.10662 \cdot 10^{-7}$) was insignificant relative to the L^2 norm of `f` ($8.62747 \cdot 10^5$), which indicates that the equation was solved rather accurately.

6.10 Zalesak's rotating disc

Zalesak's rotating disc test will show how well the volume-of-fluid reconstruction and advection algorithms work. The test was set up using the following parameters:

- *Domain*: $[-1, 1] \times [-1, 1]$
- *Initial and boundary conditions*: $u = -y$, $v = x$

A slotted disc with radius 0.3 is placed with its centre in $(0.0, 0.5)$. A rectangle $[-0.05, 0.05] \times [0.2, 0.7]$ is subtracted from the disc to create the slot. This setup is equivalent to that described in [32]. The disc is rotated through one revolution, that is, from $t = 0$ to $t = 2\pi$. I ran the test with

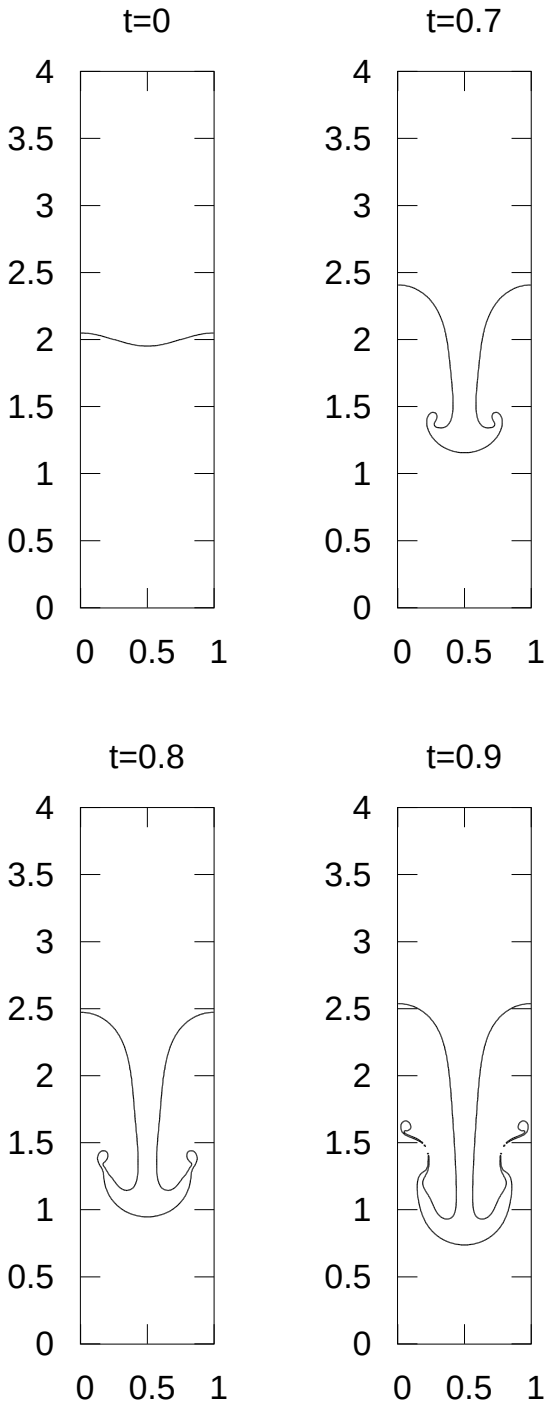


Figure 6.22: Rayleigh-Taylor instability at different time levels.

50×50, 100×100, 150×150 and 200×200 cells. I made plots (see figures 6.23, 6.24, 6.25 and 6.26) of the disc right after initialisation and after one revolution.

As expected, there is no flotsam[1] in the plots, and the shape is acceptably well preserved. Since the reconstruction algorithm expects the interface to be smooth, it is not surprising that the corners are rounded.

Even though I use VOF while a level-set variant is used in [32], my plots are visually indistinguishable from those in [32] for the 100×100, 150×150 and 200×200 grids. For 50×50, my plot looks better.

When comparing my plot for 100×100 with the figure in [13] corresponding to ELVIRA and operator split advection, my plot looks slightly less accurate. The details near the corners are also different. I do not know why this is the case, since the setup and methods used should be the same.

In [11], the disc has a shallower slot, but when comparing only the corners, my plots resemble those in the article. Both I and [11] use the VOF method, but the reconstruction and advection methods are different.

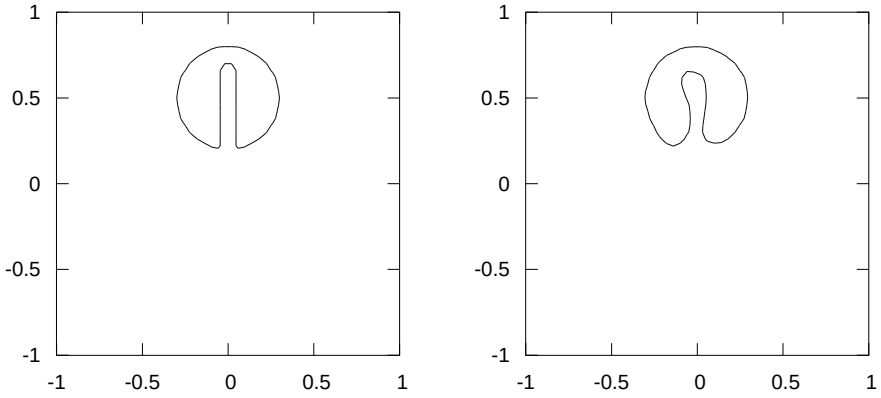
6.11 Profiling

I used the `cProfile` module in Python to profile the rising bubble test. I removed all plotting and printing calls so that only the calculations were timed. I ran the test until $t = 0.1$ at different grid resolutions on an Intel Core 2 Duo T7200 CPU at 2.0GHz with DDR2 RAM at 533MHz.

If n is the number of grid cells, then one can expect the complexity of most parts of the Navier-Stokes solver to be $O(n)$, since the solver uses an explicit scheme. The curvature estimation is only needed at the interface, which is 1D. Thus the curvature estimation should have complexity $O(\sqrt{n})$. Because the number of elements in the PPE matrix is of order $O(n)$, a plain conjugate gradient algorithm with complexity $O(n^2)$ should be able to solve the PPE accurately. Since I use the SuperLU module, I expect the complexity to be somewhat better and lie between $O(n)$ and $O(n^2)$. The PPE solver's high complexity should make it more dominant at higher grid resolutions.

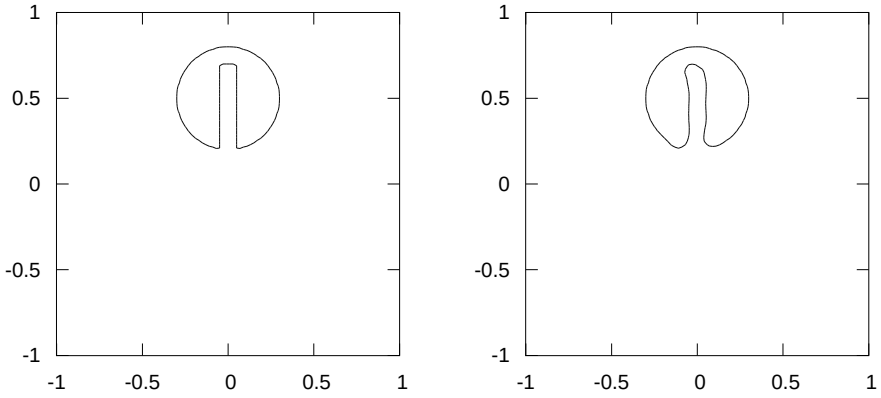
The most time consuming parts turned out to be the PPE solver, VOF advection and calculating the tentative velocity. DAC with curvature refinement is a part of calculating the tentative velocity and takes considerable time of its own. I have listed how much time was spent in each of the parts in table 6.9. I also timed the simulation without profiling to see how large the profiling overhead was. The time difference turned out to be insignificant. The table shows that the PPE solver quickly becomes the most time consuming part of the Navier-Stokes solver as expected. The time spent per iteration increase with higher grid resolutions at a lower rate than expected

[1] Flotsam is particles that break off during advection due to numerical errors.



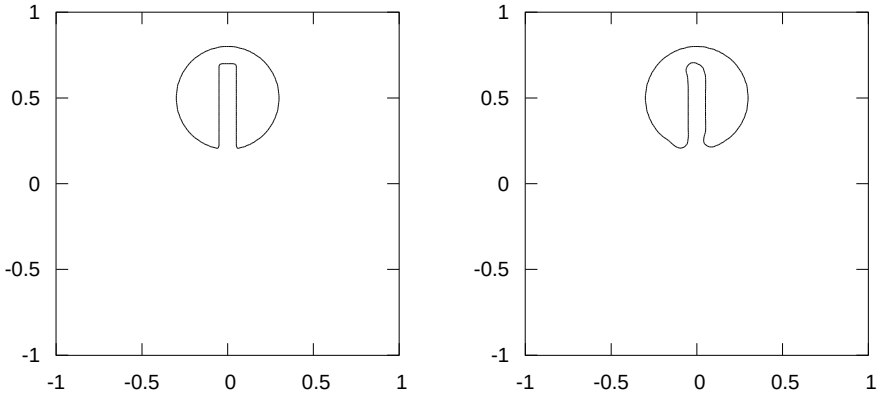
(a) Initial state (b) State after one revolution

Figure 6.23: Zalesak's rotating disc in a 50×50 grid.



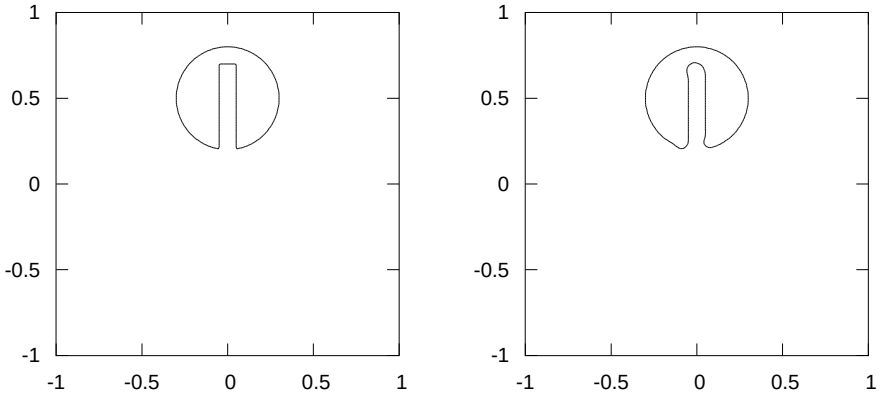
(a) Initial state (b) State after one revolution

Figure 6.24: Zalesak's rotating disc in a 100×100 grid.



(a) Initial state (b) State after one revolution

Figure 6.25: Zalesak's rotating disc in a 150×150 grid.



(a) Initial state (b) State after one revolution

Figure 6.26: Zalesak's rotating disc in a 200×200 grid.

up to resolution 100×200. The reason may be some constant overhead which becomes less and less significant as each iteration takes longer to execute. The time spent per iteration increase with a higher rate than expected at higher resolutions.

Resolution	Part	Time in sec.	Time in %	Time/it. in ms
25×50	Advect	0.017	4.93	0.85
(20 iterations)	$\boldsymbol{u}^*$	0.141	40.87	7.05
	(Curvature)	(0.077)	(22.32)	(3.85)
	Poisson	0.108	31.30	5.40
	Other	0.079	22.90	3.95
	Total	0.345	100.00	17.25
	W/o profiling	0.347	-	-
50×100	Advect	0.141	4.19	2.47
(57 iterations)	$\boldsymbol{u}^*$	0.871	25.87	15.28
	(Curvature)	(0.485)	(14.40)	(8.51)
	Poisson	1.909	56.70	33.49
	Other	0.446	13.25	7.82
	Total	3.367	100.00	59.07
	W/o profiling	3.393	-	-
100×200	Advect	1.231	2.88	7.69
(160 iterations)	$\boldsymbol{u}^*$	6.089	14.25	38.06
	(Curvature)	(2.865)	(6.70)	(17.91)
	Poisson	32.350	75.70	202.19
	Other	3.062	7.17	19.14
	Total	42.732	100.00	267.08
	W/o profiling	44.64	-	-
150×300	Advect	5.514	2.42	18.81
(293 iterations)	$\boldsymbol{u}^*$	20.075	8.80	68.52
	(Curvature)	(8.665)	(3.80)	(29.57)
	Poisson	190.925	83.72	651.62
	Other	11.551	5.06	39.42
	Total	228.065	100.00	778.38
	W/o profiling	227.029	-	-
200×400	Advect	17.191	2.02	38.12
(451 iterations)	$\boldsymbol{u}^*$	99.727	11.71	221.12
	(Curvature)	(19.936)	(2.34)	(44.20)
	Poisson	686.754	80.62	1522.74
	Other	48.174	5.66	106.82
	Total	851.846	100.00	1888.79
	W/o profiling	853.309	-	-
250×500	Advect	40.186	1.73	63.79
(630 iterations)	$\boldsymbol{u}^*$	224.480	9.67	356.32
	(Curvature)	(38.399)	(1.65)	(60.95)
	Poisson	1941.955	83.68	3082.47
	Other	114.110	4.92	181.13
	Total	2320.731	100.00	3683.70
	W/o profiling	2317.93	-	-

Table 6.9: Profiling results.

Chapter 7

Tutorial

I have implemented the following Python modules:

- `NavierStokes`: The main simulation module.
- `poisson`: Helper module for solving the PPE.
- `vof`: Helper module for volume-of-fluid related tasks such as interface reconstruction and advection, volume fraction calculations and calculating surface tension using DAC.
- `csf`: Helper module for smoothing scalar fields and calculating surface tension using CSF.
- `Keyboard`: Module for reading keys without blocking the program.

A user will only need `NavierStokes` and possibly `Keyboard`. The modules are included in a package called `kmkns`.

7.1 Installation

Before installing the `kmkns` Python package, the following programs, packages and modules must be installed.

- Python: `http://www.python.org/`
- NumPy: `http://numpy.scipy.org/`
- Gnuplot: `http://www.gnuplot.info/`
- Gnuplot.py: `http://gnuplot-py.sourceforge.net/`
- SciTools: `http://code.google.com/p/scitools/`

To install `kmkns`, perform the following steps:

1. Download "kmkns-0.1.tar.gz" from

   `http://code.google.com/p/kmkns/downloads/list`

2. Unpack the archive. In the directory where "kmkns-0.1.tar.gz" is located, type:

   `tar xvf kmkns-0.1.tar.gz`

3. Change to the unpacked directory by typing:

   `cd kmkns-0.1`

4. Install the package by typing:

   `python setup.py install`

   Or possibly:

   `sudo python setup.py install`

7.2 Hello, World!

Start a Python script by importing the necessary modules:

```
from kmkns.NavierStokes import *
import kmkns.Keyboard
```

Whether one is using `import` or `from ... import *` is not important. `NavierStokes` introduces two classes: `Grid` and `NavierStokes2D`. First, create a grid object which defines the computational domain and the grid resolution. For instance, this line will create a 3×1 domain with the origin in the lower left corner, and with 90×30 cells:

```
grid = Grid([0.0, 0.0], [3.0, 1.0], [90, 30])
```

Create a Navier-Stokes solver object. Pass fluid properties like viscosity μ and density ρ to the constructor:

```
ns = NavierStokes2D(grid, mu=1.0, rho=1.0)
```

The next step is to define initial and boundary conditions. The default initial condition is zero pressure and velocity everywhere. This is sufficient for most cases. Remember to set boundary conditions for all boundaries, or else the behaviour is undefined. This will set up a channel with walls at the top and bottom, inlet to the left and outlet to the right:

```
ns.set_bc('north+south', u=0, v=0)
ns.set_bc('west', u=1, v=0)
ns.set_bc('east', dudn=0, dvdn=0)
```

It is useful to give commands, such as "quit" or "plot" without blocking the program. This line will prevent the program from blocking when trying to read keys:

```
kmkns.Keyboard.set_poll()
```

Everything is set up, and the main loop can begin:

```
while True:
    ch = kmkns.Keyboard.get_key()
    if ch == 'q' or ch == 'Q':
        break
    ns.step(ns.find_suitable_dt())
    if ch == 'p' or ch == 'P':
        ns.plot_velocity()
```

Though not really needed, one can reset to blocking mode with this line after exiting the main loop:

```
kmkns.Keyboard.set_normal()
```

7.3 How do you do, Tellus!

A more advanced example involves two fluids. Begin the same way as for one fluid:

```
from kmkns.NavierStokes import *
import kmkns.Keyboard
grid = Grid([0.0, 0.0], [1.0, 1.0], [32, 32])
```

When creating the Navier-Stokes solver object, additional fluid information is needed, such as viscosity μ and density ρ for both fluids, gravity and possibly surface tension:

```
ns = NavierStokes2D(grid, mu_liquid=1.0, mu_gas=1e-2,
    rho_liquid=1e3, rho_gas=1.0,
    surface_tension_coeff=0.0, gravity=[0.0, -100.0])
```

By default, all of the fluid is liquid. In this example, set wall boundary conditions and add a drop of liquid in a box of gas:

```
ns.set_bc('north+south+west+east', u=0, v=0)
ns.add_gas([0.0, 0.0], [1.0, 1.0])
```

```
ns.add_drop([0.5, 0.5], 0.25)
```

The rest is pretty much the same as in the previous example:

```
kmkns.Keyboard.set_poll()

while True:
    ch = kmkns.Keyboard.get_key()
    if ch == 'q' or ch == 'Q':
        break
    ns.step(ns.find_suitable_dt())
    if ch == 'p' or ch == 'P':
        ns.plot_level([0.5])
        print 't=%g' % ns.t

kmkns.Keyboard.set_normal()
```

7.4 Fluid properties

The following fluid properties can be set in the call to the `NavierStokes2D` constructor:

- Density ρ for each fluid.
- (Dynamic) viscosity μ or kinematic viscosity ν for each fluid.
- Surface tension coefficient σ.

In addition, the gravitational acceleration vector $\boldsymbol{g}$, the method for curvature estimation and the method for outflow correction can be specified. The next example will show how to set all properties:

```
ns = NavierStokes2D(grid, mu_liquid=1.0, mu_gas=1e-2,
    rho_liquid=1e3, rho_gas=1.0,
    surface_tension_coeff=0.0, gravity=[0.0, -100.0],
    curv_method='dac', mass_conservation='scale')
```

Note that "liquid" and "gas" does not mean that one fluid must be liquid and the other gas. These are just names used to distinguish the two fluids. It could just as well have been "dark" and "light" or "primary" and "secondary".

`curv_method` indicates the method used for calculating the curvature:

- `'csf'`: Use continuum surface force method.
- `'dac'`: Use direction averaged curvature method.

- `'mdac'`: Use DAC with curvature refinement.

`mass_conservation` indicates the method for outflow correction used to achieve global mass conservation:

- `'scale'`: Scale the outflow by a constant factor.
- `'add'`: Add a constant to the outflow.
- `'ignore'`: Do not change the outflow.

7.5 Initial and boundary conditions

The initial conditions are set with a call to `set_ic`, and boundary conditions with a call to `set_bc`.

The `set_ic` method expects keyword arguments where the keyword must be `u`, `v`, `p` or `c`. `u`, `v`, `p` and `c` indicate the horizontal velocity, vertical velocity, pressure and colour function respectively. The argument value may be a scalar or a function with two arguments x and y. If it is a scalar, the entire field is set to this value. If it is a function, the value at each node is set equal to the function value at the node's location. For instance, this will set the initial velocity to a spiral:

```
ns.set_ic(u=lambda x, y: -y, v=lambda x, y: x)
```

The default initial condition is zero everywhere.

The `set_bc` method expects one string and keyword arguments. The string can be "east", "west", "north", "south" or a combination of these. It indicates which boundary or boundaries to set. When combining more boundaries, put a plus sign between them, for example: "north+south". The keywords of the keyword arguments can be the following:

- `u`: The horizontal velocity can be set to a scalar value or a function of x, y and t (space and time).
- `v`: The vertical velocity can be set to a scalar value or a function of x, y and t (space and time).
- `p`: The pressure can be set to a scalar value or a function of x, y and t (space and time).
- `c`: The colour function can be set to a scalar value or a function of x, y and t (space and time).
- `dudn`: The normal derivative of the horizontal velocity can only be set to zero.

	u	dudn	p	v	dvdn
Pressure	✓		✓		
Pressure		✓	✓		
No-slip/inlet	✓			✓	
Symmetry/inlet		✓		✓	
Outlet	✓				✓
Outlet		✓			✓

Table 7.1: Valid combination of boundary conditions and their meaning for the *north* and *south* boundary.

	u	dudn	p	v	dvdn
No-slip/inlet	✓			✓	
Symmetry/inlet	✓				✓
Outlet		✓		✓	
Outlet		✓			✓
Pressure			✓	✓	
Pressure			✓		✓

Table 7.2: Valid combination of boundary conditions and their meaning for the *east* and *west* boundary.

- `dvdn`: The normal derivative of the vertical velocity can only be set to zero.

Exactly two velocity and pressure boundary conditions should be given on each boundary. In addition, a Dirichlet boundary condition for the colour function may be given. Note that the homogeneous Neumann boundary condition for the colour function is applied if a Dirichlet boundary condition is not specified. Table 7.1 and 7.2 show which velocity and pressure boundary conditions can be combined.

The following example will set a parabolic inlet to the west, an outlet to the east and walls to the north and south:

```
ns.set_bc('north+south', u=0, v=0)
ns.set_bc('west', u=lambda x, y, t: y*(1-y), dvdn=0)
ns.set_bc('east', dvdn=0, dudn=0)
```

7.6 Two-phase flow

Four methods are provided for setting up the fluid domains. By default, all of the fluid is liquid. `add_gas` converts all liquid within a rectangle to gas. The rectangle is defined by the lower left and upper right corners. `add_bubble` converts all liquid within a circle to gas. The circle is defined

by its centre and radius. Similarly, `add_liquid` and `add_drop` convert gas to liquid within a rectangle and circle respectively. For instance, the following lines will set up Zalesak's disc by adding a circle and subtracting a rectangular slot:

```
ns.add_bubble([0, 0.5], 0.3)
ns.add_liquid([-0.05, -1.0], [0.05, 0.7])
```

The methods will not affect obstacles. The type of fluid flowing into the domain at inlets is defined by the colour function boundary condition.

7.7 Obstacles

Each cell in the domain interior can be either a fluid cell or an obstacle cell. By default, all cells are fluid cells. Rectangular obstacles can be added by calling `add_obstacle` with the lower left and upper right corner of the rectangle as arguments. `add_obstacle` will mark all cells whose centre is within the rectangle as obstacle cells. The rectangle must not extend beyond the computational domain. Example:

```
ns.add_obstacle([0.0, 0.0], [7.5, 0.75])
```

7.8 Tracer fluid

For more interesting visualisation of the flow, tracer fluid can be added. The tracer fluid has no effect on the simulation, but is advected with the flow without diffusion. Add a rectangular block of tracer fluid by calling `add_tracer` with the rectangle's lower left and upper right corners as arguments. Tracer fluid can also be added to the ghost cells at inlets to create a tracer fluid source. The next example initialises some stripes of tracer fluid:

```
ns.add_tracer([-1.0, 0.5], [0.0, 0.55])
ns.add_tracer([-1.0, 0.6], [0.0, 0.65])
ns.add_tracer([-1.0, 0.7], [0.0, 0.75])
ns.add_tracer([-1.0, 0.8], [0.0, 0.85])
ns.add_tracer([-1.0, 0.9], [0.0, 0.95])
```

7.9 Loading and saving

Since simulations can take forever, it is convenient to be able to save the state of the simulation and continue at a later time. The methods `save` and `load` implement this functionality. `save` only saves variables that change over time, such as the velocity and pressure fields and the iteration counter.

The grid, boundary conditions and obstacles must be set up manually. `load` will not raise an exception if the file does not exist, but will instead return `False`. If the file was successfully loaded, `True` is returned. The usage is illustrated in the following example:

```
from kmkns.NavierStokes import *
import kmkns.Keyboard

grid = Grid([0.0, 0.0], [1.0, 1.0], [64, 64])
ns = NavierStokes2D(grid, nu=1.0/1000.0)
ns.set_bc('west+east+south', u=0, v=0)
ns.set_bc('north', u=-1, v=0)

ns.load('my_sim.dat')

kmkns.Keyboard.set_poll()
while True:
    ch = kmkns.Keyboard.get_key()
    if ch == 'q' or ch == 'Q':
        break
    ns.step(ns.find_suitable_dt())
    if ch == 'p' or ch == 'P':
        ns.plot_stream(20)
kmkns.Keyboard.set_normal()

ns.save('my_sim.dat')
```

7.10 Plotting

A number of different plotting methods are available:

- `plot_gas_fraction(fig=1)`, `plot_liquid_fraction(fig=1)`: Plot the number of gas or liquid cells relative to the total number of cells as a function of time. A gas cell is defined as a cell having a colour function value greater than 0.5. A liquid cell is defined as a cell having a colour function value less than or equal to 0.5.

- `plot_mass_centre(fig=1)`: Plot the path of the centre of mass over time. The centre of mass is a weighted average of all the points $\boldsymbol{x}$ in the computational domain Ω, where the weight is the fluid density ρ:

$$\boldsymbol{CM} = \frac{\int_\Omega \rho \boldsymbol{x} d\Omega}{\int_\Omega \rho \; d\Omega} \tag{7.1}$$

- `plot_gas_centre(fig=1)`, `plot_liquid_centre(fig=1)`: Plot the path of the gas or liquid mass centre over time.
- `plot_mass_centre_velocity(fig=1)`: Plot the speed of the centre of mass as a function of time.
- `plot_surface_tension(fig=1)`: Plot the surface tension force as a vector field.
- `plot_tracer(fig=1)`: Plot the tracer fluid.
- `plot_velocity(fig=1)`: Plot the velocity as a vector field.
- `plot_velocity_component(component, contours=None, fig=1)`: Plot one of the velocity components as a scalar field. If `component`=0, the horizontal velocity u is plotted. If `component`=1, the vertical velocity v is plotted.
- `plot_level(contours=None, fig=1)`: Plot the colour function field where 0 defines the liquid and 1 defines the gas.
- `plot_pressure(contours=None, fig=1)`: Plot the pressure field.
- `plot_curvature(contours=None, fig=1)`: Plot the estimated curvature as a scalar field. The curvature is only defined for cells close to the interface.
- `plot_divergence(contours=None, fig=1)`: Plot the divergence of the velocity. The divergence should be close to zero everywhere.
- `plot_vorticity(contours=None, fig=1)`: Plot the vorticity or curl of the velocity. The vorticity is defined as $\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$.
- `plot_stream(contours=None, fig=1)`: Plot the stream function. The stream function ϕ is defined as the solution to $u = \frac{\partial \phi}{\partial y}$ and $v = \frac{\partial \phi}{\partial x}$. The isolines of the stream function will be tangential to the velocity, which is useful for visualising the flow.

The `fig` keyword argument is the index of the figure or window to plot in, beginning with index 1. When plotting more than one plot, be sure to assign different indices to each plot, for instance:

```
ns.plot_velocity(fig=1)
ns.plot_pressure(fig=2)
ns.plot_stream(fig=3)
```

Some of the methods also accept a `contours` keyword argument:

- `contours=None`: Plot a colour map.

- `contours` is an integer n: Plot n evenly spaced contour lines.
- `contours` is a list: Plot contour lines for the values in the list.

Example:

```
stream_contours = [-0.1, -0.08, -0.06, -0.04, -0.02,
    -0.01, -3e-3, -1e-3, -3e-4, -1e-4, -3e-5, -1e-5,
    -3e-6, -1e-6, -1e-7, -1e-8, -1e-9, -1e-10, 0.0,
    1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 3e-6, 1e-5, 3e-5,
    1e-4, 3e-4, 1e-3, 3e-3, 0.01, 0.03, 0.05, 0.07,
    0.09, 0.1, 0.11, 0.115, 0.1175]
ns.plot_stream(contours=stream_contours, fig=1)
ns.plot_vorticity(contours=20, fig=2)
ns.plot_pressure(contours=None, fig=3)
```

7.11 Example files

The following example files are provided with the Navier-Stokes solver:

- `advection_test.py`
- `bubble_test.py`
- `channel_test.py`
- `cylinder_test.py`
- `drop_test.py`
- `poisson_test.py`
- `pressure_test.py`
- `profile_test.py`
- `rayleigh_taylor_test.py`
- `square_test.py`
- `static_drop_test.py`
- `step_test.py`
- `tutorial1.py`
- `tutorial2.py`

Chapter 8

Conclusion

The Navier-Stokes solver I have implemented is second order accurate in space and first order accurate in time. It seems to perform acceptably well in the tests I have run (for instance square cavity, backward facing step, rising bubble, static drop and Rayleigh-Taylor instability). It is easy to initialise and run simulations; about 10 lines of code is enough to initialise most of the common test problems. However, there are lots of things that can be added such as:

- Contact angle boundary condition for the interface between fluids.
- Heat and chemical transport
- Turbulence
- Non-rectangular obstacles
- Better visualisation independent of EasyViz and Gnuplot
- Multi-threading
- 3D
- Adaptive mesh refinement
- Move calculations to the GPU

Based on the profiling results, I believe there is a lot to gain from using the multi-threaded version instead of the single-threaded version of SuperLU which is currently being used in the PPE solver. Since two or more CPU cores is now common even in home computers, it is possible to reduce the computation time considerably by distributing the workload among several cores.

The second most time consuming part of the solver is the calculation of the tentative velocity. Even though this part is vectorised in Python, one can

probably achieve a small speed-up by translating it to C++. Vectorisation causes many temporary arrays to be allocated and deallocated for each iteration. In C++, the calculations will not be (completely) vectorised[1], thus temporary values can be held in registers or local variables. However, as long as solving the PPE takes most of the computation time, any speed-up in other parts of the solver will not be very noticeable.

If one is extending the `NavierStokes` module to 3D, I suggest adding a new class `NavierStokes3D`. Implementing the solver in 3D should be straight forward for most parts. However, ELVIRA and DAC with curvature refinement cannot readily be extended to 3D.

Since an explicit scheme is used in the solver, time-step restrictions are required for stability. Because of the viscosity term, the time-step restrictions become very strict for high resolution grids. For high Reynolds numbers and resolutions, the simulation becomes too slow for any practical use. Therefore, it might be a good idea to add support for a semi-implicit scheme.

Though obstacles are supported by my Navier-Stokes solver, its implementation has not yet been verified. A typical test problem for obstacles is flow past a circular cylinder. In my solver, the circle must be approximated with rectangles in a fine grid. Adding support for non-rectangular obstacles should be feasible for single-phase flow, for instance by using the immersed boundary method (IB). However, I do not think it is straight forward to mix VOF with IB.

Adaptive mesh refinement (AMR) is a method for increasing the grid resolution locally where the flow becomes more detailed, for instance near the interface in two-phase flows or around vortices. The mesh may dynamically be refined or coarsened. With AMR, one can spend time and memory only where it is needed, and the grid resolution does not need to be decided beforehand. However, adding AMR to the solver will complicate things considerably.

The Gnuplot Python module for Windows XP performed poorly. It often caused `IOError` to be raised, mostly during the first plots in a program and when several figures were plotted in quick succession. The module also held on to memory from old plots. Programs which plotted many plots therefore ended up eating all the memory on the computer and caused thrashing. If a program tries to save a plot and then exit, Gnuplot exits before it is done with the saving. Similarly, if a program tries to save a plot, then plot in the same figure, there is a great risk of overwriting parts of the first plot *while being saved*. One should consider switching to UNIX, using another backend for EasyViz than Gnuplot, using another plotting module such as matplotlib or if one is feeling ambitious, implementing one's own plotting

[1] Vectorising locally over a few elements, may be beneficial. This reduces the risk of stalling the CPU (where the CPU has to wait for the result of a previous instruction) and vector instruction (SIMD) may be used. Yet the temporary data is small enough to stay in registers.

module. Using different Gnuplot settings might also help.

There seems to be a gap between what is tought in textbooks and what is covered by articles. University courses and textbooks tend to be theoretical and only cover the high level aspects or the simplest parts of numerical programming. They focus on giving an overview of many different methods. Only basic examples, where everything fits beautifully, are shown. [9] and [33] are exceptions and are quite practical. Articles on the other hand usually assume that the reader has the basic knowledge already and skip the implementation details. They often omit details such as the implementation of boundary conditions, and the time-step restrictions are often incomplete.

I spent much time trying to find out why the velocity and pressure boundary conditions were implemented like they were. The no-slip boundary condition is well documented[28, 8] and the symmetry boundary condition looks reasonable. The outlet boundary condition, on the other hand, was not obvious. The implementation described in [9] led to a PPE without a solution, and the homogeneous Neumann boundary condition and the continuity equation could not both be fulfilled near the boundary. I ended up writing an email to the programmers of NaSt3DGP, the successor of NaSt2D[9], at the University of Bonn. I received a short email stating that the implementation in NaSt3DGP is the way to do it, leaving me none the wiser. Nonetheless, I implemented the boundary condition much like in NaSt3DGP. I also had trouble finding the details on the implementation of the pressure boundary condition, since the method described in [33] did not fit well into my solver. In the end, I implemented my own version.

Bibliography

[1] Robert W. Barber and Anne Fonty. A numerical study of laminar flow over a confined backwardfacing step using a novel viscous-splitting vortex algorithm. In *4th GRACM Congress on Computational Mechanics*, 2002.

[2] J. U. Brackbill, D. B. Kothe, and C. Zemach. A continuum method for modelling surface tension. *Journal of Computational Physics*, 100:335–354, 1992.

[3] Charles-Henri Bruneau and Mazen Saad. The 2d lid-driven cavity problem revisited. *Computers & Fluids*, pages 326–348, 2006.

[4] R. Fernandez-Feria and E. Sanmiguel-Rojas. An explicit projection method for solving incompressible flows driven by a pressure difference. *Computers & Fluids*, 33:463–483, 2004.

[5] Joel H. Ferziger and Milovan Peric. *Principles of Computational Fluid Dynamics*. Springer, 2001.

[6] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics*, volume II. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 1988.

[7] Marianne M. Francois, Sharen J. Cummins, Edward D. Dendy, Douglas B. Kothe, James M. Sicilian, and Matthew W. Williams. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, 213:141–173, 2006.

[8] Philip M. Gresho and Robert L. Sani. On pressure boundary conditions for the incompressible navier-stokes equations. *International Journal for Numerical Methods in Fluids*, 7(10):1111–1145, 1987.

[9] Michael Griebel, Thomas Dornsheifer, and Tilman Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. Society for Industrial Mathematics, 1997.

[10] Denis Gueyffier, Jie Li, Ali Nadim, Ruben Scardovelli, and Stéphane Zaleski. Volume-of-fluid interface tracking with smoothed surface stress methods for three-dimensional flows. *Journal of Computational Physics*, 152:423–456, 1999.

[11] Dalton J. E. Harvie and David F. Fletcher. A new volume of fluid advection algorithm: the defined donating region scheme. *International Journal for Numerical Methods in Fluids*, 35:151–172, 2001.

[12] Ashley J. James, Marc K. Smith, and Ari Glezer. Vibration-induced drop atomization and the numerical simulation of low-frequency single-droplet ejection. *Journal of Fluid Mechanics*, 476:29–62, 2003.

[13] James Edward Pilliod Jr. and Elbridge Gerry Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *Journal of Computational Physics*, 199:465–502, 2004.

[14] Hans Petter Langtangen, Kent-Andre Mardal, and Ragnar Winther. Numerical methods for incompressible viscous flow. *Advances in Water Resources*, 25:1125–1146, 2002.

[15] Hans Petter Langtangen. *Computational Partial Differential Equations.* Springer, 2003.

[16] Daniel Lörstad, Marianne Francois, Wei Shyy, and Laszlo Fuchs. Assessment of volume of fluid and immersed boundary methods for droplet computations. *International Journal for Numerical Methods in Fluids*, 46:109–125, 2004.

[17] Daniel Lörstad and Laszlo Fuchs. High-order surface tension vof-model for 3d bubble flows with high density ratio. *Journal of Computational Physics*, 200:153–176, 2004.

[18] Markus Meier, George Yadigaroglu, and Brian L. Smith. A novel technique for including surface tension in plic-vof methods. *European Journal of Mechanics - B/Fluids*, 21(1):61–73, 2002.

[19] Elin Olsson and Gunilla Kreiss. A conservative level set method for two phase flow. *Journal of Computational Physics*, 210:225–246, 2005.

[20] Stéphane Popinet and Stéphane Zaleski. A front-tracking algorithm for accurate representation of surface tension. *International Journal for Numerical Methods in Fluids*, 30:775–793, 1999.

[21] Elbridge Gerry Puckett, Ann S. Almgren, John B. Bell, Daniel L. Marcus, and William J. Rider. A high-order projection method for tracking fluid interfaces in variable density incompressible flows. *Journal of Computational Physics*, 130:269–282, 1997.

[22] M. Raessi, J. Mostaghimi, and M. Bussmann. Advecting normal vectors: A new method for calculating interface normals and curvatures when modeling two-phase flows. *Journal of Computational Physics*, 226:774–797, 2007.

[23] Yuriko Renardy and Michael Renardy. Prost: A parabolic reconstruction of surface tension for volume-of-fluid method. *Journal of Computational Physics*, 183:400–421, 2002.

[24] William J. Rider and Douglas B. Kothe. Reconstructing volume tracking. *Journal of Computational Physics*, 141:112–152, 1998.

[25] Gihun Son and Nahmkeon Hur. A coupled level set and volume-of-fluid method for the buoyancy-driven motion of fluid particles. *Numerical Heat Transfer, Part B: Fundamentals*, 42(6):523–542, 2002.

[26] Mark Sussman and Elbridge Gerry Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162:301–337, 2000.

[27] Mark Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *Journal of Computational Physics*, 187:110–136, 2003.

[28] R. Temam. Remark on the pressure boundary condition for the projection method. *Theoretical and Computational Fluid Dynamics*, pages 181–184, 1991.

[29] Murilo F. Tomé, Brian Duffy, and Sean McKee. A numerical technique for solving unsteady non-newtonian free surface flows. *Journal of Non-Newtonian Fluid Mechanics*, pages 9–34, 1996.

[30] Albert Y. Tong and Zhaoyuan Wang. A numerical method for capillarity-dominant free surface flows. *Journal of Computational Physics*, 221:506–523, 2007.

[31] Kristian Valen-Sendstad. Development of difference-method-based navier-stokes solver. Master's thesis, The Norwegian University of Science and Technology, 2006.

[32] S. P. van der Pijl, A. Segal, C. Vuik, and P. Wesseling. A mass-conserving level-set method for modelling of multi-phase flows. *International Journal for Numerical Methods in Fluids*, 47:339–361, 2005.

[33] H. K. Versteeg and W. Malalasekera. *An introduction to Computational Fluid Dynamics*. Prentice Hall, 1995.

[34] Pieter Wesseling. *Principles of Computational Fluid Dynamics.* Springer, 2001.