Technische Universität München

Department of Mathematics

Simulation of Thermohydraulic Phenomena in the PDE Framework Peano

Diploma Thesis by

Ralf Sangl

Supervisor:	Prof. Dr. Hans-Joachim Bungartz[†]
Advisor:	Dr. rer. nat. Tobias Neckel[†]
	Dipl. Ing. Michael Lieb, M. Sc.[†]
	Dipl.-Phys. Philipp Schöffel[‡]
	Dr.-Ing. Fabian Weyermann[‡]
Date:	February 24, 2011

[†]Technische Universität München, Chair of Scientific Computing

[‡]Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH

Statement of Authorship

I pledge that this thesis is the result of my own work. Material from other published or unpublished works of others, which is referred to in this thesis, is credited to the author in the text. No other sources or unauthorized aid has been used.

Garching bei München
February 24, 2011

Contents

Acknowledgements

Deepest gratitude goes to my advisors who showed imperturbable patience to all my unnerving questions: Tobias Neckel, Michael Lieb, Philipp Schöffel, and Fabian Weyermann. A successful completion of this thesis would not have been possible without their help. The support of GRS is also gratefully acknowledged.

Sincere thanks go to my friends Sa, Anh Thu, Moritz, and Thomas, without whose help I would never even have finished the first semester. I also want to thank Jilly for her assistance in correcting my English.

Furthermore I want to thank my beloved parents Gisela and Josef and my sister Janine for their affectionate support and being there for me throughout my whole life. Without them I wouldn't even have started studying.

To Lisa-Marie and Bianca.

1 Introduction

Scientific computing has become a significant part of engineering and physics. It is the connection between a theoretical mathematical model and real world applications. Real test rigs should and cannot be replaced completely, but scientific computing is cheaper, faster, and allows to optimize and analyze the design, properties, and behavior of a product which is in development. This includes the possibility to examine different setups at the same time, and picking out the best. Sometimes it is not even possible to build a test rig under certain physical conditions – e.g., when investigating a core meltdown in a nuclear power plant.

One typical field of application of scientific computation is fluid dynamics such as aerodynamics of cars and airplanes, meteorology, and internal flow through pipelines. They all deal with fluid flow, the natural science of fluids in motion. This is the main subject of this thesis. A flow can be described by laws of conservation for mass, momentum and energy – leading to a system of mathematical equations. The flow of several kinds of fluids are examined, ranging from water through honey to air. Note that fluid and flow are dependent on each other, but both have their own properties:

- Different types of flows are represented by a modified set of partial differential equations. Typical examples for flow properties are: steady vs. non-steady flow, laminar vs. turbulent flow, compressible vs. incompressible flow, flow with or without friction, single- and two-phase flow.
- The mechanical and thermodynamical fluid properties are described by an equation of state and constitutive laws just as, e.g., Fourier's law for heat conduction, the behavior of Newtonian fluids under shear stresses, or compressibility.

Be aware of compressibility and incompressibility being a property of both the flow and the fluid. The fluid's compressibility is a prerequisite for a compressible flow, but incompressible flows of compressible fluids are possible.

The *Chair of Scientific Computing* (SCCS) at the *Technische Universität München* (TUM) provides a framework for efficient calculation of partial differential equations (PDE) called *Peano*, developed by Tobias Weinzierl [15] and Tobias Neckel [11]. A component dealing with fluid dynamics on regular and adaptively refined Cartesian grids is already implemented in Peano. However Peano does not yet support the treatment of thermal energy. This is what this thesis is about: the derivation of the corresponding mathematical equation – the energy equation – and its implementation in Peano.

We simulate viscous, incompressible, laminar, single-phase flow of incompressible, Newtonian fluids with constant fluid parameters. We show that the thermal energy equation describes the energy of the system using these assumptions exhaustively. Thus, we introduce the temperature as a degree of freedom and are able to simulate and visualize the distribution and impacts of temperature differences. The Boussinesq approximation is used as a modification to approximate density differences that result in buoyancy forces. The model and the implementation are validated using natural convection and driven flow scenarios. Besides promising results we present an application to safety in nuclear power plants.

This thesis arose in close collaboration of the SCCS and the *Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) mbH.*

GRS carries out safety analyses of nuclear facilities such as nuclear power plants, research reactors, and fuel cycle and waste disposal facilities in Germany and abroad. In particular scientific software is developed and assessed for the simulation of nuclear power plant behavior under accident conditions. In the future, Peano shall be applied for the detailed simulation of complex, multidimensional flow phenomena in the cooling circuit of a *pressurized water reactor* (PWR). Of particular interest is the simulation of boron-dilution and temperature transients that can occur after loss-of-coolant accidents (LOCA) associated with the injection of emergency coolant necessary for the cooling of the reactor core. The imperfect mixing of cold safety injection water (30 °C) with the hot inventory of the primary circuit (300 °C) leads to significant temperature gradients which are essential for the analysis of the thermal shock related fracture of the reactor pressure vessel (RPV). To cope with these mixing phenomena with Peano satisfactorily, the provided implementation of an energy equation in combination with the Boussinesq-approximation is an essential prerequisite.

The fluid-fluid mixing was experimentally investigated in the frame of the *Transient and Accident Management* (TRAM) test series C1 and C2, carried out at the *Upper Plenum Test Facility* (UPTF) in Karlsruhe (Germany). These tests provide extensive data for the assessment and further validation of Peano and the models and methods used.

In Chapter 2 we take a look at the mathematical model, the Navier-Stokes equations. These are modified to use the flow and fluid properties we assume. Chapter 3 gives a short introduction to Peano and a discussion on implementation details. A short guide on how to use Peano is included as well. Chapter 4 is about the validation of both the mathematical model and the implementation. The validation happens both qualitatively and quantitatively, where Griebel's book [6] provides a good guideline. An application of Peano for the simulation of safety scenarios in nuclear power plants is discussed in Chapter 5. We conclude this thesis with an overview of achieved results and possible further enhancements for the further development of Peano.

2 Fluid Dynamics

Fluid dynamics is a field of fluid mechanics where the flow of a fluid is of particular interest. The flow and the fluid each have their own properties, which have been introduced in Chapter 1. The fluid dynamic equations are introduced for three-dimensional flows. These equations can easily be transformed into their two-dimensional equivalent.

We are interested in calculating the degrees of freedom at every point of the domain $\Omega \subset \mathbb{R}^3$ and at every time $t \in [t_0, t_{end}]$. Typically one has $t_0 = 0$ for the start time t_0. $t_{end} \in \mathbb{R}$ with $t_0 \leq t_{end}$ is the end time. The flow itself is described by a time dependent vector field $\vec{u}$ – the velocity. The Peano solution variables that we are interested in are the velocity $\vec{u}$, the pressure p and the temperature T:

$$\begin{aligned}
\vec{u} &: \Omega \times [t_0, t_{end}] \mapsto \mathbb{R}^3, \\
p &: \Omega \times [t_0, t_{end}] \mapsto \mathbb{R}, \\
T &: \Omega \times [t_0, t_{end}] \mapsto \mathbb{R}.
\end{aligned}$$

Since we will often need the components of $\vec{u}$ separately, we will use the common abbreviations instead of $\vec{u}_x$, $\vec{u}_y$ and $\vec{u}_z$:

$$u, v, w \quad : \quad \Omega \times [t_0, t_{end}] \mapsto \mathbb{R}, \tag{2.1}$$

$$\vec{u}(x,t) = \begin{pmatrix} u(x.t) \\ v(x,t) \\ w(x,t) \end{pmatrix}. \tag{2.2}$$

Our goal is to compute these functions using a mathematical model, that describes the flow – dependent on its properties, the domain, the boundary and the fluids properties of course. Regarding compressible, viscous flows the Navier-Stokes equations (NSE) are the underlying set of partial differential equations (PDE). The present report will investigate lamina, single phase flows of Newtonian, incompressible fluids.

In this Chapter we first derive the equations that are implemented in Peano using the mentioned assumptions. Afterwards, we take a look at boundary conditions that will be used in the simulations. At last we will state the dimensionless equations that can be useful in some cases.

2.1 Navier-Stokes Equations

The Navier-Stokes equations are a set of coupled, nonlinear partial differential equations based on the conservation laws for mass, momentum and energy. They will be introduced and modified, using assumptions concerning the flow's and fluid's properties.

2.1.1 Continuity Equation

The first law we will take a look at is the conservation of mass. Considering a closed system without any mass sources, the total mass inventory stays constant for all time. The continuity equation in its conservative form, where no additional assumptions are made for the density ρ (e.g., it does not need to be constant), reads (see [9, 11])

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) \;=\; 0. \tag{2.3}$$

The second term on the left hand side can be written as

$$\nabla \cdot (\rho \vec{u}) \;=\; \rho \nabla \vec{u} + \vec{u} \cdot \nabla \rho. \tag{2.4}$$

A thermal equation of state describing the thermodynamical properties of a fluid, says that the density ρ is solely dependent on the pressure p and the temperature T:

$$\rho \;=\; \rho(p,T) \;=\; \rho(p(x,t),T(x,t)). \tag{2.5}$$

For liquids incompressibility can be assumed and thus $\left(\frac{\partial \rho}{\partial p}\right)\Big|_{T\equiv const.} = 0$. The negligence of $\left(\frac{\partial \rho}{\partial T}\right)\Big|_{p\equiv const.}$ is as well a realistic assumption for liquids. Therefore, we assume a constant density at all times and at every point in the domain:

$$\rho(p,T) \;\equiv\; \rho_\infty \qquad \forall p, T, \tag{2.6}$$

where $\rho_\infty \in \mathbb{R}^+$. Thus, we can simplify equation (2.3) by using

$$\frac{\partial \rho}{\partial t} \;=\; 0, \tag{2.7}$$

$$\nabla \rho \;=\; 0, \tag{2.8}$$

and get the continuity equation for incompressible fluids,

$$\nabla \cdot \vec{u} \;=\; 0. \tag{2.9}$$

For the sake of completeness the equation is given explicitly as well,

$$\frac{\partial u}{\partial x}(x,t) + \frac{\partial v}{\partial y}(x,t) + \frac{\partial w}{\partial z}(x,t) \;=\; 0 \qquad \forall\,(x,t) \in \Omega \times [t_0, t_{end}]\,. \tag{2.10}$$

A derivation using the integral form of the continuity equation can be found in [9, 11].

2.1.2 Momentum Equation

The momentum equation postulates that the momentum of a (fluid) particle is a conserved quantity, which may only be changed by acting surface forces (e.g., pressure) and body forces (e.g., gravity). Momentum is the product of mass and velocity and is often referred to as linear momentum. The momentum equation often is considered to be the *central* equation of the NSE since it reflects the convective transport of flow. It is a direct application of Newton's second law of motion. This law states that the acceleration – and therefore the total movement – of an object is solely dependent on the net force and the mass of the object.

As before we start with the equation that does *not* assume a constant density. The gravity $g \in \mathbb{R}^3$ is the only surface force that we consider. Thus, the momentum equation reads (see [9, 11]):

$$\frac{\partial}{\partial t}(\rho\vec{u}) + (\vec{u}\cdot\nabla)(\rho\vec{u}) + (\nabla\cdot\vec{u})(\rho\vec{u}) - \nabla\cdot\boldsymbol{\sigma} = \rho g. \tag{2.11}$$

The stress tensor $\boldsymbol{\sigma}$ has to be chosen appropriately to the assumed material properties. Since we deal with viscous flows of Newtonian fluids $\boldsymbol{\sigma}$ takes the form (see [4])

$$\begin{aligned}\boldsymbol{\sigma} &= -p\boldsymbol{I} + \boldsymbol{\tau} \\ &= \left(-p - \frac{2}{3}\mu\nabla\cdot\vec{u}\right)\boldsymbol{I} + 2\mu\boldsymbol{\delta},\end{aligned} \tag{2.12}$$

where μ is the dynamic viscosity, $\boldsymbol{\tau}$ is the viscous part of the stress tensor, $\boldsymbol{\delta}$ is the strain tensor and $\boldsymbol{I} \in \mathbb{R}^{3\times 3}$ is the identity tensor.

The strain tensor $\boldsymbol{\delta}$ can be written as:

$$\boldsymbol{\delta} = \frac{1}{2}\begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} & 2\frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} & \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} & 2\frac{\partial w}{\partial z} \end{pmatrix}. \tag{2.13}$$

The divergence of $\boldsymbol{\sigma}$,

$$\nabla\cdot\boldsymbol{\sigma} = -\nabla p + \lambda\nabla(\nabla\cdot\vec{u}) + 2\mu\nabla\cdot\boldsymbol{\delta}, \tag{2.14}$$

appears in the momentum equation (2.11) where

$$\nabla\cdot\boldsymbol{\delta} = \frac{1}{2}\Delta\vec{u} + \nabla(\nabla\cdot\vec{u}) \tag{2.15}$$

can be obtained by using Schwarz' theorem (symmetry of second derivatives):

$$\frac{\partial^2\vec{u}_i}{\partial\vec{x}_j\partial\vec{x}_k} = \frac{\partial^2\vec{u}_i}{\partial\vec{x}_k\partial\vec{x}_j} \qquad \forall i,j,k \in \{1,2,3\}. \tag{2.16}$$

By assuming assuming incompressible fluids with constant density $\rho \equiv \rho_\infty$, we can use the continuity equation (2.9) so that $\nabla \cdot \vec{u}$ vanishes. Therefore, the momentum equation (2.11) – inserting the stress tensor $\boldsymbol{\sigma}$ (Eq. (2.14)) and the strain tensor (Eq. (2.15)) – can be written as follows:

$$\rho_\infty \frac{\partial \vec{u}}{\partial t} + \rho_\infty (\vec{u} \cdot \nabla) \vec{u} - \mu \Delta \vec{u} + \nabla p \quad = \quad \rho_\infty g. \tag{2.17}$$

In [9, 11] the reader can find a derivation using the integral form of the momentum equation.

2.1.3 Energy Equation

Similar to the balance equation for momentum and mass, there exists an empirical law saying that the total energy is conserved in an isolated system – total energy remains constant over time. By constant we mean, that energy can only be transformed from one form into the other, but neither can energy be created nor deleted. As it turns out throughout this Section, the conservation equation for energy can be simplified under certain conditions, resulting in a convection-diffusion equation for the temperature T.

There exist many formulations of the law of conservation of energy (see [8]. We derive the energy equation that is used in Peano from the general energy equation, which can be found in [7]:

$$\rho \frac{\mathrm{D}H}{\mathrm{D}t} \quad = \quad \kappa \Delta T + \rho \vec{u} \cdot g + \frac{\partial p}{\partial t} + \mathcal{D}. \tag{2.18}$$

For easier notation we introduce the material derivative for functions and vector fields,

$$\begin{aligned} \frac{\mathrm{D}}{\mathrm{D}t} &= \frac{\partial}{\partial t} + \vec{u} \cdot \nabla \qquad (2.19) \\ &= \frac{\partial}{\partial t} + u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} + w \frac{\partial}{\partial z}. \end{aligned}$$

H is the total specific enthalpy,

$$\begin{aligned} H &= h + \frac{1}{2} (\vec{u} \cdot \vec{u}) \\ &= h + \frac{1}{2} \left(u^2 + v^2 + w^2 \right), \qquad (2.20) \end{aligned}$$

where $\frac{1}{2} (\vec{u} \cdot \vec{u})$ is the specific kinetic energy of the system. h is the specific enthalpy,

$$h \quad = \quad e + \frac{p}{\rho}, \tag{2.21}$$

where e is the specific internal energy. The diffusion $\mathcal{D}$ comprises the viscous transport of the kinetic energy and its dissipation into heat due to friction. It can be written as:

$$\mathcal{D} = \nabla \cdot (\boldsymbol{\tau}\vec{u}) , \tag{2.22}$$

where $\boldsymbol{\tau}$ is the viscous part of the stress tensor $\boldsymbol{\sigma}$ (see Eq. (2.12)).

Using the definition of the total specific enthalpy (2.20) in the energy equation (2.18) we get:

$$\rho\frac{\mathrm{D}h}{\mathrm{D}t} + \frac{\rho}{2}\frac{\mathrm{D}}{\mathrm{D}t}\left(u^2+v^2+w^2\right) \;=\; \kappa\Delta T + \rho\vec{u}\cdot g + \frac{\partial p}{\partial t} + \mathcal{D}, \tag{2.23}$$

where the thermal conductivity κ – describing the ability to conduct heat – is used.

For constant fluid parameters we can simplify the material derivate of the specific enthalpy as follows (see [7]):

$$\begin{aligned}\rho\frac{\mathrm{D}h}{\mathrm{D}t} &= \rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} + \frac{\mathrm{D}p}{\mathrm{D}t}\\ &= \rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} + \frac{\partial p}{\partial t} + \vec{u}\cdot\nabla p.\end{aligned} \tag{2.24}$$

The second term of equation (2.23) can be rewritten:

$$\frac{\rho}{2}\frac{\mathrm{D}}{\mathrm{D}t}\left(u^2+v^2+w^2\right) \;=\; \rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t}, \tag{2.25}$$

using $\frac{\mathrm{D}u^2}{\mathrm{D}t} = 2u\frac{\mathrm{D}u}{\mathrm{D}t}$. These equations ((2.24) and (2.25)) are inserted into (2.23) providing the energy equation for constant fluid parameters:

$$\rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} - \kappa\Delta T + \rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} \;=\; -\vec{u}\cdot\nabla p + \rho\vec{u}\cdot g + \mathcal{D}. \tag{2.26}$$

2.1.3.1 Thermal Energy Equation

The energy equation that is presented in Eq. (2.26) contains contributions that are related to the mechanical energy, e.g., the kinetic energy and gravity contributions. As it will turn out, we can derive a pure thermal energy equation from the general energy equation by using the assumption of constant fluid parameters, still preserving the mechanical energy. Therefore we will rewrite the diffusion $\mathcal{D}$ from Eq. (2.22):

$$\begin{aligned}\mathcal{D} \;=\; \mu\Bigg(&\frac{\partial}{\partial x}\left[\frac{\partial u^2}{\partial x} + v\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right) + w\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right)\right]\\ &+\frac{\partial}{\partial y}\left[\frac{\partial v^2}{\partial y} + u\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right) + w\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\right]\\ &+\frac{\partial}{\partial z}\left[\frac{\partial w^2}{\partial z} + u\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right) + v\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\right]\Bigg).\end{aligned} \tag{2.27}$$

Applying the usual rules for derivation, e.g.,

$$\frac{\partial}{\partial x}\left(\frac{\partial u^2}{\partial x}\right) = 2u\frac{\partial^2 u}{\partial x^2} + 2\left(\frac{\partial u}{\partial x}\right)^2, \tag{2.28}$$

and Schwarz' theorem (see Eq. (2.16)) to Eq. (2.27), we get:

$$\begin{aligned}\mathcal{D} = {} & 2\mu\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right] \\ & +\mu\Bigg[\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)^2 \\ & + u\left(\Delta u + \frac{\partial}{\partial x}\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right]\right) \\ & + v\left(\Delta v + \frac{\partial}{\partial y}\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right]\right) \\ & + w\left(\Delta w + \frac{\partial}{\partial z}\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right]\right)\Bigg].\end{aligned} \tag{2.29}$$

Now we can insert the continuity equation (2.10) and simplify (2.29):

$$\begin{aligned}\mathcal{D} = {} & 2\mu\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right] \\ & +\mu\left[\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)^2\right] \\ & +\mu\vec{u}\cdot\Delta\vec{u}\end{aligned} \tag{2.30}$$

Note that we automatically assume constant density $\rho \equiv \rho_\infty$ by using the continuity equation. Nevertheless, we denote the density with ρ instead of ρ_∞ from now on. The dissipation Φ, which represents the part of the kinetic energy, that is transformed into heat due to viscous friction, reads (see [7]):

$$\begin{aligned}\Phi = {} & 2\mu\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right] \\ & +\mu\left[\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)^2\right].\end{aligned} \tag{2.31}$$

Thus, we can rewrite the diffusion $\mathcal{D}$ in Eq. (2.30):

$$\mathcal{D} = \Phi + \mu\vec{u}\cdot\Delta\vec{u}. \tag{2.32}$$

In order to simplify the energy equation (2.26) we need the three-dimensional momentum equation (2.17). It is multiplied by $\vec{u}$ from the left to result in a one-dimensional equation:

$$\rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} - \mu\vec{u}\cdot\Delta\vec{u} + \vec{u}\cdot\nabla p \quad = \quad \rho\vec{u}\cdot g. \tag{2.33}$$

This equation is then subtracted from the energy equation (2.26):

$$\begin{aligned} 0 \quad &= \quad \rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} - \kappa\Delta T + \rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} + \vec{u}\cdot\nabla p - \rho\vec{u}\cdot g - \mathcal{D} \\ &\qquad\qquad - \rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} - \vec{u}\cdot\nabla p + \rho\vec{u}\cdot g + \mu\vec{u}\cdot\Delta\vec{u} \\ &\overset{(2.32)}{=} \quad \rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} - \kappa\Delta T - \mu\vec{u}\cdot\Delta\vec{u} - \Phi \\ &\qquad\qquad + \mu\vec{u}\cdot\Delta\vec{u} \\ &= \quad \rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} - \kappa\Delta T - \Phi, \end{aligned} \tag{2.34}$$

resulting in the thermal energy equation,

$$\rho c_p\frac{\mathrm{D}T}{\mathrm{D}t} - \kappa\Delta T \quad = \quad \Phi. \tag{2.35}$$

Finally, using the thermal diffusivity α,

$$\alpha \quad = \quad \frac{\kappa}{\rho c_p}, \tag{2.36}$$

we get:

$$\frac{\partial T}{\partial t} + \vec{u}\cdot\nabla T - \alpha\Delta T \quad = \quad \frac{1}{\rho c_p}\Phi, \tag{2.37}$$

which is a classical convection-diffusion equation stating that temperature is convected with the flow ($\vec{u}\cdot\nabla T$) but also diffuses ($\alpha\Delta T$) uniformly in all directions.

2.1.3.2 Mechanical Energy Equation

By subtracting the momentum equation from the energy equation, we actually showed that the mechanical energy equation,

$$\rho\vec{u}\cdot\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} \quad = \quad -\vec{u}\cdot\nabla p + \rho\vec{u}\cdot g + \mathcal{D} - \Phi, \tag{2.38}$$

and the momentum equation (2.33) – multiplied with $\vec{u}$ from the left – coincide for constant fluid parameters. Thus, Eq. (2.38) is not an independent equation, in contrast

Table 2.1: Equations and degrees of freedom with dimension.

Equation Name	Equation	Dimension
continuity equation	(2.9)	1
momentum equation	(2.17)	3
thermal energy equation	(2.37)	1
mechanical energy equation	(2.38)	1

Degree of Freedom	Notation	Dimension
Velocity	$\vec{u}$	3
Pressure	p	1
Temperature	T	1

to Eq. (2.37). The latter describes the energy for a fluid with constant parameters *exhaustively*. Note that there are *5* degrees of freedom and *6* equations, where the mechanical energy equation is included in the momentum equation (Table 2.1).

Also note that – by assuming constant fluid properties – both the mass and momentum equation become independent from temperature and thus decouple from the thermal energy equation. The thermal energy equation represents a transport equation for the temperature and is convected by $\vec{u}$, which is calculated from the mass and momentum equation.

2.1.4 Boussinesq Approximation

In order to deal with the temperature in our equations we still need to modify our model. First of all note that there is not yet a coupling of the temperature with the momentum equation, so that the temperature does not have an influence on the flow. In general, the density is dependent on the temperature and pressure as is described in Section 2.1.1. Since we deal with incompressible fluids the pressure does not influence the fluid's properties and thus the density is not dependent on the pressure any more – but note that the pressure itself is *not* constant.

The Boussinesq approximation is a common approach to accomplish this modification. The following assumptions are made:

- The **density is constant except in buoyancy terms**. Therefore, we take into account that heated fluid rises due to decreasing density (and increasing specific volume) and cooled fluid drops down since it is more dense. On the other hand the density in the convective part of the momentum equation remains constant.

- There is a **linear relation between density and temperature** in the buoyancy term.
- All **other fluid parameters are constant** at all points in the domain and at all times (e.g., the viscosity μ and the thermal diffusivity α).
- The **viscous dissipation is negligibly small**; hence, we do not take into account that viscous friction leads to thermal energy.

Since buoyancy forces are inherently linked to gravity, the first assumption results in a modification of the momentum equation (see Eq. (2.17)):

$$\rho\frac{\partial \vec{u}}{\partial t} + \rho\left(\vec{u}\cdot\nabla\right)\vec{u} - \mu\Delta\vec{u} + \nabla p \;=\; \rho(T)g. \tag{2.39}$$

The second Boussinesq assumption then specifies the temperature dependent density $\rho(T)$ more precisely:

$$\rho(T) \;=\; \rho_\infty\left(1-\beta\left(T(x,t)-T_\infty\right)\right). \tag{2.40}$$

T_∞ is a reference quantity and β is the coefficient of thermal expansion,

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)\bigg|_{p\equiv const.}, \tag{2.41}$$

which is assumed to be constant as well (according to the third assumption of the Boussinesq approximation). For some $c \in \mathbb{R}^+$ we have $\rho(T_\infty + c)\,\|g\| < \rho_\infty\,\|g\|$ so that the gravity has less influence on a heated fluid and thus models the occurrence of Buoyancy forces.

The very final version of the momentum equation using the Boussinesq approximation and $\rho = \rho_\infty$ reads:

$$\rho\frac{\partial \vec{u}}{\partial t} + \rho\left(\vec{u}\cdot\nabla\right)\vec{u} - \mu\Delta\vec{u} + \nabla p \;=\; \rho\left(1-\beta\left(T-T_\infty\right)\right)g, \tag{2.42}$$

where the temperature $T = T(x,t)$ is dependent on both the point in the domain x and the time t of course.

The continuity equation (2.9) already made heavy use of the assumption that the density is constant and therefore remains as is.

The last simplification affects the thermal energy equation (2.37). Neglecting the viscous dissipation Φ results in:

$$\frac{\partial T}{\partial t} + \vec{u}\cdot\nabla T - \alpha\Delta T = 0. \tag{2.43}$$

The Boussinesq approximation is adequate for moderate temperature and density differences (see [6]). We have now finished all mathematical modifications for the Peano set of the Navier-Stokes equations.

2.1.5 Summary

Following, we give a summary of all equations that are used in Peano and need to be discretized. As the density is assumed to be constant, we will no longer use ρ_∞ for the density and use ρ throughout the rest of this and all following chapters.

The continuity equation (2.9) is a one-dimensional equation for conservation of mass:

$$\nabla \cdot \vec{u} \;=\; 0.$$

The three-dimensional momentum equation (2.17) conserves momentum and takes Buoyancy forces into account:

$$\rho \frac{\partial \vec{u}}{\partial t} + \rho \left(\vec{u} \cdot \nabla \right) \vec{u} - \mu \Delta \vec{u} + \nabla p \;=\; \rho \left(1 - \beta \left(T - T_\infty \right) \right) g.$$

And finally, the (thermal) energy equation (2.43) for conservation of energy is one-dimensional again:

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T - \alpha \Delta T \;=\; 0.$$

2.2 Boundary Conditions and Initial Values

Since we want to solve a partial differential equation in the domain Ω starting at time t_0 we need to specify initial values and boundary conditions. There are two types of boundary conditions – *Dirichlet* and *Neumann* boundaries – each of which tries to simulate some kind of real world boundary.

2.2.1 Initial Values

The initial values are separately defined functions for each degree of freedom,

$$\begin{aligned} \vec{u}_I &: \Omega \mapsto \mathbb{R}^3, \\ T_I &: \Omega \mapsto \mathbb{R}, \end{aligned}$$

which are then set for the degrees of freedom at time t_0:

$$\begin{aligned} \vec{u}(\vec{x}, t_0) &= \vec{u}_I(\vec{x}) \qquad \forall \vec{x} \in \Omega, \\ T(\vec{x}, t_0) &= T_I(\vec{x}) \qquad \forall \vec{x} \in \Omega. \end{aligned}$$

Note that the pressure p does not have an initial value. This is because the pressure is the solution of a linear system in Peano – the *pressure Poisson equation*. The derivation of this equation can be found in [11]. Nevertheless, it is possible to set pressure boundary conditions, e.g., at the inlet and outlet of a channel – using additional forces, which is also described in [11].

2.2.2 Dirichlet Boundary Condition

We need to specify a boundary condition for each degree of freedom at every point $\vec{x} \in \Gamma$ and every time $t \in [t_0, t_{end}]$. Therefore, we define

$$\Gamma^{\vec{u}} = \Gamma, \quad \text{and} \quad \Gamma^{T} = \Gamma,$$

to differentiate between the velocity boundary and the temperature boundary.

A Dirichlet boundary (subscripted with D, e.g, Γ^T_D) is the most intuitive boundary condition, since it sets the values explicitly. It is described by the functions

$$\begin{aligned} \vec{u}_D &: \Gamma^{\vec{u}}_D \times [t_0, t_{end}] \mapsto \mathbb{R}^3, \\ T_D &: \Gamma^{T}_D \times [t_0, t_{end}] \mapsto \mathbb{R}, \end{aligned}$$

and is set appropriately:

$$\begin{aligned} \vec{u}(\vec{x},t) &= \vec{u}_D(\vec{x},t) \qquad \forall\,(\vec{x},t) \in \Gamma^{\vec{u}}_D \times [t_0, t_{end}]\,, \\ T(\vec{x},t) &= T_D(\vec{x},t) \qquad \forall\,(\vec{x},t) \in \Gamma^{T}_D \times [t_0, t_{end}]\,. \end{aligned}$$

For a wall typically $\vec{u}_D \equiv 0$ is used so that a fluid moving along this wall will be slowed down due to its viscosity. This special kind of Dirichlet boundary is often called *no-slip* boundary for obvious reasons. Most scenarios have an inlet, where some kind of pump pushes the fluid into the domain – typically using a special inflow profile that is constant over time (such as a parabolic velocity profile). The same holds for the temperature, we will often have a heated wall or a heated inflow, so that we need to use Dirichlet boundaries for the temperature there.

2.2.3 Neumann Boundary Condition

For a Neumann boundary (subscripted with N, e.g., $\Gamma^{\vec{u}}_N$) not everything is as set as for the Dirichlet boundary. They are often used to simulate open boundaries where the program simply does not know what happens outside of the domain. We want that the flow is not at all influenced by this boundary and adjusts appropriately to the flow. Mathematically, we want the partial derivative to vanish in both the normal and the tangential direction:

$$\begin{aligned} \frac{\partial \vec{u}_n}{\partial n}(\vec{x},t) &= 0 \qquad \forall\,(\vec{x},t) \in \Gamma^{\vec{u}}_N \times [t_0, t_{end}]\,, \\ \frac{\partial \vec{u}_\tau}{\partial \tau}(\vec{x},t) &= 0 \qquad \forall\,(\vec{x},t) \in \Gamma^{\vec{u}}_N \times [t_0, t_{end}]\,, \end{aligned}$$

where n is the outer normal and τ is the tangential direction. These conditions are often used in pipes where we have an inflow using Dirichlet boundary conditions and the outflow getting a Neumann boundary. For the temperature these are used to simulate, for example, adiabatic (insulated) walls.

2.2.4 Slip-Wall Boundary Condition

For multidimensional degrees of freedoms like the velocity the former two boundary condition types can be mixed up and result in the so called *slip-wall* or *mixed Dirichlet-Neumann* condition (subscripted with $slip - wall$). Then they represent a boundary where we do not allow an influence of the boundary in the tangential direction (applying Neumann conditions) and have a usual wall in normal direction resulting in no-slip Dirichlet conditions:

$$\begin{aligned} \vec{u}_n(\vec{x},t) &= 0 \qquad \forall\, (\vec{x},t) \in \Gamma^{\vec{u}}_{slip-wall} \times [t_0, t_{end}]\,, \\ \frac{\partial \vec{u}_\tau}{\partial \tau}(\vec{x},t) &= 0 \qquad \forall\, (\vec{x},t) \in \Gamma^{\vec{u}}_{slip-wall} \times [t_0, t_{end}]\,. \end{aligned}$$

An example of this boundary type can be found in Section 4.3 on Page 64, which is the *Flat Plate in Parallel Flow*-scenario. This kind of boundary condition makes no sense for one-dimensional degrees of freedom like the temperature, since there is no parameter left after applying a Dirichlet condition.

2.3 Dimensionless Equations

So far we have not talked about units for all of our parameters. Thus, we imply that SI units are used for these (e.g., the time t is measured in seconds s). Nevertheless, one often reads about dimensionless equations and parameters. They are useful when one needs to compare, e.g., flows on different domain sizes.

In order to state the dimensionless equations we need dimensionless parameters, which are denoted by a superscripted asterisk (e.g., t^*). To define those we need characteristic reference values subscripted by an infinity symbol (e.g., T_∞). The parameters can be split up in dimensionless coordinates:

$$\begin{aligned} \vec{x}^* &= \frac{\vec{x}}{L_\infty}, \\ t^* &= \frac{u_\infty}{L_\infty} t, \end{aligned}$$

and dimensionless states:

$$\begin{aligned}\vec{u}^* &= \frac{\vec{u}}{u_\infty}, \\ p^* &= \frac{p - p_\infty}{\rho_\infty u_\infty^2}, \\ T^* &= \frac{T - T_\infty}{T_H - T_C}, \\ g^* &= \frac{L_\infty}{u_\infty^2} g, \text{ and} \\ \beta^* &= (T_H - T_C)\,\beta,\end{aligned}$$

where T_H and T_C are the highest and lowest temperature respectively (considering natural convection phenomena a heated wall will have temperature T_H, e.g.).

Aside from these quantities we get dimensionless numbers, which can be used to compare different kinds of fluid in different kinds of geometries. Typically the most important is the Reynolds number Re,

$$\mathrm{Re} = \frac{\rho_\infty u_\infty L_\infty}{\mu} = \frac{u_\infty L_\infty}{\nu}, \tag{2.44}$$

giving a measure of the ratio of inertial to viscous forces. $\nu = \frac{\mu}{\rho_\infty}$ is the kinematic viscosity of the fluid. The characteristic velocity u_∞ is usually set to the inflow velocity and therefore this quantity does not have that much relevance in natural convection scenarios, where no external flow is involved.

Another dimensionless number is the Prandtl number Pr,

$$\mathrm{Pr} = \frac{\nu}{\alpha}, \tag{2.45}$$

which is the ratio of momentum diffusivity to thermal diffusivity. Note that this number does not contain a length scale and therefore is a property of the fluid itself, whereas the Reynolds number is dependent on both the scenario and the fluid properties.

These numbers are then used to give a dimensionless set of equations (see [6]). The continuity equation does not hold any fluid specific property and thus is not altered by division and multiplication:

$$\nabla^* \cdot \vec{u}^* = 0.$$

The dimensionless momentum equation with Boussinesq approximation reads:

$$\frac{\partial \vec{u}^*}{\partial t^*} + (\vec{u}^* \cdot \nabla^*)\,\vec{u}^* - \frac{1}{\mathrm{Re}}\Delta^* \vec{u}^* + \nabla^* p^* = (1 - \beta^* T^*)\,g^*.$$

Finally, we state the dimensionless energy equation:

$$\frac{\partial T^*}{\partial t^*} + \vec{u} \cdot \nabla^* T^* - \frac{1}{\mathrm{Re}\,\mathrm{Pr}}\Delta^* T^* = 0.$$

3 Peano

Peano is a framework that allows numerical simulation of partial differential equations (PDE) in any dimension and is programmed in `C++`. In particular Peano has a component that supports the computation of fluid dynamics – the `fluid` component. Our goal is to use this component and extend it by the heat equation. First we give a short survey of Peano and the general concept – a sophisticated description of Peano can be found in [15] and [11]. Afterwards, we show implementation details for the `chemical` component. These contain the modification of the momentum equation, the introduction of the energy equation and other partly technical details. The name `chemical` developed historically, since it originally was intended to simulate the transport of chemical substances. Since this transport is very similar to the transportation of heat, we decided to keep the name and thus when talking about the `chemical` component we think of the transportation of heat.

3.1 General Concept

Peano is a memory efficient framework for solving PDEs on regular (`trivialgrid`) and adaptively refined Cartesian grids (`grid`). The name is based on the Peano curve, which is a space-filling curve, originally thought of as two dimensional, but easily extended to an arbitrary dimension. The domain Ω is split up in pairwise disjoint rectangular cells. These are then put on stacks as described in [11, 15] and allow a very efficient traversal of the grid. The two different types of grids – `trivialgrid` and `grid` – is explained further in Section 3.7. Until then we refer to the `trivialgrid` and thus to a regular discretization of the domain Ω.

The general concept of Peano is to do everything cell-wisely, so that every cell can be treated independently. A cell does not know anything about its neighbors nor the domain, it solely knows itself, whether it is inside or outside of Ω and the 2^d vertices that belong to the cell. The degrees of freedom are either put on a vertex or in the center of a cell. In the `fluid` component of Peano the velocity $\vec{u}$ is placed on the vertices, whereas the pressure p is a cell degree of freedom. Since the `chemical` component is a subcomponent of `fluid` these stay as they are. Additionally, the temperature is implemented as a scalar vertex degree of freedom.

3.2 The `fluid` Component

The `fluid` component implements the incompressible Navier-Stokes equations that have been derived in the previous chapter without the energy equation. A finite element method (FEM) approach is used to discretize the spatial operators. A good introduction to FEM can be found in [1, 5]. The ansatz and test functions – called Φ and v, respectively – are bi-linear functions for the velocity and piecewise constant functions for the pressure. In the `fluid` component the velocity is a vertex degree of freedom, whereas the pressure is a cell degree of freedom.

3.3 Vertex and Cell

First of all we need to introduce a special vertex data type that enhances the `fluid` vertex. The tool *DaStGen* (Data Structure Generator, [3]) is used to generate `C++` classes. We generate the record class `FluidChemicalVertexDoF`, which holds the same attributes as a `FluidVertexDoFWithPersistentCellNumber` vertex. In addition to the `fluid` attributes the `chemical` vertex has the following attributes:

- `T` (`double`) for the temperature T,
- `TTransport` (`double`) for the temperature transport, which is used as a temporary variable throughout a grid traversal on each vertex, and
- `heatScenarioVertexType` (`enum HeatScenarioVertexType`) to distinguish whether the temperature degree of freedom is on a heated (`HEAT_DIRICHLET_HEAT`), a cooled boundary (`HEAT_DIRICHLET_COOL`), an adiabatic (isolated) boundary (`HEAT_NEUMANN`), in the domain (`HEAT_INNER`) or outside of it (`HEAT_OUTER`).

The templated record-proxy class `TrivialgridFluidChemicalVertexEulerExplicit` (subclass of `TrivialgridFluidVertexEulerExplicit`) defines setters and getters for all chemical attributes. This proxy class is useful, because it allows to add separate functionality for the `chemical` vertex, which cannot be generated by DaStGen. When using the `trivialgrid` component one typically uses the concrete instantiation `TrivialgridFluidChemicalVertexEE`.

The cell does not hold anything specific to the `chemical` component. Hence, we decided to use the `fluid` cell `TrivialgridFluidCellWithPersistentCellNumber` in combination with the `trivialgrid`. In contrast to that, there is a `chemical` cell class for the `grid` implementation as is explained in Section 3.7.

3.4 Momentum Equation

We altered the momentum equation (2.17) by including the Boussinesq approximation. Thus, the implementation needs to be adapted accordingly. Furthermore, it should be taken into account that not all future Peano simulations will be using the `chemical` component. Therefore, the gravity factor in the momentum equation,

$$\rho\frac{\partial\vec{u}}{\partial t} + \rho\left(\vec{u}\cdot\nabla\right)\vec{u} - \mu\Delta\vec{u} + \nabla p \quad = \quad \rho\underbrace{\left(1-\beta\left(T-T_\infty\right)\right)}_{=\,\texttt{GravityFactor}}g,$$

is implemented as an extra class `ComputeGravityFactor`. An UML class diagram (Fig. 3.1) shows that the `fluid` component is independent of the `chemical` component as far as the implementation of the `GravityFactor` is concerned. The class `AbstractCalculateF` holds an instance of `ComputeGravityFactor`. The former needs to know about the gravity factor when `accumulateFValues` is called, since it belongs to the `F` part of the momentum equation (see [11] and Section 3.5). The method is called "`accumulate`" since a vertex typically does not belong to one cell alone and everything happens in a cell-wise manner in Peano. Thus, the computations have to be split up and the results have to be accumulated on the vertices.

The `ComputeGravityFactor` class has a method called `computeGravityVector`, where a `Vector` of temperature values is supplied. When running a standard `fluid` simulation this method returns a `Vector` containing only values equal to 1, since there is no Boussinesq approximation involved and no real temperatures are available. For a `chemical` simulation the method is overridden in the `ComputeHeatGravityFactor` class to compute the factor

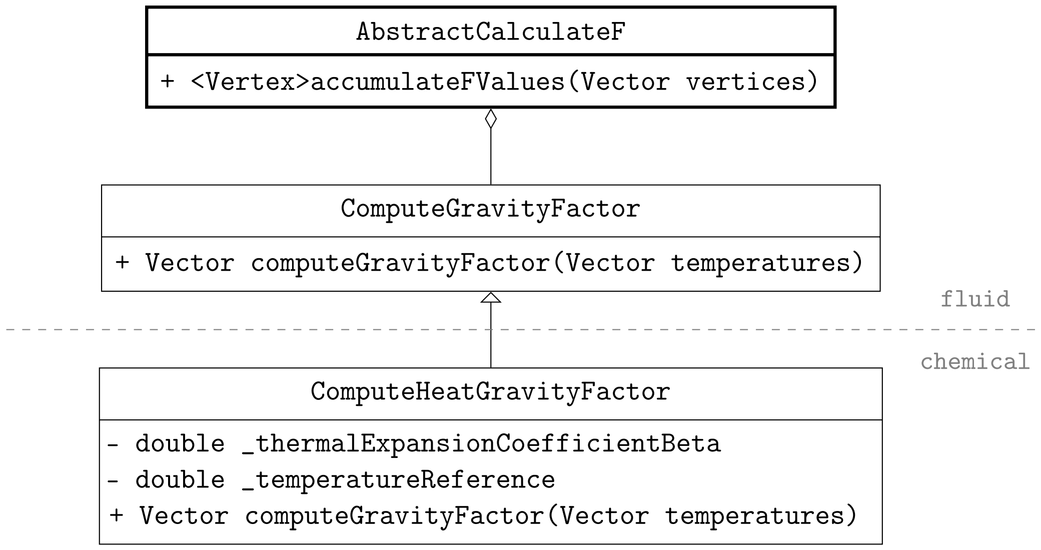

AbstractCalculateF
+ <Vertex>accumulateFValues(Vector vertices)
ComputeGravityFactor
+ Vector computeGravityFactor(Vector temperatures)
fluid
chemical
ComputeHeatGravityFactor
- double _thermalExpansionCoefficientBeta
- double _temperatureReference
+ Vector computeGravityFactor(Vector temperatures)

Fig. 3.1: UML Class diagramm for `ComputeHeatGravityFactor`.

for each given temperature. These temperatures belong to the vertices of the currently treated cell. The factor is loaded while generating a child of `AbstractCalculateF` in the operator factory of the `fluid` (`OperatorFactory`) and `chemical` (`ChemicalOperatorFactory`) component, respectively.

3.5 Energy Equation

It has been pointed out that everything in Peano happens in a cell-wise manner and the computations for vertices have to be split up accordingly. This approach is reflected in the implementation of the adapters. Each adapter has several methods like the following – for the `trivialgrid` component – that are called during a grid traversal:

- `touchVertexFirstTime(...)` is called whenever a vertex is touched for the first time for the current adapter in the current grid traversal,
- `handleElement(...)` is called on a cell after all neighboring vertices received their call to `touchVertexFirstTime`, and
- `touchVertexLastTime(...)` is the last method that is invoked for every vertex, after all cells to which the vertex belongs have been treated with.

The interface of the `TrivialgridEventHandle2TemperatureTransportAdapter` adapter is shown in Figure 3.2. We do not need to care about the grid traversal and at which particular moment a vertex is touched for the first time. This is all done by other components and thus an advantage worth noting. Also note that only *one* adapter is necessary to implement all the functionality for `TTransport`.

The energy equation (2.43) has a term for convective and diffusive heat transport, very similar to the terms in the momentum equation (2.42). Therefore, the implementation resembles the implementation of the `F`-term of the momentum equation (2.42),

$$\rho\frac{\partial\vec{u}}{\partial t} + \underbrace{\rho\left(\vec{u}\cdot\nabla\right)\vec{u} - \mu\Delta\vec{u} - \rho\left(1-\beta\left(T-T_\infty\right)\right)g}_{=\,\mathtt{F}} + \nabla p = 0,$$

in the `fluid` component of Peano. The energy equation has no internal heat source which would correspond to the gravity – let it be zero. The term are looking at now is called `TTransport`,

$$\frac{\partial T}{\partial t} + \underbrace{\vec{u}\cdot\nabla T - \alpha\Delta T}_{=\,\mathtt{TTransport}} = 0,$$

and every vertex (both the `trivialgrid` and the `grid` vertex) holds a `double` attribute with this name. The spatial operators that are used for `F` are the same for `TTransport`.

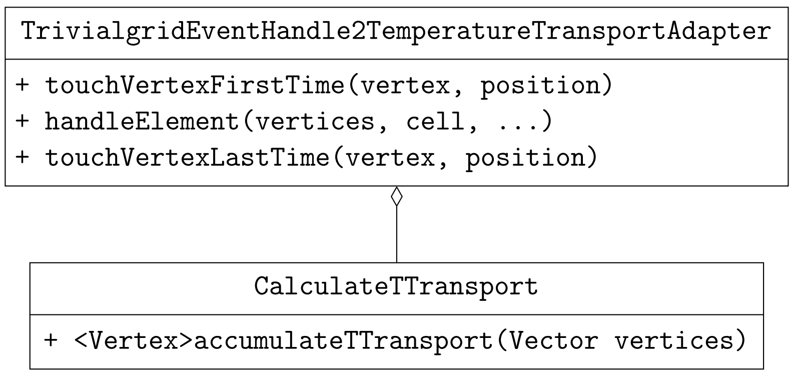

TrivialgridEventHandle2TemperatureTransportAdapter
+ touchVertexFirstTime(vertex, position)
+ handleElement(vertices, cell, ...)
+ touchVertexLastTime(vertex, position)
CalculateTTransport
+ <Vertex>accumulateTTransport(Vector vertices)

Fig. 3.2: UML class diagram for `CalculateTTransport`.

Note that the operators are applied to each component, so that is no problem to use the same operators for the three-dimensional velocity and the scalar temperature.

We implemented the class `CalculateTTransport` which corresponds to `AbstractCalculateF`. Unlike the `F` term, the `TTransport` term only supports one operator discretization[1] so far, so that `CalculateTTransport` does *not* need to be a templated class. An UML diagram (Fig. 3.2) again shows the simple relation between the adapter and the concrete implementation – which is independent of the used grid component in contrast to the adapter.

During a grid traversal we issue the following calls concerning the temperature update on a vertex: `resetTTransport`, `accumulateTTransport`, and `updateTemperatureWithExplicitEuler` (Table 3.1). The first and the last of these methods are implemented for both, the `trivialgrid` and the `grid` vertex. `resetTTransport` just sets the `TTransport` attribute to zero.

The time-dependent temperature update is then computed using the explicit Euler method (first order approximation),

$$\frac{\partial T}{\partial t}(x,t) \quad \doteq \quad \frac{T(x,t+\tau)-T(x,t)}{\tau},$$

where τ is the current time step size. This update is computed in the vertex method `updateTemperatureWithExplicitEuler`. This concludes the treatment of the energy equation. Note that the momentum equation is not considered when calculating the update for the temperature, since the update – the calculation of `TTransport` and the call to `updateTemperatureWithExplicitEuler` – is performed before the momentum equation is dealt with (in the current time step). This is a common approach which can be seen in [6].

[1]The `fluid` component also supports divergence free and enhanced divergence free elements.

Table 3.1: Event and action on vertices and cells during grid traversal for `TTransport` calculations in the `trivialgrid` adapter `TrivialgridEventHandle2TemperatureTransportAdapter`.

Event	Action
`touchVertexFirstTime(...)`	`resetTTransport`
`handleElement(...)`	`accumulateTTransport`
`touchVertexLastTime(...)`	`updateTemperatureWithExplicitEuler`

3.6 Adaptive Time Step Size

The equations we want to solve contain both a spatial derivative and a time derivative. The former is dealt with using FEM which is described in Section 3.5, whereas for the latter an explicit Euler method,

$$\frac{df}{dt}(t) \doteq \frac{f(t+\tau)-f(t)}{\tau},$$

is implemented in the `chemical` component. As with every numerical method that is time dependent, the algorithm needs a time step size τ. The user has two methods of providing an accurate τ.

The first is to explicitly provide τ and the number of steps that should be performed in the `ode` tag of the configuration file. The corresponding attribute names are `tau` and `number-of-time-steps`, see Fig. 3.9 for an example. Additionally, there are attributes for the start time t_0 (`start-time`) and the total running time t_{end} (`end-time`), which need to be specified in the configuration file as well.

The second method is to use the adaptive time step size computation that assures numerical stability. Some of these restrictions are called CFL (Courant-Friedrichs-Lewy) conditions and are well presented in [6, 11, 14, 16]. These conditions state that a fluid particle may not be transported further than the size of a cell in every direction,

$$\begin{aligned}\tau_u &= \frac{h_x}{2u_{max}},\\ \tau_v &= \frac{h_y}{2v_{max}}, \text{ and}\\ \tau_w &= \frac{h_z}{2w_{max}},\end{aligned}$$

where h_x, h_y and h_z is the minimum mesh width in each dimension and u_{max}, v_{max} and w_{max} is the maximal velocity in each dimension, respectively. The CFL condition for the time step size update then reads:

$$\tau_{cfl} = S_{cfl}\ \min\{\tau_u, \tau_v, \tau_w\},$$

where $S_{cfl} = 0.5$ is a safety factor. Further conditions that include the fluid's properties, like the kinematic viscosity ν, exist[6, 11]:

$$\tau_{visc} \quad = \quad S_{visc} \, \frac{1}{2\nu} \left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2} \right)^{-1},$$

using $S_{visc} = 0.5$. This equation assures the numerical stability of the momentum equation (2.42). Additionally, to these we have to provide a stability condition for the energy equation (2.43):

$$\tau_{thermal} \quad = \quad S_{thermal} \, \frac{1}{2\alpha} \left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2} \right)^{-1},$$

with $S_{thermal} = 0.5$. Note that both the condition due to viscosity and thermal diffusivity have a *quadratic* dependency on the mesh size. Thus, by refining the mesh a smaller time step size is necessary as well. The time step size τ for the next time step is then calculated as the minimum of all conditions:

$$\tau \quad \leq \quad S_{global} \; \min \left\{ \tau_{cfl}, \tau_{visc}, \tau_{thermal} \right\},$$

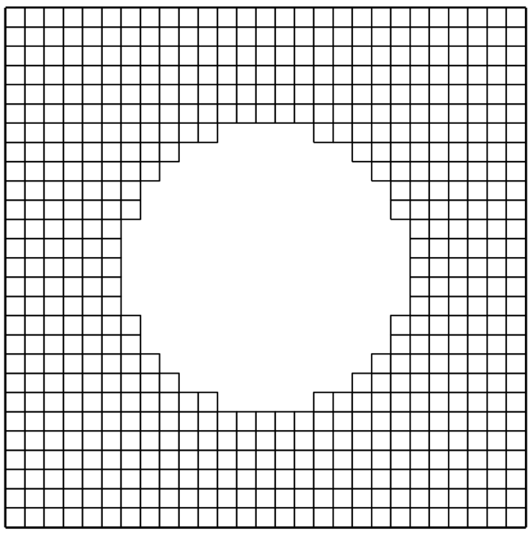
where $S_{global} = 0.8$ is the global security factor.

These stability conditions are implemented in the classes `FluidSimulation` and `FluidChemicalSimulationBase`, which is not discussed further. In order to use the adaptive time step size computation, the `tau` attribute of the `ode` tag has to be omitted in the configuration file (see Section 3.9).

3.7 Adaptive Grid

Peano has two different types of grids to discretize the domain Ω. The `trivialgrid` is the canonical way of splitting Ω into cells. The user specifies the number of cells for each dimension in the configuration file (see `trivialgrid` tag in Fig. 3.9). The domain is then divided accordingly and all the algorithm needs to know about is the cell width in each dimension. The mesh of a box (edge length 1 m) with a sphere (radius 0.25 m) in the center shows a rather ragged contour (Fig. 3.3(a)), when using 27×27 cells. In order to get it smoother, one needs to refine the whole domain leading to a lot of cells, where only a few might be necessary at the border of the sphere.

The other type of grid Peano supports is an adaptive grid – further referred to as `grid`. The `grid` component is the strength of Peano where the advantages of the cell-wise evaluation approach is fully exploited (stack based cell treatment). It allows to trisect a cell and therefore refine a certain part of the domain discretization. At boundaries this is done automatically by defining a maximum mesh width and a minimum mesh width


(a) `trivialgrid` mesh using 27 × 27 cells.

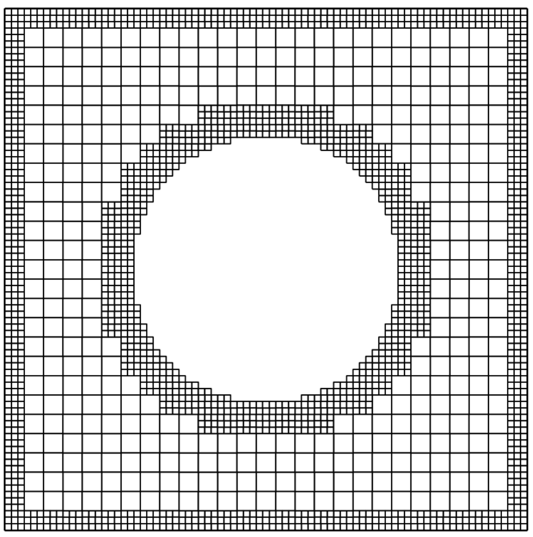

(b) `grid` mesh using $h_{min} = 1/27$, $h_{max} = 1/81$.

Fig. 3.3: Comparison of a `trivialgrid` and a `grid` mesh on a square domain with edge length 1 m with an obstacle (sphere with radius 0.25 m) in the center.

in each `geometry` tag of the configuration file. An example of the `grid` in comparison to the `trivialgrid` is shown in Fig. 3.3(b).

The independent trisection of cells introduces the concept of levels, where the root level consists of the coarsest cells without any refinement. A trisected cell itself is a cell and can therefore be refined again (compare to Fig. 3.4 with 3 levels of refinement). Note that the `fluid` component of Peano currently only supports to have a level difference of 1 on neighboring cells!

The main difference is the existence of the so called *hanging nodes*. These nodes exist along edges of refinement level transitions and do not hold a real degree of freedom – on the current level. Therefore, they need to be interpolated linearly – when using linear FEM ansatz functions – from the coarser to the finer and restricted from the finer to the coarser level, respectively. The weight factors are shown in Figure 3.5. Since interpolation and restriction are used quite often on the vertices, the according methods are placed in the vertex class. In the `chemical` component they can be called for both the temperature `T` (`interpolateT(...)`) and the temperature transport `TTransport` (`restrictTTransport`) (see Section 3.5). The same methods exist for the velocity and other parameters in the `fluid` component, see [11] for more information.

When implementing the `grid` support for the `chemical` component, we need to introduce a new vertex type `GridFluidChemicalVertexEulerExplict` and a new cell class

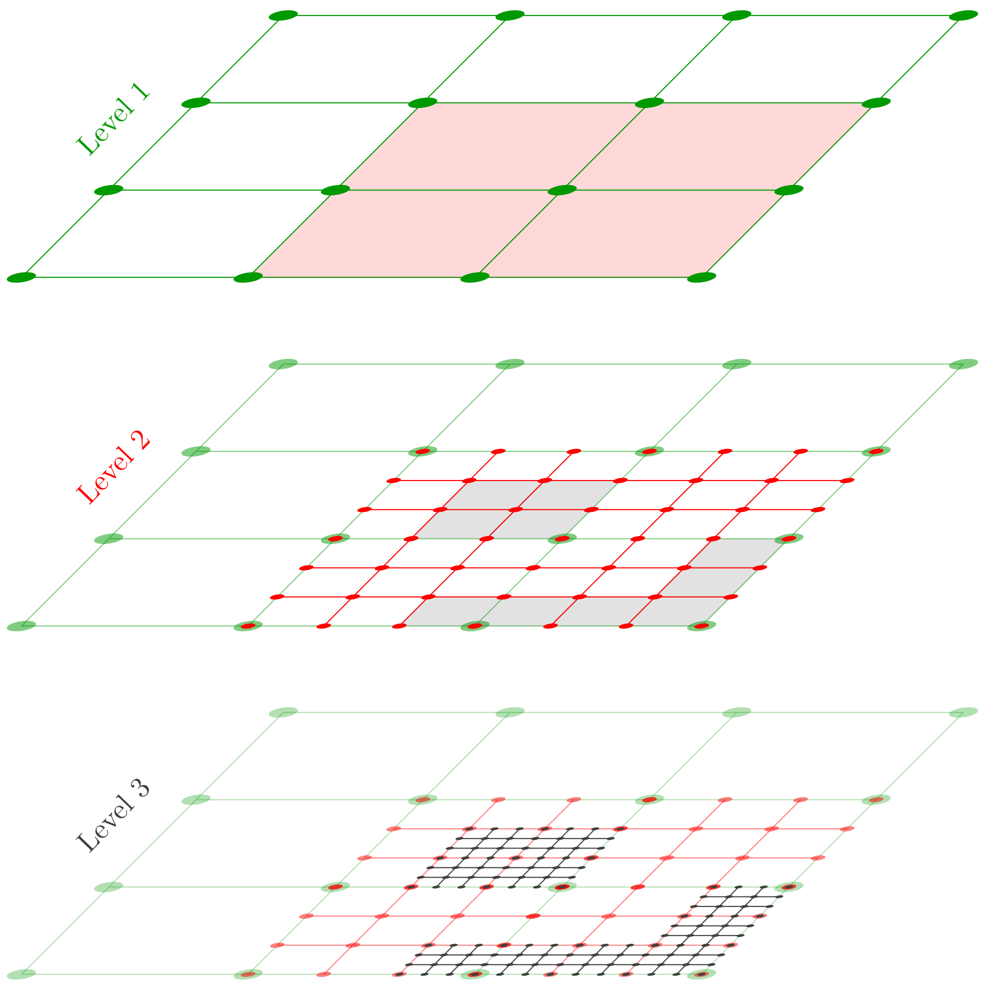


Fig. 3.4: Scheme of a grid with 3 levels of refinement.

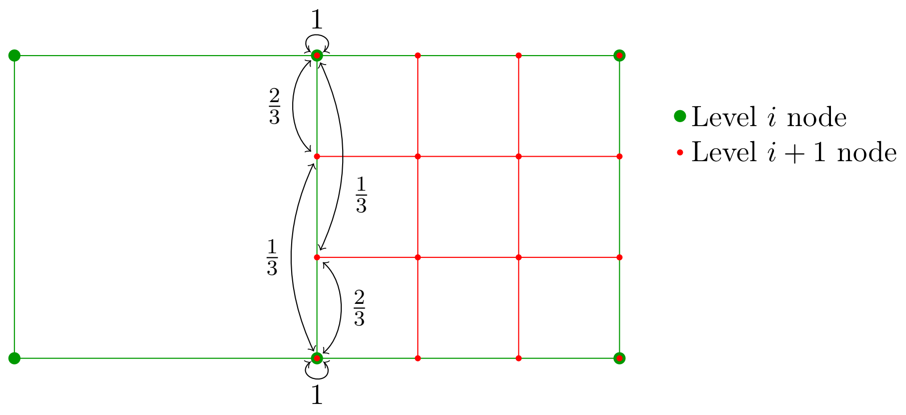


Fig. 3.5: Linear interpolation and restriction weights for hanging nodes at inter-level boundary.

Table 3.2: Event and action for TTransport calculations in the grid adapter GridEventHandle2TemperatureTransportAdapter with interpolation and restriction of T and TTransport.

Event	Action
touchVertexFirstTime(...)	resetTTransport
startStepsDown(...)	interpolateVelocities interpolateT
enterElement(...)	accumulateTTransport
finishedStepsUp(...)	restrictTTransport
touchVertexLastTime(...)	updateTemperatureWithExplicitEuler

GridAbstractFluidChemicalCell. The former is basically the same as the trivialgrid vertex with additional methods for interpolation and restriction. The latter inherits from the AbstractFluidCell class and adds no further functionality so far. Nevertheless it is implemented so that it can be extended more easily.

The grid has a new set of adapters, which need to be extended to use the new vertex and cell type. The numerical treatment of the equations remains the same and thus we do'nt discuss the modifications for most of the existing adapters, since they do not deal with the temperature.

The adapter that is responsible for TTransport in the energy Eq. (2.43) is called GridEventHandle2TemperatureTransportAdapter, see Section 3.5. Table 3.2 shows the relevant calls to the vertices during a grid traversal in this adapter. When stepping down from a coarser level, we need the temperature at every node and therefore at the hanging nodes as well. Thus, it is interpolated as well as the velocities. Afterwards, all degrees of freedom are available and the cell-wise accumulation of TTransport is called. When all finer cells have been treated with, the results have to be restricted onto the coarser level using restrictTTransport. The temperature is then updated with an explicit Euler method just like for the trivialgrid vertex.

Another adapter that needs modification is the GridEventHandle2FAdapter which handles the calculation of F including the Boussinesq approximation in the momentum equation (2.42). There the interpolateT method is called on stepping down (startStepsDown) along with the interpolation of the velocity. No restriction needs to be made in this adapter for the temperature, since no updates are calculated for the temperature and this degree of freedom is *used* only.

In Section 4.4 we discuss a numerical phenomena when using this type of grid.

3.8 Scenario

So far we dealt with the equations and their implementations. Now we turn to the scenarios which are responsible for assigning boundary types and boundary conditions to the boundary numbers given by the geometry. Accordingly a scenario deals with the initial values and the boundary values for Dirichlet boundaries. Note that geometry and scenario are two different things. Geometry is a description of the domain telling whether a voxel is inside or outside of the domain and defining numbers for every boundary. In contrast to that a scenario takes these numbers and uses the information to deal with actual values.

So far initial values and boundary conditions had to be specified for the velocity $\vec{u}$ only. This was done in a `FluidScenario`. Now it is necessary to define these values for the temperature – resulting in a `HeatScenario` – as well, but *only* when the `chemical` component is used. Furthermore, it should be possible to combine any `FluidScenario` with any `HeatScenario` to avoid code duplication.

Therefore, we chose to implement a *Decorator* pattern, where a *Decoratable* class (`FluidScenario`) can be decorated by a *Decorator* (`HeatScenario`) (Fig. 3.6). The interface `HeatScenario` inherits from the interface `FluidScenario` allowing the usual overloading techniques and so on. Additionally, it holds a `FluidScenario` as an attribute, thus making it possible to "add" a concrete `FluidScenario`. This is the great advantage we've been aiming for, since this allows to combine any `FluidScenario` with any `HeatScenario` – even decorating a `HeatScenario` with another `HeatScenario` is possible. Note that every `HeatScenario` itself needs to implement the interface of the `FluidScenario`. Since we want that calls to the actual `FluidScenario` – like `getVelocityForBoundaryNode()` – are delegated to the corresponding attribute `_fluidScenario`, the `FluidScenario` interface is implemented by forwarding the calls to the *Decoratable.*

The big picture is as follows:

- The adapter `TrivialgridGeometryHandle2FluidScenarioAdapter` holds an instance of `HeatScenarioAdapterImplementation` iff the `chemical` component is compiled. When there is a call to `createDegreeOfFreedom` (for creating a vertex or cell) the corresponding initial values need to be set for both, the `fluid` degrees of freedom and the temperature as a `chemical` degree of freedom.
- The latter are set in the `setDataForVertexChemical` and `setDataForCellChemical` methods of the `HeatScenarioAdapterImplementation` class, respectively. These methods issue calls to the `HeatScenario` to get the temperature boundary types and to get initial and boundary temperature values for Dirichlet boundaries. Another call is made to the parent class methods `setDataForVertex` and `setDataForCell`, which sets the velocity $\vec{u}$ and deals with the rest of the `fluid` boundaries and degrees of freedom.

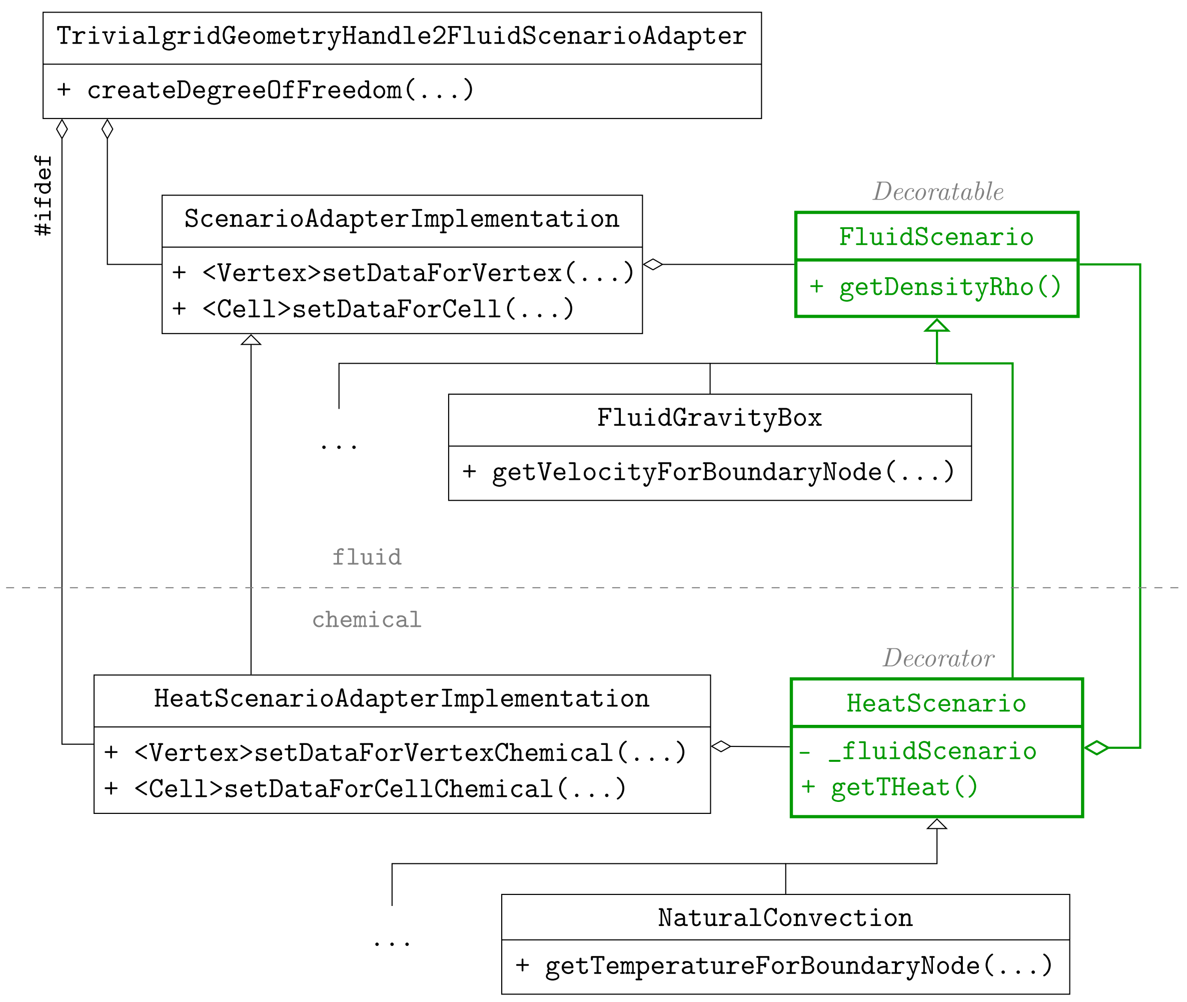

TrivialgridGeometryHandle2FluidScenarioAdapter
+ createDegreeOfFreedom(...)
#ifdef
ScenarioAdapterImplementation
+ <Vertex>setDataForVertex(...)
+ <Cell>setDataForCell(...)
Decoratable
FluidScenario
+ getDensityRho()
...
FluidGravityBox
+ getVelocityForBoundaryNode(...)
fluid
chemical
Decorator
HeatScenarioAdapterImplementation
+ <Vertex>setDataForVertexChemical(...)
+ <Cell>setDataForCellChemical(...)
HeatScenario
- _fluidScenario
+ getTHeat()
...
NaturalConvection
+ getTemperatureForBoundaryNode(...)

Fig. 3.6: UML class diagram for `HeatScenario` with Decorator Pattern.

- Since every `HeatScenario` is a `FluidScenario` the `ScenarioAdapterImplementation` and the `HeatScenarioAdapterImplementation` basically hold the same scenario – the constructor of `HeatScenarioAdapterImplementation` passes the `HeatScenario` to the constructor of `ScenarioAdapterImplementation` casting it to a `FluidScenario`. Thus, the above mentioned calls end up in the `HeatScenario` where they are delegated to the `_fluidScenario`.

A complete list of all `FluidScenario` scenarios (Fig. 3.7) and all `HeatScenario` scenarios (Fig. 3.8) shows the scenarios that can be decorated or used as Decorator. Due to the pile of information – this is a *complete* list – these are hard to visualize. For the usage in this thesis the most important scenarios are: `GravityBox`, `FlatPlateInParallelFlow` and `FluidPreCICEColdLeg` and all `HeatScenario` implementations of course. The abstract

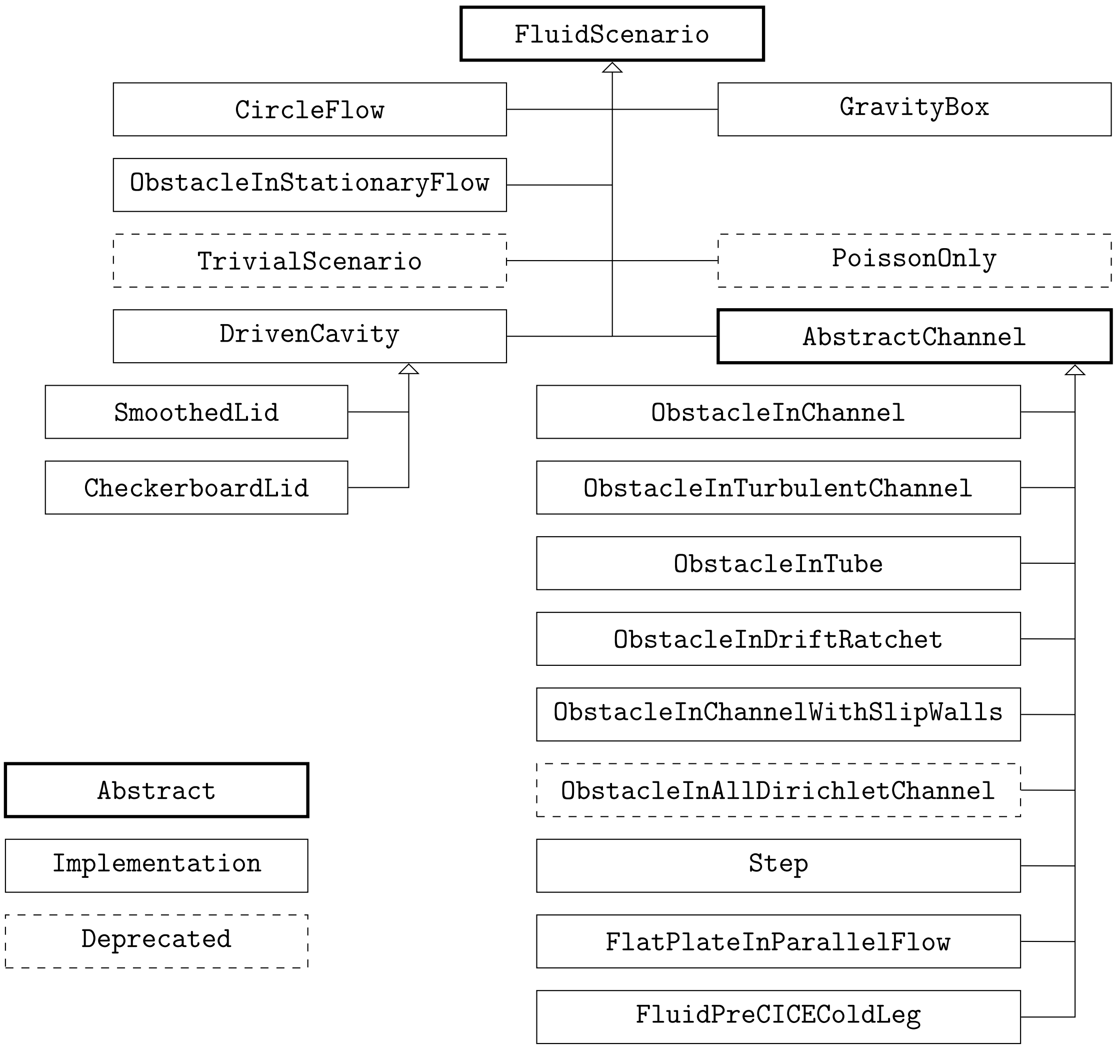


Fig. 3.7: UML class diagram for all `FluidScenario` scenarios in the `fluid` component.

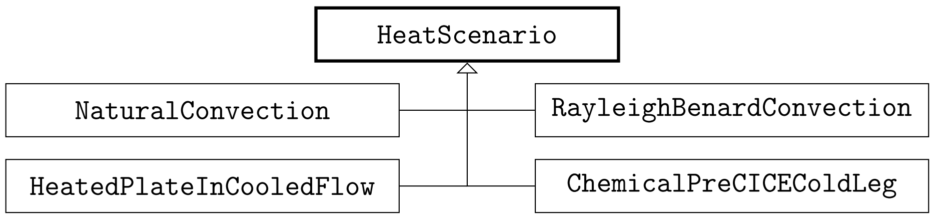


Fig. 3.8: UML class diagram for all `HeatScenario` scenarios in the `chemical` component.

scenario `AbstractChannel` provides useful methods and attributes that are needed when working with external (driven) flow having an inlet and/or an outlet.

3.9 Configuration

Peano is a `C++` framework consisting of different components which extend each other or accomplish different tasks throughout a simulation of any kind. SCons[2] may be used for compiling and a typical call to SCons reads as follows

```
» scons target=fluid-chemical dim=3 build=release
```

where the `target` is a tag which is known to SCons, so that the corresponding components are compiled. The `dim` argument specifies the dimension d of the system. Using `debug` instead of `release` for the `build` argument is recommended when a component is in development.

In order to run a simulation with Peano one needs the executable and a configuration file. Any input the user provides is stored in this XML file, which is the one and only argument that is passed on to the executable as in the statement below.

```
» ./peano-fluid-chemical configuration.xml
```

Like any other XML file the configuration consists of different tags, their attributes and subtags. An example for a complete configuration file for a `chemical` simulation can be found in Figure 3.9.

Following there is a list of attributes of the `chemical` tag (the brackets show an example):

- `name` (`"natural-convection"`), identifier of the `HeatScenario`,
- `t-heat` (`"21.0"`), temperature for boundary of type `HEAT_DIRICHLET_HEAT`,
- `t-cool` (`"19.0"`), temperature for boundary of type `HEAT_DIRICHLET_COOL`,
- `t-initial` (`"20.0"`), temperature at time t_0 (initial value) and reference temperature T_∞ for Boussinesq approximation (2.40),
- `thermal-conductivity-kappa` (`"5.97e-1"`), fluid parameter thermal conductiviy κ (used for calculation of α (2.36)),
- `specific-heat-at-constant-pressure-c_p` (`"4184.0"`), fluid parameter specific heat at constant pressure c_p (used for calculation of α, (2.36))
- `thermal-expansion-beta` (`"2.1e-4"`), fluid parameter thermal expansion β (used in Boussinesq approximation in the momentum equation, (2.17))

[2] *SCons: A Software Construction Tool*, see `http://www.scons.org/`

- ◦ `plot-chemical-data-at-specific-point` (`"yes"`), optional, flag to indicate whether or not Peano should write the chemical data of a specific point to a file in *every* time step (not only in the VTK output); if set to `"yes"` user has to provide `specific-point-for-chemical-data-xI` in the dimension of the simulation, and
- ◦ `specific-point-for-chemical-data-xI` (`"0.05"`) , coordinates of the point whose data should be plotted, `I` $\in [0, \dots, d]$; coordinates of a vertex (exactly).

The tags that have the ◦ symbol instead of the • symbol are optional. But as stated above, the specification of `specific-point-for-chemical-data-xI` is mandatory iff `plot-chemical-data-at-specific-point` is set to `"yes"`.

Next we list (not exhaustively) the attributes of the `fluid` tag:

- • `name` (`"gravity-box"`), identifier of the `FluidScenario`,
- • `rho` (`"1000.0"`), density ρ,
- • `eta` (`"1e-3"`), dynamic viscosity η,
- ◦ `velocity-mean-value` (`"1.0"`), inflow velocity $\vec{u}_\infty$, also used for calculation of the Reynolds number Re,
- ◦ `velocity-profile` (`"parabola"`), profile of velocity inflow at inlet; either `"const"` or `"parabola"`,
- ◦ `Re` (`"19.0"`), Reynolds number Re; has to be supplied when there is an inlet and a inflow velocity,
- ◦ `gravity-vector-xI` (`"19.0"`), gravity vector $g \in \mathbb{R}^d$, `I` $\in [0, \dots, d]$; only gravity in one of the axis directions is allowed,
- ◦ `maximum-elevation` (`"19.0"`), maximum elevation in direction of gravity; used for hydrostatic pressure correction; only used, when gravity is set and non-zero,
- ◦ `inlet-pressure` (`"100.0"`), reference pressure that acts on the inlet; can only be specified when `velocity-mean-value` is not, and
- ◦ `outlet-pressure` (`"10.0"`), reference pressure that acts on the outlet.

Note that this is *not* a complete list of all `fluid` tags that can or have to be specified. For more information on these tags see [11] and the documentation of Peano.

Additionally to this two tags there are different others in a configuration file controlling other components of Peano:

- • `domain` – control offset and length of the domain in each dimension
- ◦ `geometry` – geometric primitives that can be combined to give the domain Ω
- • `trivialgrid` – number of cells in each dimension
- • `ode` – configuring the solver for the time dependent update of the differential equation

- `solver` – tolerances and types of linear solvers (for the Poisson Pressure Equation)
- `plotter` – type of output

Note that the `domain` tag provides a superset of the actual domain Ω, where we want to do the computations. The latter is specified by a combination of `geometry` tags. An example configuration file for a `chemical` simulation using the `NaturalConvection` scenario is shown below (Fig. 3.9).

3.10 freesteam

freesteam[3] is an open source `C` implementation of international-standard IAPWS-IF97[4] steam tables from the International Association for the Properties of Water and Steam (IAPWS). The `chemical` component of Peano uses this library in order to calculate most of the fluid parameters (density ρ, dynamic viscosity μ, kinematic viscosity ν, specific heat at constant pressure c_p, thermal conductivity κ, thermal diffusivity α, coefficient of thermal expansion β). Note that this is a table for *water* and *steam* only!

All the parameters can be calculated by freesteam by supplying an ambient pressure p and a temperature T. The user denotes these two parameters as an attribute in the `chemical` tag

- `use-freesteam="yes"` to indicate that freesteam should be used
- `freesteam-pressure="1e5"` defines an ambient pressure of 1 bar
- `freesteam-temperature="293.15"` uses a reference temperature of 293.15 K

Note that the pressure is given in Pascal and the temperature in Kelvin and that all computed values are constant in space and time. Also note the necessity to define dummy values for `rho` and `eta` in the `fluid` tag for technical reasons. They are not be used for any other calculation than the computation of the Reynolds number. The definition of c_p, α and β can be omitted in the `chemical` tag of the configuration file if freesteam is used.

In order to compile Peano using freesteam, one needs to use SCons with the target `fluid-chemical-with-freesteam`. Note that freesteam has to be located in the search path as a library.

[3]see `http://freesteam.sourceforge.net/`
[4]see `http://www.iapws.org/`

```xml
<?xml version="1.0"?>

<configuration>

  <run-trivialgrid-fluid-chemical>
    <experiment-name> Configuration Example </experiment-name>

    <domain x0="0.0" x1="0.0" x2="0.0"
            h0="1.0" h1="1.0" h2="1.0" />

    <trivialgrid>
      <number-of-cells nx0="20" nx1="20" nx2="20" />
    </trivialgrid>

    <geometry name="hexahedron"
              geometry-base-number="0"
              invert="false">
      <bounding-box h0="1.0" h1="1.0" h2="1.0" />
    </geometry>

    <solver name="PETSc"
            type="CG"
            max-iterations="200"
            preconditioner="JACOBI"
            relative-tolerance="1e-7"
            absolute-tolerance="1e-7"
            divergence-tolerance="1e5" />

    <ode solver="euler-explicit"
         start-time="0.0"
         end-time="10.0"
         tau="0.1"
         number-of-time-steps="100"
         print-delta="100" />

    <chemical name="natural-convection"
              t-heat="1.0"
              t-cool="0.0"
              t-initial="0.5"
              thermal-conductivity-kappa="1e-4"
              specific-heat-at-constant-pressure-c_p="1.0"
              thermal-expansion-beta="2.1e-4" />

    <fluid name="gravity-box"
           gravity-vector-x0="0.0"
           gravity-vector-x1="-9.81"
           gravity-vector-x2="0.0"
           inlet-dimension-x0="0.0"
           inlet-dimension-x1="0.0"
           inlet-dimension-x2="0.0"
           inlet-offset-x0="0.0"
           inlet-offset-x1="0.0"
           inlet-offset-x2="0.0"
           velocity-mean-value="1.0"
           initiate-velocity-everywhere="no"
           velocity-profile="parabola"
           random-noise-weight="0.0"
           characteristic-length="1.0"
           Re="1.0"
           eta="1.0"
           rho="1.0"
           adjustment-factor="1.0"
           element-type="d-linear"
           use-divergence-correction="no" >
    </fluid>

    <plotter name="vtk"
             path="./"
             filename="output-prefix"
             use-standard-file-name-extension="yes"
             use-binary-format="no"
             plot-leaves-only="yes"
             plot-vertex-type="yes" />

  </run-trivialgrid-fluid-chemical>

</configuration>
```

Fig. 3.9: Example configuration file for a three-dimensional `trivialgrid` simulation using the `chemical` component with `gravity-box` as `FluidScenario` and `natural-convection` as `HeatScenario`.

4 Validation

Before tackling a real scenario, one should validate both the models that are used to simplify the reality and the implementation which is prone to bugs and human errors. Since it is not always possible to produce real world data, one cannot always compare the results quantitatively. Therefore, we have two scenarios – *Natural Convection with Heated Lateral Walls* (Section 4.1) and *Rayleigh-Bénard Convection* (Section 4.2) – both giving a qualitative comparison. The ideas for these scenarios have been taken from Griebel [6]. Note that Peano uses a finite element method (see [1, 5]), whereas Griebel's implementation is based on a finite difference method, so that a comparison between our results and Griebel's needs to take this into account. Finally, there is the scenario *Flat Plate in Parallel Flow* (Section 4.3) allowing us to do a quantitative analysis on boundary layers. We also describe problems and phenomena that we encountered throughout testing.

If not denoted explicitly the simulations and tests in this section have been performed using the `trivialgrid` component of Peano, since the `grid` (see Section 3.7) implementation was not available yet. Only Section 4.4 uses the `grid` component and shows a phenomena that occurs only for simulations with a level difference of at least one.

4.1 Natural Convection with Heated Lateral Walls

Natural convection is also called buoyancy-driven flow, since the flow evolves only due to density differences in the fluid. These density differences are caused by temperature differences, which again are caused by e.g. cooling or heating the fluid at the boundary. Note that there is no internal heat generation since the right hand side of the energy equation is zero (see Eq. (2.43) on Page 11). Since we are using the occurring buoyancy forces, a proper acceleration such as gravity is vital. This is especially true when using the Boussinesq approximation, which assumes that density is constant except in buoyancy terms in the momentum equation (Eq. (2.42) on Page 11).

The Rayleigh number,

$$\mathrm{Ra} \;=\; \frac{\|g\|\,\beta\,(T_H - T_C)\,L^3}{\alpha\nu}, \tag{4.1}$$

is an important dimensionless number for buoyancy-driven flows and highly dependent on both, the fluid parameters (such as thermal diffusivity α) and the scenario parameters

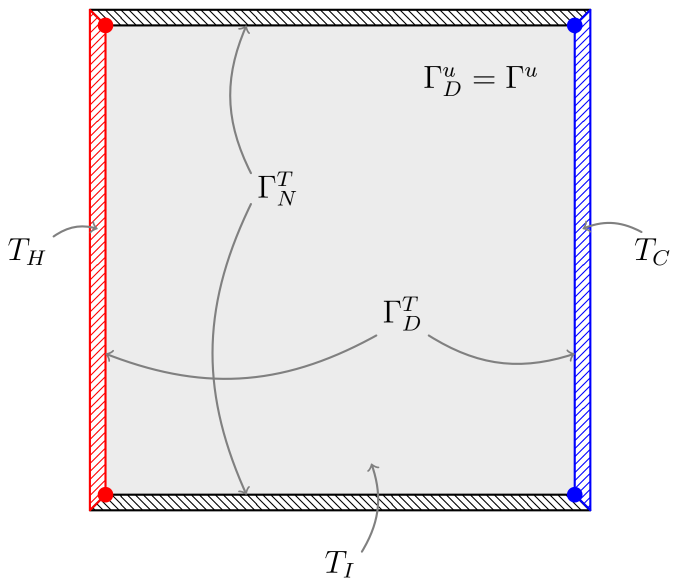


Fig. 4.1: Natural Convection with Heated Lateral Walls, scenario setup.

(e.g. the gravity g). Being below a critical value, it leads to heat being transferred mainly by diffusion, whereas the heat is transferred by convection, when the Rayleigh number is above this value. An example for the first case can be found in Section 4.1.1, whereas the latter case is shown in Section 4.1.2.

In the first scenario *Natural Convection with Heated Lateral Walls* we have a closed, square box with a heated, a cooled and two adiabatic (insulated) walls, which is filled with a fluid (see [6]).Fig. 4.1 shows the setup for this scenario. The left wall is heated at temperature T_H, whereas the right wall is cooled at T_C. The fluid itself and the adiabatic walls at the top and the bottom of the box have an initial temperature T_I.

The initial temperature T_I is not really important for this scenario, since the mean temperature $\overline{T}_\Omega$ – averaged over the whole domain – is the average of the heating and the cooling temperature,

$$\overline{T}_\Omega \quad = \quad \frac{T_H + T_C}{2},$$

for all fluids with $\alpha > 0$ when the steady state is reached. Since the transients and therefore the computational times can become very long, one can at least reduce the time until that state is reached by setting $T_I = \overline{T}_\Omega$.

In Peano we use the fluid-scenario `gravity-box`, which sets no-slip Dirichlet boundary conditions for the velocity at all boundaries ($\Gamma^{\vec{u}} = \Gamma^{\vec{u}}_D$):

$$\vec{u}(\vec{x},t) \quad = \quad 0\,\frac{\mathrm{m}}{\mathrm{s}} \qquad \forall\,(\vec{x},t) \in \Gamma^{\vec{u}} \times [t_0, t_{end}]\,.$$

The chemical-scenario is `natural-convection` setting Dirichlet boundary conditions for the temperature at the left and right wall (Γ_D^T) and Neumann boundary conditions at the top and bottom wall (Γ_N^T), respectively. Note that the vertices in the very corners technically belong to the temperature Dirichlet boundary Γ_D^T, as suggested by the marks in Fig. 4.1.

In the following we take a look at different fluid parameters. As it turns out, the evolving flow and the computational challenges are highly dependent on these parameters. Griebel presents two types of fluids, which are discussed first. Afterwards, we take a look at two less viscous fluids, one that is a little more viscous than water and water itself.

4.1.1 Diffusive Heat Transfer (Griebel 1)

The first parameters that Griebel presents for this scenario, describe an artificial fluid with high viscosity and good heat diffusivity (conductivity). Further on they are called *Griebel 1*-parameters:

$$\begin{aligned} \mathrm{Pr} &= 7, \quad \mathrm{Re} = 985.7, \quad \mathrm{Ra} = 140, \\ \beta &= 2.1 \times 10^{-4}, \quad T_H = 1, \quad T_C = 0, \\ g &= \left(0, -9.706 \times 10^{-2}\right)^T, \quad L = 1. \end{aligned} \tag{4.2}$$

Pr is the Prandtl number, Re is Reynolds number, the Rayleigh number Ra, β is the coefficient of thermal expansion, g the gravity and L the characteristic length. Note that Griebel uses the dimensionless formulation of the NSE and energy equation and therefore uses dimensionless parameters as well. Especially, when looking at the gravity g this might seem strange. If wanted, one can change it to be the earth's gravity $g \approx (0, -9.81)^T \,\mathrm{m\,s^{-2}}$, but then one would need to adapt some other parameter in order to maintain the Rayleigh number. Also note that Peano allows to choose the axis along which the gravity acts. This is done by specifying a gravity vector in the configuration file (`gravity-x0`, `gravity-x1`). Since we want the gravity to act "downwards" in the pictures, we need to specify a negative value along the y-axis.

The characteristic length given by Griebel is without any unit as well, but use the same value for our computations, so that we have:

$$L = 1\,\mathrm{m},$$

leading to a square domain with edge length L.

Another thing to mention is that we use temperatures without any units, since only the difference is interesting in the Boussinesq approximation (see Eq. (2.40) on Page 11) and the energy equation (2.43) only uses derivatives of the temperature. Hence, it would not make any difference if we used $T_H = 20\,°\mathrm{C}$ or $T_H = 293\,\mathrm{K}$ instead of $T_H = 1$ as long as the reference temperature T_∞ is understood as parameter without unit as well. In Peano T_I is used as the reference temperature for the Boussinesq approximation.

The Reynolds number Re itself is not really significant, since we cannot tell anything about the characteristic velocity u_∞ beforehand, as there is no external flow involved. Therefore, we typically set

$$u_\infty \;=\; 1\,\frac{\mathrm{m}}{\mathrm{s}}$$

for our simulations.

If wanted, one could calculate the time dependent Reynolds number in each time step, by setting $u_\infty = \|\vec{u}_{max}\|$,

$$\mathrm{Re}_t \;=\; \frac{L\,\left\|\vec{u}^{\,t}_{max}\right\|}{\nu}, \tag{4.3}$$

where $\vec{u}^{\,t}_{max}$ is the 2-norm largest velocity in the domain at time t and $\vec{u}_{max} = \vec{u}^{\,t_{end}}_{max}$. Nevertheless, since Griebel provided a global Reynolds number, we use it to specify additional parameters (such as the kinematic viscosity ν) that are necessary for the Peano configuration file:

$$\begin{gathered} T_I \;=\; T_\infty \;=\; 0.5, \\ \kappa \;=\; 1.4493\times10^{-4}\,\tfrac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}, \quad c_p \;=\; 1\,\tfrac{\mathrm{J}}{\mathrm{kg}\cdot\mathrm{K}}, \\ \eta \;=\; 1.0145\times10^{-3}\,\mathrm{Pa}\cdot\mathrm{s}, \quad \rho \;=\; 1\,\tfrac{\mathrm{kg}}{\mathrm{m}^3}, \quad u_\infty \;=\; 1\,\tfrac{\mathrm{m}}{\mathrm{s}}, \end{gathered} \tag{4.4}$$

where the thermal conductivity κ, the specific heat at constant pressure c_p, the dynamic viscosity η, the density ρ are set to fit the Griebel values in Eq. (4.2).

These parameters lead to the kinematic viscosity ν,

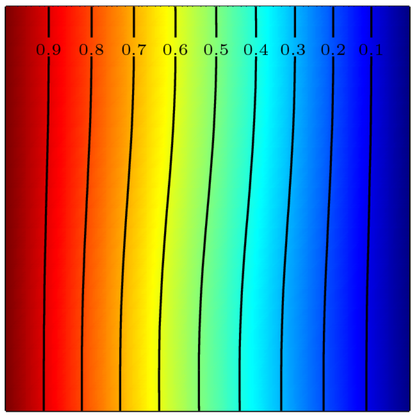
$$\nu \;=\; \frac{\eta}{\rho} \;=\; 1.0145\times10^{-3}\,\frac{\mathrm{m}^2}{\mathrm{s}},$$

and thermal diffusivity α,

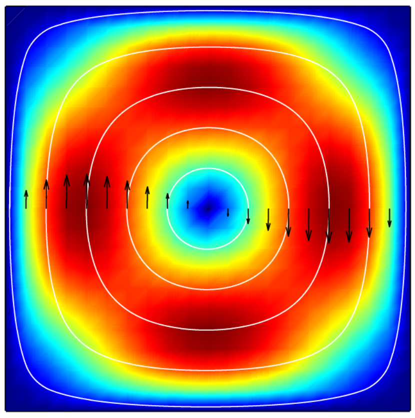
$$\alpha \;=\; \frac{\nu}{\mathrm{Pr}} \;=\; \frac{\kappa}{c_p\rho} \;=\; 1.4493\times10^{-4}\,\frac{\mathrm{m}^2}{\mathrm{s}}.$$

Using these parameters we have $\mathrm{Ra} \approx 140$.

As there is no external flow and the temperature difference is small, the evolving flow is quite slow. Thus, it takes a long time until the steady state is reached. For this scenario we have $t_{end} = 3000\,\mathrm{s}$, leading to long computational running times, depending on the mesh and time step size τ of course.



(a) Isolines for temperature, color-coded temperature, $T_H = 1$, $T_C = 0$.



(b) Streamlines of flow, color-coded velocity magnitude, $\|\vec{u}_{max}\| = 7.94 \times 10^{-5}\,\mathrm{m\,s^{-1}}$.

Fig. 4.2: Natural Convection with Heated Lateral Walls, Griebel 1, $t = 3000\,\mathrm{s}$, $\tau = 0.1$, 20×20 cells, temperature isolines and velocity streamlines.

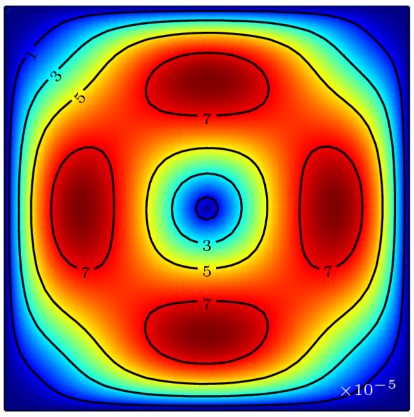
We tested on three different regular grids (trivialgrid only) with 20×20, 50×50 and 100×100 cells (given in number of cells per dimension), leading to square cells with edge lengths 5 cm, 2 cm and 1 cm, respectively. For each grid several time step sizes τ have been used. A list of all runs that have been performed can be found at the end of this Section in Table 4.1 on Page 42. Now everything is set up for our first run of Peano with the chemical component.

In Fig. 4.2 we see the steady state for the temperature and the velocites at $t_{end} = 3000$ for a 20×20 mesh with time step size $\tau = 0.1$. These show that the heat is transported mainly by diffusion (see Fig. 4.2(a)) and only slightly by convection – note that the lines are slightly bend to the right at the top and left at the bottom, respectively. This is exactly what is to be expected, when using a low Rayleigh number $\mathrm{Ra} \approx 140$.

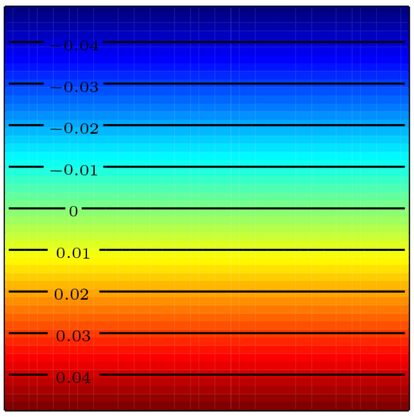
Fig. 4.2(b) shows the streamlines of the flow. There is one current flowing through the whole domain, rising at left hand side, being pushed to the right, where it drops down due to rising density and is pushed back to the left again. The color in the background shows the magnitude of the velocity – which is $\|\vec{u}(\vec{x}, t)\|$ – at each point $\vec{x}$ of the domain at time t. The minimal velocity (dark blue) is $0\,\mathrm{m\,s^{-1}}$ for all upcoming Figures, whereas the maximal velocity (dark red) is given in the caption of each Figure. The flow presented in Fig. 4.2(b) has the following maximal velocity:

$$\|\vec{u}_{max}\| \quad = \quad 7.94 \times 10^{-5}\,\frac{\mathrm{m}}{\mathrm{s}}.$$

Next we take a look at the isolines of the velocities and the pressure for the steady state on a 50×50 mesh (Fig. 4.3). The high velocity spots ($\|\vec{u}\| \geq 7 \times 10^{-5}\,\mathrm{m\,s^{-1}}$) on the



(a) Velocity isolines, color-coded velocity, $\|\vec{u}_{max}\| = 7.907 \times 10^{-5}\,\mathrm{m\,s^{-1}}$.



(b) Isolines of pressure, color-coded pressure.

Fig. 4.3: Natural Convection with Heated Lateral Walls, Griebel 1, $t = 3000\,\mathrm{s}$, $\tau = 0.05$, 50×50 cells, isolines of velocities and pressure.

left and right in Fig. 4.3(a) are located at an approximate distance of 20 cm to the boundary. This shows that the viscosity – and therefore the friction at the wall due to zero velocities – has more influence on the flow than the occurring buoyancy forces due to density changes. This changes as soon as the fluid gets less viscous. Then the boundary layer – the distance between the boundary and the corresponding highest velocity – gets thinner.

Fig. 4.3(b) shows the color-coded pressure with pressure isolines. As the pressure is a cell degree of freedom in Peano, the values are calculated in the center of the cell. Therefore, we use the a superscripted C to indicate the usage of a value computed by Peano. Also note that we do not set the pressure at any point in the domain. Since the momentum equation (2.42) only contains the pressure gradient absolute values are not necessary and only the differences play a vital role for Peano. One could add 1 bar to the Peano pressure to simulate an ambient pressure of 1 bar in the domain.

Here we have the following cell pressure values:

$$\begin{aligned} p^C_{max} &= 4.756 \times 10^{-2}, \\ p^C_{min} &= -4.756 \times 10^{-2}, \end{aligned}$$

at the bottom and top of the box, respectively.

The analytical hydrostatic pressure is:

$$p^{an}_s = \rho\,\|g\|\,h = 9.706 \times 10^{-2}\,\mathrm{Pa},$$

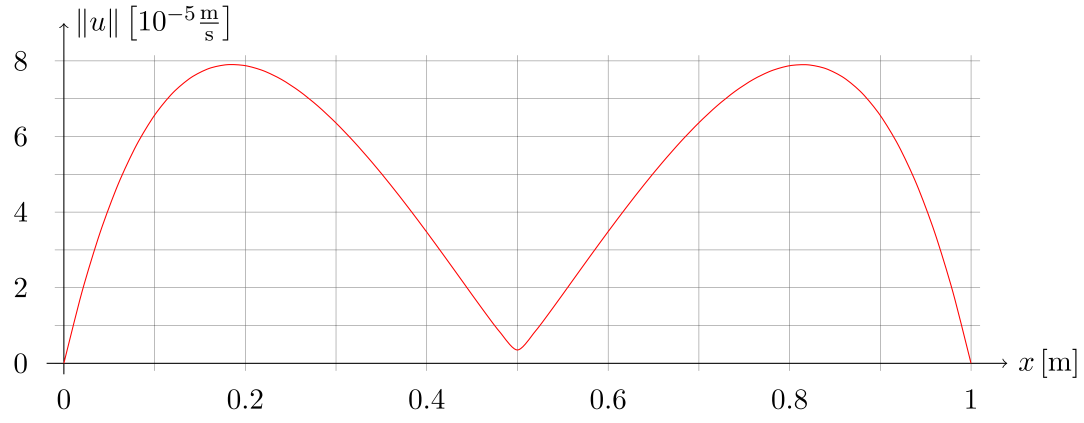


Fig. 4.4: Natural Convection with Heated Lateral Walls, Griebel 1, $t = 3000\,\mathrm{s}$, $\tau = 0.05$, 50×50 cells, velocity magnitude, line plot over horizontal center line at $y = 0.5\,\mathrm{m}$.

where we have $h = L$ for the height h of the water column. Since the pressure is calculated in the middle of the cell in Peano, we have to extrapolate the pressure from the cell center to the cell edges at the top and bottom. We get

$$p_s = \frac{n+1}{n}\left(\left|p^C_{max}\right| + \left|p^C_{min}\right|\right) = 9.702 \times 10^{-2}\,\mathrm{Pa} \approx p^G_s\,,$$

where $n = 50$ is the number of cells. There is no visible hydrodynamic pressure,

$$p^G_d = \frac{1}{2}\rho\,\|\vec{u}\|^2\,, \tag{4.5}$$

since $\|\vec{u}_{max}\|^2 = 6.3 \times 10^{-9}\,\mathrm{m\,s^{-1}}$ and these differences cannot be resolved by the color.

Finally, we have Fig. 4.4 showing absolute values of the velocity at time t_{end} for the horizontal line at $y = 0.5$.

So all in all we see good results that can be compared Griebel's results (Fig. 9.4 in [6]), where a 50×50 grid computation is shown. Note that we were able to qualitatively produce the same results by only using a 20×20 grid, but cannot compare the results quantitatively due to lack of data.

Table 4.1 shows the meshes and time step sizes that have been used for this scenario. Additionally, the maximal velocity is at time $t_{end} = 3000\,\mathrm{s}$ is given. The symbol Ψmeans that there was an error for this computation, e.g. when the solution of the PPE diverges due to an unstable time step size.

Table 4.1: Natural Convection with Heated Lateral Walls, Griebel 1, list of runs, $t_{end} = 3000\,\mathrm{s}$.

Mesh	τ	$\Vert\vec{u}_{max}\Vert$ $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$ at t_{end}
	1	7.951×10^{-5}
20×20	0.5, 0.1	7.938×10^{-5}
	0.01	7.937×10^{-5}
	0.2	Ψ
50×50	0.15	7.909×10^{-5}
	0.01, 0.05	7.907×10^{-5}
100×100	0.05	Ψ
	0.01, 0.005	7.901×10^{-5}

4.1.2 Convective Heat Transfer (Griebel 2)

We now take a look at another fluid – referred to as *Griebel 2* – which is less viscous and has smaller thermal diffusivity than the first fluid. The dimensionless parameters as presented in Griebel are:

$$\begin{aligned} \mathrm{Pr} &= 7, \quad \mathrm{Re} = 11\,063, \quad \mathrm{Ra} = 2 \times 10^5, \\ \beta &= 2.1 \times 10^{-4}, \quad T_H = 1, \quad T_C = 0, \\ g_\infty &= (0, -1.1005)^T, \quad L = 1. \end{aligned} \tag{4.6}$$

Like before, we use $T_I = \overline{T} = 0.5$ in Peano. In this run the non-dimensionless gravity $g = (0, -9.81)^T\,\mathrm{m\,s^{-2}}$ is used by transforming g_∞, applying the formula:

$$g_\infty = \left(0, -\frac{L\,\Vert g\Vert}{u_\infty^2}\right)^T, \tag{4.7}$$

as presented in [6]. This leads to $u_\infty = 2.98\,\mathrm{m\,s^{-1}}$, since we still use the characteristic length $L = 1\,\mathrm{m}$ for the height and width of the box. Again, u_∞ and the Reynolds number are not significant for our calculations, but they are used to reproduce the fluid that Griebel is using. Additionally, they are listed here, since Peano's configuration file needs these parameters.

Next, by setting:

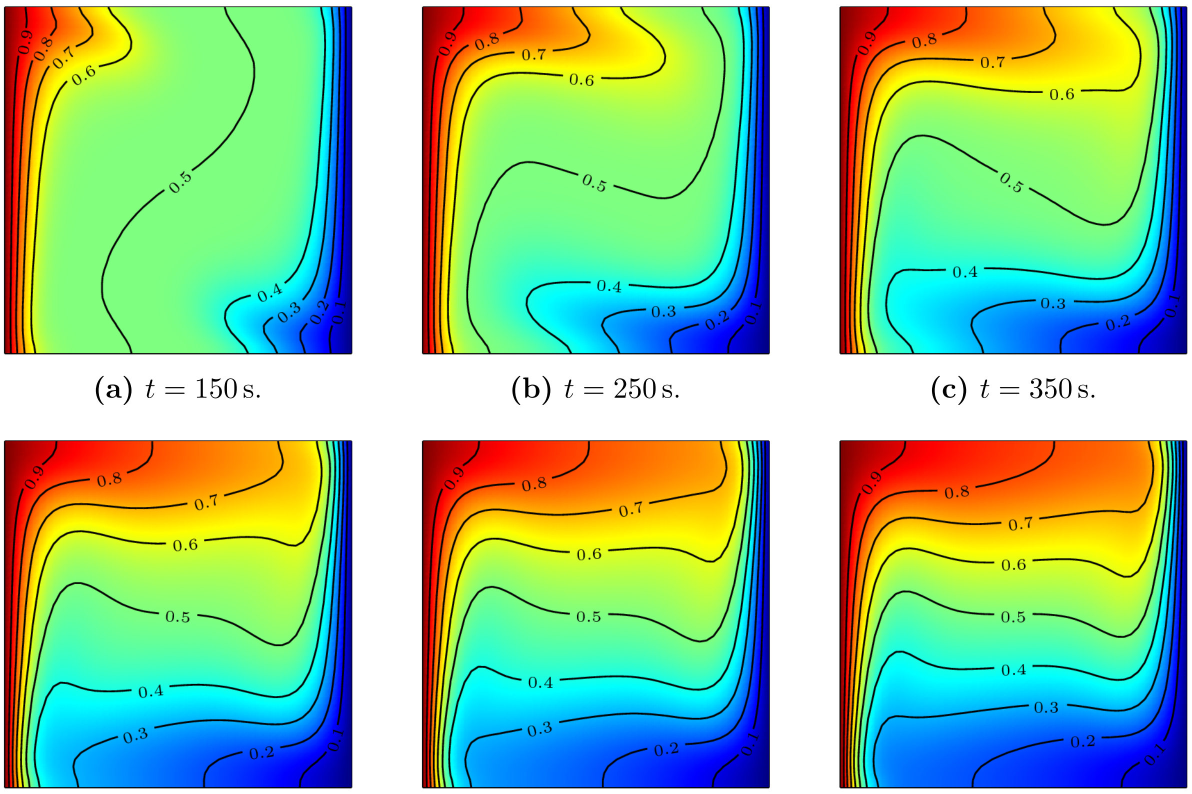
$$\rho = 1\,\tfrac{\mathrm{kg}}{\mathrm{m}^3}, \quad c_p = 1\,\tfrac{\mathrm{J}}{\mathrm{kg}\cdot\mathrm{K}}, \quad \nu = 2.67 \times 10^{-4}\,\tfrac{\mathrm{m}^2}{\mathrm{s}},$$

we finally get:

$$\begin{aligned} \eta &= \nu\rho = 2.67 \times 10^{-4}\,\mathrm{Pa\cdot s}, \text{ and} \\ \alpha &= \frac{\kappa}{c_p\rho} = 3.86 \times 10^{-5}\,\frac{\mathrm{m}^2}{\mathrm{s}}. \end{aligned}$$

As total simulation time we again use $t_{end} = 3000\,\mathrm{s}$ and a fixed time step size $\tau = 0.05$ on a 50×50 mesh (cell size $2\,\mathrm{cm} \times 2\,\mathrm{cm}$). Note that we have a much larger Rayleigh number for this fluid and scenario configuration than before, so that the heat is transferred mainly by convection.

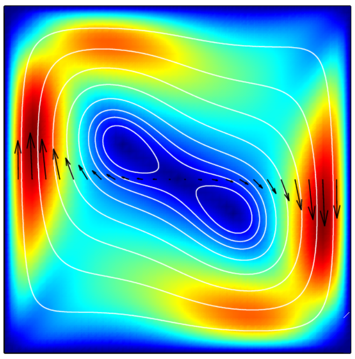
This can be seen in Fig. 4.5, where the temperature contour lines are plotted for times 150 s, 250 s, 350 s, 550 s, 750 s and 3000 s, respectively. Warm fluid is pushing to the right at the top of the box (Fig. 4.5(a) to Fig. 4.5(c)). As we've seen before, this problem is symmetric and therefore the same happens at the bottom of the box from the right to the left and takes the same time. The isoline for temperature $T = 0.5$ in Fig. 4.5(c) clearly shows that the warm fluid is dragged downwards by the cooled fluid. Note the development of the isolines for $T = 0.3$, and $T = 0.7$ at times 550 s, 750 s and 3000 s. Since heat was transferred by conduction before, there was no real boundary layer for the temperature. This time we have a moderate boundary layer that needs to be discretized fine enough by the mesh, which is shown later.

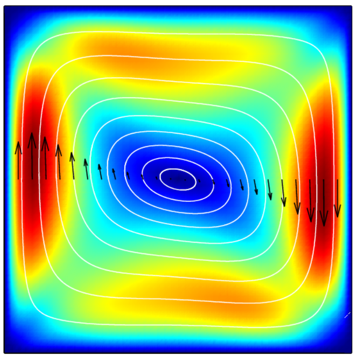


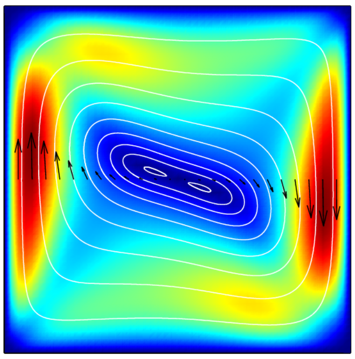
(a) $t = 150\,\mathrm{s}$. **(b)** $t = 250\,\mathrm{s}$. **(c)** $t = 350\,\mathrm{s}$.

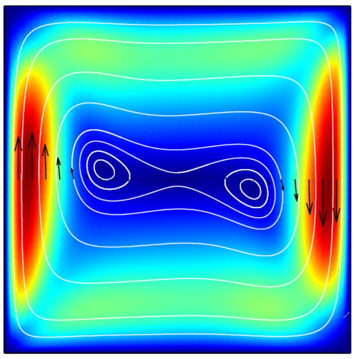
(d) $t = 550\,\mathrm{s}$. **(e)** $t = 750\,\mathrm{s}$. **(f)** $t = t_{end} = 3000\,\mathrm{s}$.

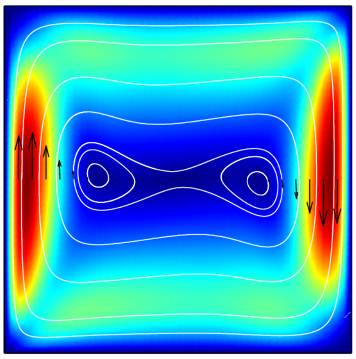
Fig. 4.5: Natural Convection with Heated Lateral Walls, Griebel 2, 50×50 mesh, $\tau = 0.05$, evolution of temperature, isolines with color-coded temperature.


(a) $t = 150\,\mathrm{s}$, $\|\vec{u}_{max}\| = 6.0 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.


(b) $t = 250\,\mathrm{s}$, $\|\vec{u}_{max}\| = 6.1 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.


(c) $t = 350\,\mathrm{s}$, $\|\vec{u}_{max}\| = 5.3 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.


(d) $t = 550\,\mathrm{s}$, $\|\vec{u}_{max}\| = 4.5 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.


(e) $t = 750\,\mathrm{s}$, $\|\vec{u}_{max}\| = 4.2 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

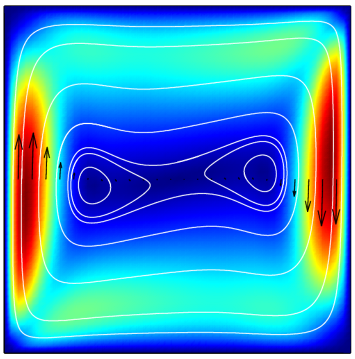

(f) $t = t_{end} = 3000\,\mathrm{s}$, $\|\vec{u}_{max}\| = 4.1 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

Fig. 4.6: Natural Convection with Heated Lateral Walls, Griebel 2, 50×50 mesh, $\tau = 0.05$, evolution of flow, streamlines with color-coded velocity magnitude.

Fig. 4.6(a) to Fig. 4.6(f) show the streamlines of the evolving flow at the same times as before. To be able to use the full range of color, the color-coded velocity magnitude is normalized to its own maximal velocity, which is written in the caption to be able to compare them to the other Figures (minimum is $0\,\mathrm{m\,s^{-1}}$ for all time steps of course). At the beginning we see that there form two internal loops within the outer flow. The left loop then moves downwards, the right upwards, melting together at time $t = 250\,\mathrm{s}$, before they finally split up again at time $t = 350\,\mathrm{s}$. Fig. 4.6(f) shows the steady state. Note that the last modifications of the flow take a long time to reach their final state even though the changes look very small – $t = 750\,\mathrm{s}$ vs. $t = 3000\,\mathrm{s}$. Also note that the speed of the flow is rising until time $t = 250\,\mathrm{s}$ and especially that there is high velocity at the top and bottom of the box when the fluid is pushed towards the right and left side for the first time, respectively. When comparing Fig. 4.6(f) to Fig. 4.2(b) one can

Table 4.2: Natural Convection with Heated Lateral Walls, Griebel 2, list of runs, $t_{end} = 3000\,\mathrm{s}$.

Mesh	τ	$\|\vec{u}_{max}\|$ $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$ at t_{end}
25×25	1	3.834×10^{-3}
	0.5, 0.1, 0.05	3.833×10^{-3}
50×50	0.75	4.09×10^{-3}
	0.5, 0.1, 0.05	4.089×10^{-3}
100×100	0.1, 0.05, 0.01, 0.001	4.091×10^{-3}
150×150	0.1	Ψ
	0.05, 0.01, 0.001, 0.005	4.091×10^{-3}
200×200	0.01, 0.005	4.091×10^{-3}
400×400	0.01, 0.005	4.092×10^{-3}

see that the fluid parameters lead to smaller velocity boundary layers. Especially the lower kinematic viscosity ν results in the fluid moving more easily and faster and the lower thermal diffusivity α corresponds to heat not being transferred as far into the box by conduction.

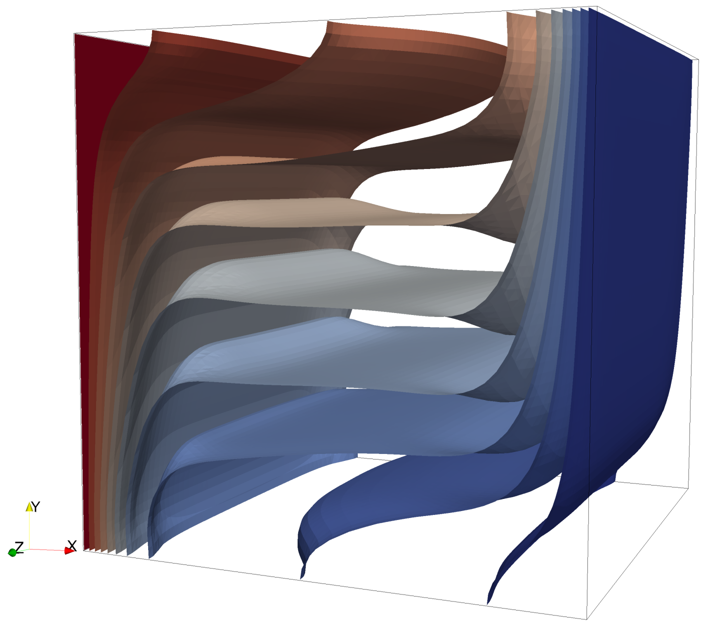
Table 4.2 shows the list of runs that have been performed for the Natural Convection with Heated Lateral Walls scenario with Griebel 2 fluid parameters. In general, all final flows show the same steady state where only the maximal velocity differs for some mesh resolutions. In contrast to the first fluid, it seems that there is quite a significant discretization error for the 25×25 grid. This can be led back to the smaller velocity boundary layers, which has been pointed out before.

Again, when comparing the results to Griebel's (using a 50×50 mesh, see Fig. 9.5 in [6]), we qualitatively get the same results.

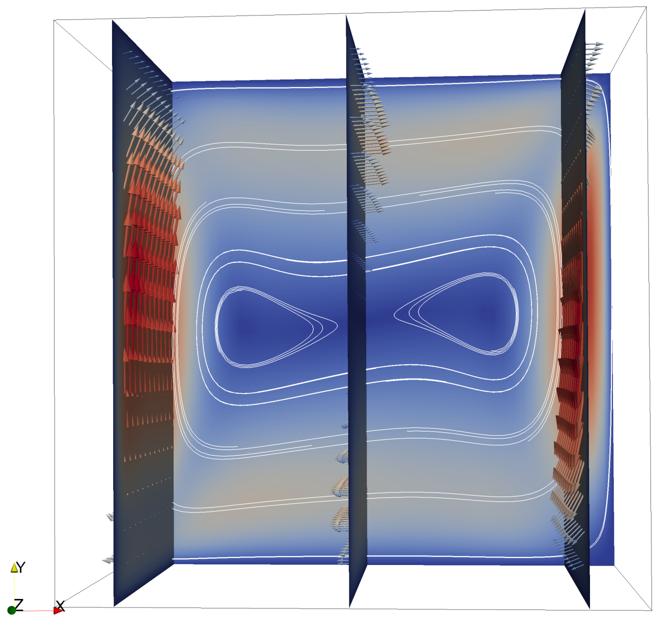
Fig. 4.7 shows a three-dimensional simulation on a $50 \times 50 \times 50$ mesh (cell size is $2\,\mathrm{cm} \times 2\,\mathrm{cm} \times 2\,\mathrm{cm}$). The temperature isosurfaces in Fig. 4.7(a) clearly show the influence of the front and back wall of the domain – holding no-slip Dirichlet boundaries. The fluid is slowed down at the boundary and thus the convective transport of the temperature is influenced accordingly. The streamlines in Fig. 4.7(b) are qualitatively very similar to Fig. 4.6(f). An overall maximal velocity of $\|\vec{u}_{max}\| = 4.2 \times 10^{-3}\,\mathrm{m\,s^{-1}}$ also indicates a good approximation of the flow, when comparing it to the two-dimensional simulations (see Table 4.2). But note that the flow *does* have three-dimensional effects as the maximal velocity in z-direction is:

$$w_{max} = 4.09 \times 10^{-4}\,\mathrm{m\,s^{-1}},$$

which is approximately one scale below the maximal velocity in both other directions.



(a) Temperature isosurfaces



(b) Streamlines ($z = 0.5$) with glyphs at three slices (x =0.1, 0.5 and 0.9)

Fig. 4.7: Natural Convection with Heated Lateral Walls, Griebel 2, $50 \times 50 \times 50$ mesh, $t = 3000\,\mathrm{s}$, adaptive τ computation, $\|\vec{u}_{max}\| = 4.17 \times 10^{-3}\,\mathrm{m\,s^{-1}}$, $u_{max} = 2.14 \times 10^{-3}\,\mathrm{m\,s^{-1}}$, $v_{max} = 4.17 \times 10^{-3}\,\mathrm{m\,s^{-1}}$, $w_{max} = 4.09 \times 10^{-4}\,\mathrm{m\,s^{-1}}$.

4.1.3 Artificial Fluid

Now we take a look at a fluid which is close to water, specified by the following parameters:

$$\begin{aligned} \text{Pr} &= 7, \quad L = 1\,\text{m}, \quad g = (0, -9.81)^T \tfrac{\text{m}}{\text{s}^2}. \\ \beta &= 2.1 \times 10^{-4} \tfrac{1}{\text{K}}, \quad T_H = 1, \quad T_C = 0, \quad T_I = 0.5, \\ \rho &= 1000 \tfrac{\text{kg}}{\text{m}^3}, \quad c_p = 4183 \tfrac{\text{J}}{\text{kg} \cdot \text{K}}. \end{aligned} \tag{4.8}$$

Next we set:

$$\eta = 1 \times 10^{-2}\,\text{Pa} \cdot \text{s},$$

and therefore define:

$$\begin{aligned} \nu &= \tfrac{\eta}{\rho} = 1 \times 10^{-5} \tfrac{\text{m}^2}{\text{s}}, \\ \alpha &= \tfrac{\nu}{\text{Pr}} = 1.4 \times 10^{-6} \tfrac{\text{m}^2}{\text{s}}, \\ \kappa &= \alpha c_p \rho = 5.9 \tfrac{\text{W}}{\text{m} \cdot \text{K}}. \end{aligned}$$

By assuming $u_\infty = 1\,\text{m}\,\text{s}^{-1}$ we get:

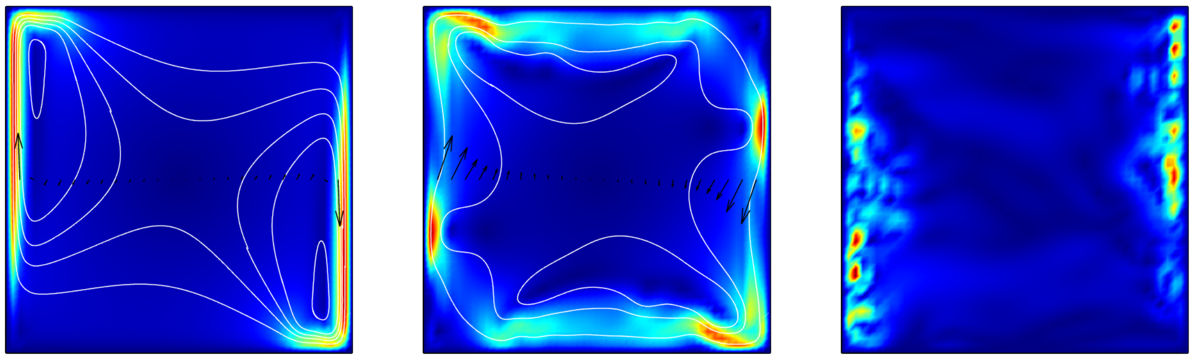
$$\text{Re} = 1 \times 10^5 \quad \text{and} \quad \text{Ra} = 1.47 \times 10^8.$$

The gentle reader immediately realizes that the difference to water is to be found in the dynamic viscosity η. We chose this artificial fluid to close the gap between Griebel 2 and water. For Griebel 2 we had $\nu = 2.67 \times 10^{-4}\,\text{m}^2\,\text{s}^{-1}$ and water has $\nu = 1 \times 10^{-6}\,\text{m}^2\,\text{s}^{-1}$, so that choosing $\nu = 1 \times 10^{-5}\,\text{m}^2\,\text{s}^{-1}$ is a reasonable value for this purpose. It led to some interesting phenomena, that are likely to occur when using water as fluid. For a discussion on water see Section 4.1.4. As total simulation time we have:

$$t_{end} = 10\,000\,\text{s}.$$

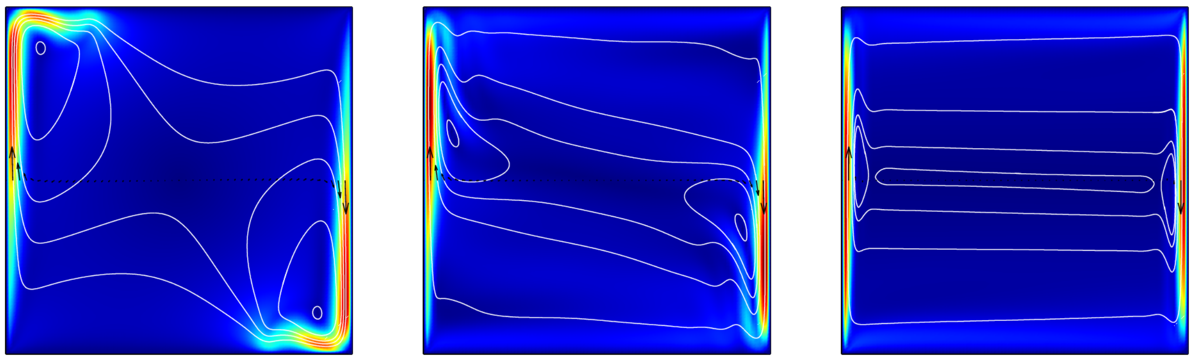
We've seen before that lower kinematic viscosity or thermal diffusivity lead to thinner thermal and flow boundary layers at the heating and cooling wall, respectively. Compared to the previously used fluids the viscosity and the diffusivity are smaller by approximately one scale for the current fluid.

Therefore, it is to be expected that a mesh resolution of 25 cells (cell size $4\,\text{cm} \times 4\,\text{cm}$) in each dimension is not enough for this configuration. And in fact even a cell size of $2\,\text{cm} \times 2\,\text{cm}$ (50×50 cells) is too coarse. This can be seen in Fig. 4.8. For the first shown time $t = 125\,\text{s}$ (Fig. 4.8(a)) everything looks fine, but after some more time has passed ($t = 375\,\text{s}$, Fig. 4.8(b)) the flow gets noisy. In Fig. 4.8(c) the streamlines are not even drawn any more since the flow obviously is not correct – in a phyiscal way – any more. Especially when looking at the high Rayleigh number, one might be mislead to believe that the flow is turbulent.



(a) $t = 125\,\mathrm{s}$, $\|\vec{u}_{max}\| = 6.8 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.
(b) $t = 375\,\mathrm{s}$, $\|\vec{u}_{max}\| = 7.3 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.
(c) $t = 10\,000\,\mathrm{s}$, $\|\vec{u}_{max}\| = 8.1 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

Fig. 4.8: Natural Convection with Heated Lateral Walls, Artificial Fluid, 50×50 mesh, $\tau = 0.05$, evolution of flow, streamlines with color-coded velocity magnitude.

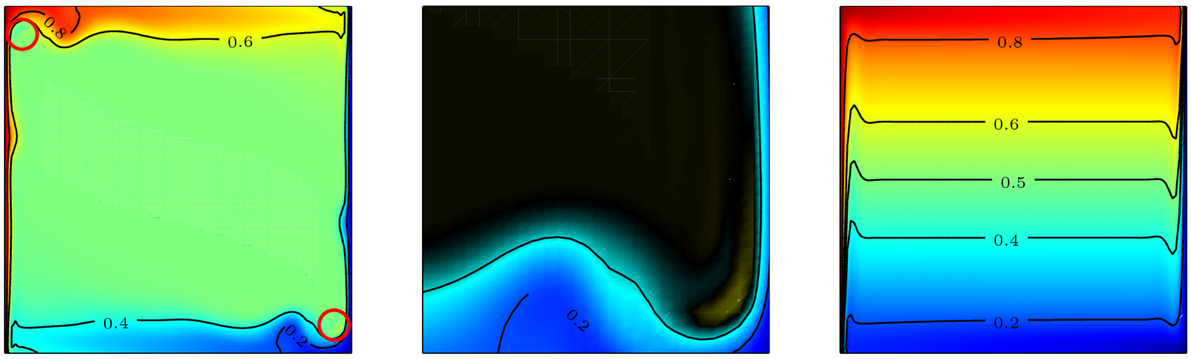


(a) $t = 150\,\mathrm{s}$, $\|\vec{u}_{max}\| = 8.4 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.
(b) $t = 1500\,\mathrm{s}$, $\|\vec{u}_{max}\| = 5.5 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.
(c) $t = 10\,000\,\mathrm{s}$, $\|\vec{u}_{max}\| = 3.9 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

Fig. 4.9: Natural Convection with Heated Lateral Walls, Artificial Fluid, 100×100 mesh, $\tau = 0.05$, evolution of flow, streamlines with color-coded velocity magnitude.

Using a finer resolution of i.e. 100×100 (cell size $1\,\mathrm{cm} \times 1\,\mathrm{cm}$) shows a completely different result (Fig. 4.9). Note that the time for Fig. 4.9(a) is $t = 150\,\mathrm{s}$, when comparing it to Fig. 4.8(a) showing the flow at time $t = 125\,\mathrm{s}$. When taking a look at Fig. 4.9(b) one can see that the fluid starts to drop and rise close to the heating and cooling wall, respectively. This happens in a width of less than $10\,\mathrm{cm}$, explaining why a cell width of $2\,\mathrm{cm}$ (for the 50×50 mesh) might have been too coarse to resolve this flow.

The steady state at time $t = 10\,000\,\mathrm{s}$ shows three vortices embedded in the global flow running through the whole domain. Note that the maximal velocity is decreasing in



(a) $t = 650\,\mathrm{s}$.

(b) $t = 650\,\mathrm{s}$, lower right heat spot with different colormap to increase contrast.

(c) $t = 10\,000\,\mathrm{s}$.

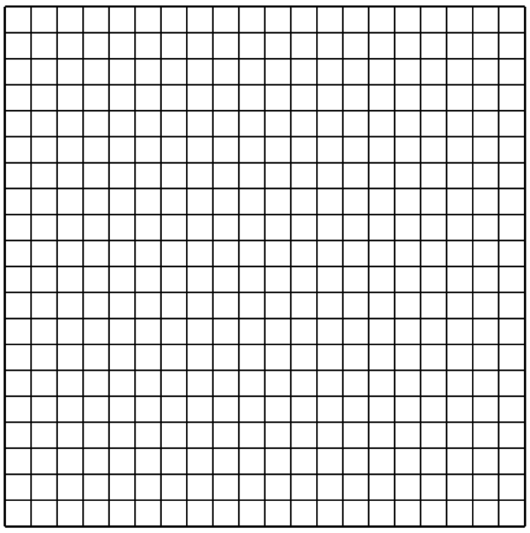
Fig. 4.10: Natural Convection with Heated Lateral Walls, Artificial Fluid, 100×100 mesh, $\tau = 0.05$, evolution of temperature, isolines of temperature.

time, whereas initially of course there is no flow at all, so that it is increasing until time $t = 150\,\mathrm{s}$ or before.

Despite these good looking results, when taking a closer look at the temperature in Fig. 4.10(a), we see that there are unnatural heat spots at the upper left and lower right corner of the box (see the red circled markers and note the color differences within theses markers). There we have cold fluid completely surrounded by warmer fluid and warm fluid by colder fluid, respectively. Fig. 4.10(b) shows a magnification of the lower right heat spot with a different colormap ($T = 0.5$ is now black instead of green) in order to emphasize the heat spot. As we neglect energy conversion from fluid friction into heat, there is no physical cause for this heat spots to occur, especially since the fluid did not pass through the whole domain yet. These spots disappear when the flow evolves – and do not occur at the beginning of the simulation – but they suggest that the discretization is still to coarse. Fig. 4.10(c) shows the flow for the 100×100 mesh at time $t = 10\,000\,\mathrm{s}$. These clearly show the low consistency and diffusivity of the fluid, as the boundary layers are quite small.

Due to the heat spots, we ran another simulation with a 250×250 mesh (cell width $0.4\,\mathrm{cm}$). And indeed, when using this mesh we do not see these spots any more. Too, the bulges that can be seen in Fig. 4.10(a) below and above the heat spots, respectively, disappear. A picture of this simulation is not included since the overall image does not change.

On the one hand it is a good result that we are able to get control of these problems. But, on the other hand we use 62 500 cells for a $1\,\mathrm{m} \times 1\,\mathrm{m}$ box and a 250×250 mesh,


(a) Regular refinement.

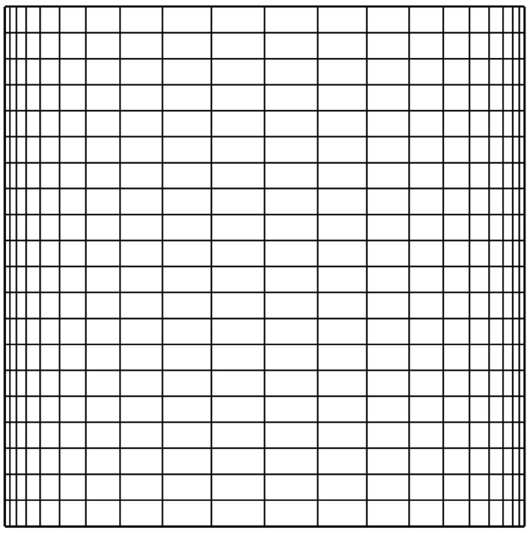
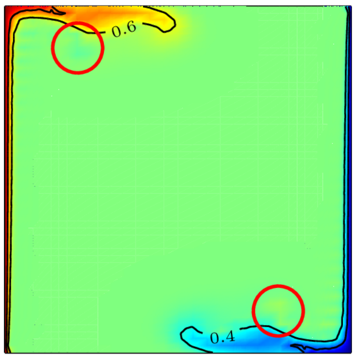

(b) Mesh stretching.

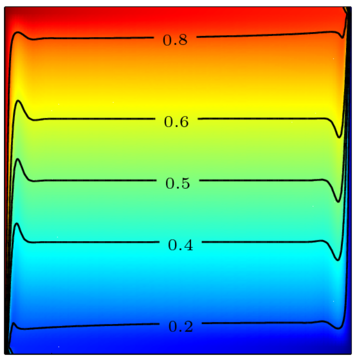
Fig. 4.11: Comparison of a 20×20 mesh with (a) regular refinement and (b) activated mesh stretching along x-axis (`mesh-stretching-direction="0"`) with factor 2 (`mesh-stretching-parameter="2.0"`).

resulting in rather high computing times for such a small scenario. Then again, it is important to notice that – in comparisons to previous fluids – only the boundary layer got smaller, whereas the structure in the whole domain principally remained the same. Regarding this fact, we used a 200×100 mesh, leading to non-square rectangular cells of size $0.5\,\mathrm{cm} \times 1\,\mathrm{cm}$. This works quite well, in the sense that the heat spots disappear – as for the 250×250 mesh. But this mesh still uses 20 000 cells. Therefore, it would be nice to say that we want a finer resolution at the left and right boundary, whereas the rest of the domain uses coarser cells. Since at this point of the work only the `trivialgrid` was available for the `chemical`-component of Peano, we were not able to refine those areas on its own.

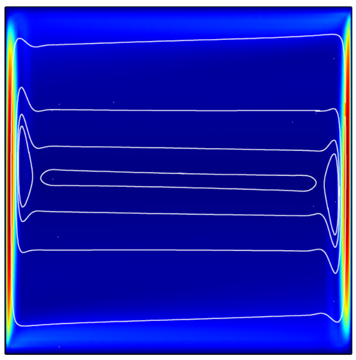
For these cases Peano supports *mesh-stretching*, which can be activated by using the configuration tag `use-mesh-stretching` as attribute of the `trivialgrid` configuration argument. We can define an axis along which the cells are being stretched with a factor. There is no need to change any code, since all operators are implemented to use the actual cell size along each axis solely and we are therefore able to use this feature without any adaptions. In Fig. 4.11 one can see the meshes of a 20×20 `trivialgrid` with regular cells (Fig. 4.11(a)) compared to a 20×20 `trivialgrid` with `mesh-stretching` (Fig. 4.11(b)) along the x-axis (`mesh-stretching-direction="0"`) with a stretch-factor of 2 (`mesh-stretching-parameter="2.0"`).



(a) $t = 250\,\mathrm{s}$, isolines of temperature, color-coded temperature, heat spots.



(b) $t = 10\,000\,\mathrm{s}$, isolines of temperature, color-coded temperature.



(c) $t = 10\,000\,\mathrm{s}$, streamlines, color-coded velocity magnitude, $\|\vec{u}_{max}\| = 4.019 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

Fig. 4.12: Natural Convection with Heated Lateral Walls, Artifical Fluid, 50×50 mesh-stretched grid (along x-axis, factor 2), $\tau = 0.05$.

When using this technique on a 50×50 grid, a cell at the very left or right of the domain approximately has size $3\,\mathrm{mm} \times 20\,\mathrm{mm}$, whereas a cell at the center of the domain has size $42\,\mathrm{mm} \times 20\,\mathrm{mm}$. The results for a computation with this mesh – still using the time step size $\tau = 0.05$ – are shown in Fig. 4.12(a), where still some heat spots occur (see red markers). But what actually is more interesting is that we get an accurate steady state. This can be seen in Fig. 4.12(b) showing the temperature isolines and Fig. 4.12(c) showing the streamlines at time $t = 10\,000\,\mathrm{s}$. This was not possible using the regularly refined 50×50 grid (Fig. 4.8(c)). For a 100×100 mesh-stretched grid we even get rid of the heat spots, which is not shown here (maximal velocity at t_{end} is $\|\vec{u}_{max}\| = 3.998 \times 10^{-3}\,\mathrm{m\,s^{-1}}$).

So, for this particular case – when accepting the heat spots as visible discretization error – we can reduce the number of cells from 10 000 (100×100 mesh) to 2500 (stretched 50×50 mesh) just by using the *mesh-stretching* technique, in order to get the same results. When the heat spots are not acceptable, we get 62 500 cells (250×250 mesh) vs. 10 000 cells (stretched 100×100 mesh), still leading to significantly shorter computation times.

Finally, we have the Table of runs 4.3 for this scenario and the Artificial Fluid. The kind of mesh – with or without *mesh-stretching* – is denoted in the mesh column. Additionally, there is an extra column to indicate whether or not heat spots occurred. The time step size τ did not play such a vital role as in the cases before and has therefore not been the focus of our simulations.

Table 4.3: Natural Convection with Heated Lateral Walls, Artificial Fluid, list of runs, $t_{end} = 10\,000\,\text{s}$.

Mesh		τ	$\|\vec{u}_{max}\|$ $\left[\frac{\text{m}}{\text{s}}\right]$ at t_{end}	Heat Spots
50×50		0.05	Ψ	
	stretched	0.05	4.019×10^{-3}	yes
100×75		0.1	3.875×10^{-3}	yes
100×100		0.05	3.875×10^{-3}	yes
	stretched	0.05	3.998×10^{-3}	no
200×100		0.1	3.919×10^{-3}	no
200×200	stretched	0.01	3.994×10^{-3}	no
250×250		0.05	3.978×10^{-3}	no
400×400		0.01	3.982×10^{-3}	no

4.1.4 Water

In the last section we saw that a high Rayleigh number of $\text{Ra} = 1.47 \times 10^8$ leads to problems that can be solved by refining the grid. Now, when turning to water, we have the following parameters:

$$\begin{aligned} \text{Pr} &= 7, \quad L = 1\,\text{m}, \quad g = (0, -9.81)^T\,\tfrac{\text{m}}{\text{s}^2}, \\ \beta &= 2.1 \times 10^{-4}\,\tfrac{1}{\text{K}}, \quad T_H = 20.5\,°\text{C}, \quad T_C = 19.5\,°\text{C}, \quad T_I = 20.0\,°\text{C}, \\ \rho &= 1000\,\tfrac{\text{kg}}{\text{m}^3}, \quad c_p = 4184\,\tfrac{\text{J}}{\text{kg}\cdot\text{K}}, \quad \kappa = 0.597\,\tfrac{\text{W}}{\text{m}\cdot\text{K}}. \end{aligned} \tag{4.9}$$

Note that we're now using a temperature with unit °C. This time we set the dynamic viscosity η,

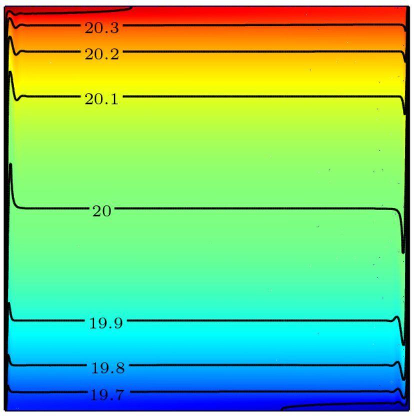
$$\eta = 1 \times 10^{-3}\,\text{Pa}\cdot\text{s},$$

appropriate to the properties of water. Finally, we get:

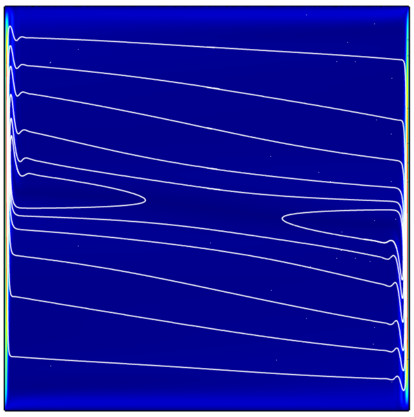
$$\begin{aligned} \nu &= 1 \times 10^{-6}\,\tfrac{\text{m}^2}{\text{s}}, \\ \alpha = \tfrac{\kappa}{c_p \rho} &= 1.4 \times 10^{-7}\,\tfrac{\text{m}^2}{\text{s}}, \end{aligned}$$

and with $u_\infty = 1\,\text{m}\,\text{s}^{-1}$:

$$\text{Re} = 1 \times 10^6 \quad \text{and} \quad \text{Ra} = 1.47 \times 10^{10}.$$



(a) Isolines of temperature, color-coded temperature.



(b) Streamlines, color-coded velocity magnitude, $\|\vec{u}_{max}\| = 4.9 \times 10^{-3}\,\mathrm{m\,s^{-1}}$.

Fig. 4.13: Natural Convection with Heated Lateral Walls, Water, 400×400 mesh-stretched grid, $\tau = 0.01$, $t = 9375\,\mathrm{s}$.

As threshold for the transition of laminar to turbulent flow a critical Rayleigh number of $\mathrm{Ra} \approx 1 \times 10^9$ is stated in [10]. Since there is no turbulence-model introduced to the chemical component so far, we cannot resolve this phenomena. So the only way to get a hold on this is to reduce the cell size a lot, increasing computational costs. And in fact this does not really resolve the turbulence completely – only on a macroscopic level.

Fig. 4.13 shows a simulation using a 400×400 mesh with activated mesh-stretching using a time step size of $\tau = 0.01$. By looking at the streamlines, one can see that the boundary layers are very small for water due to its low viscosity (Fig. 4.13(b)). The maximal velocity of $\|\vec{u}_{max}\| = 4.9 \times 10^{-3}\,\mathrm{m\,s^{-1}}$ at $t = 9375\,\mathrm{s}$ is a bit higher than it was for the artificial fluid but still in the same order of magnitude. We see that we can use water as a fluid, but the number of cells that is necessary to resolve this type of flow might be too high – a simulation with a 200×200 mesh did not show satisfying results. It is possible that using a turbulence model in Peano allows to work on a coarser mesh.

4.2 Rayleigh-Bénard Convection

The next scenario which was used for validation and testing is the Rayleigh-Bénard Convection. Like in the previous scenario, the flow is temperature-driven and no external flow is introduced. Fig. 4.14 shows the setup, the top wall is cooled at temperature T_C and the bottom wall heated at T_H, respectively, resulting in Dirichlet boundary conditions (Γ_D^T) at these two walls. We chose the walls at the left and right to have

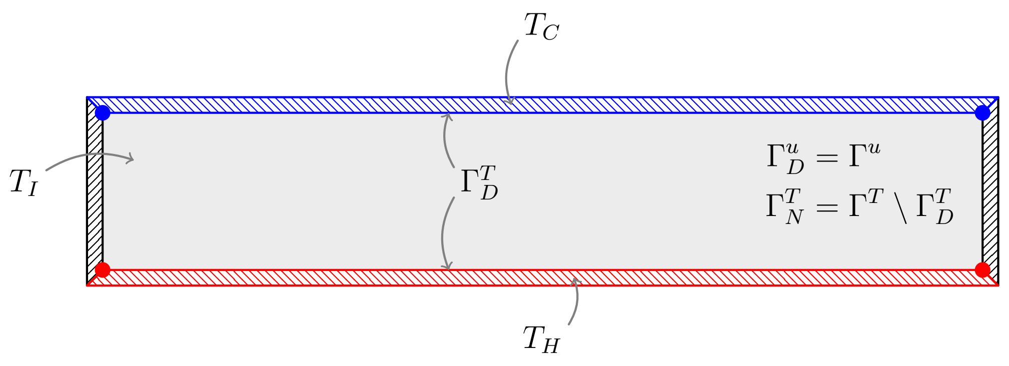


Fig. 4.14: Rayleigh-Bénard Convection, scenario setup.

Neumann boundaries regarding the temperature (Γ_N^T) to simulate adiabatic walls. For the velocities we have no-slip Dirichlet boundaries conditions at all walls ($\Gamma_D^{\vec{u}} = \Gamma^{\vec{u}}$).

In Peano the scenario name for the fluid-scenario is `gravity-box`, whereas the chemical-scenario is called `rayleigh-benard-convection`.

Again, like before, note that the corner vertices belong to the Dirichlet boundary of the temperature and therefore is heated and cooled, respectively. What we expect to happen, is that at first the fluid is heated and cooled by conduction and almost no actual flow occurs. Then, after conduction is established, convection comes into play and a flow with a special structure is forming, the so-called *Rayleigh-Bénard convection cells*. A Rayleigh-Bénard cell is an eddy in the domain, where the fluid rises and drops due to density differences caused by temperature difference in a regular manner. These typically occur in shallow layers, leading to very thin but broad domains. Griebel states that there is no mass exchanged between these eddies for two-dimensional flows, in contrast to three-dimensional cells. It is also stated that Rayleigh-Bénard convection cells only occur when the critical Rayleigh number,

$$\mathrm{Ra} \approx 1108,$$

is exceeded. To avoid influences from the left and the right wall, these need to be far enough apart. Like Griebel [6] we take a look at different fluids: Glycerine, Air and Water.

4.2.1 Glycerine

We start with glycerine as the first fluid for the Rayleigh-Bénard Convection scenario. The scenario configuration is:

$$g = (0, -9.81)^T \, \frac{\mathrm{m}}{\mathrm{s}^2},$$
$$T_H = 294.78\,\mathrm{K}, T_C = 291.2\,\mathrm{K}, T_I = 293\,\mathrm{K},$$

so that we have temperature difference of 3.58 K. For the fluid Griebel presents the following parameters (this time with units)

$$\begin{aligned} \mathrm{Pr} &= 12\,500, \quad \mathrm{Re} = 33.73, \quad \mathrm{Ra} \approx 1\times 10^4, \\ \beta &= 5\times 10^{-4}\,\tfrac{1}{\mathrm{K}}, \\ \rho &= 1264\,\tfrac{\mathrm{kg}}{\mathrm{m}^3}, \quad \eta = 1.499\,\mathrm{Pa}\cdot\mathrm{s}, \end{aligned} \tag{4.10}$$

and therefore

$$\begin{aligned} \nu &= 1.186\times 10^{-3}\,\mathrm{Pa}\cdot\mathrm{s}, \\ \alpha &= \tfrac{\nu}{\mathrm{Pr}} = 9.49\times 10^{-8}\,\tfrac{\mathrm{m}^2}{\mathrm{s}}. \end{aligned}$$

Since Peano only uses α in the simulations, it is not important that c_p and κ really fit the fluids properties. Nevertheless, we need to specify them in the configuration file, therefore we set:

$$c_p = 1\,\frac{\mathrm{J}}{\mathrm{kg}\cdot\mathrm{K}},$$

and get:

$$\kappa = \alpha\rho c_p = 1.199\times 10^{-4}\,\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}.$$

Griebel suggests a 38 cm × 4 cm box, leading to:

$$L = 0.04\,\mathrm{m},$$

and we finally assume $u_\infty = 1\,\mathrm{m\,s}^{-1}$ as always. After all parameter have been defined the Rayleigh number is:

$$\mathrm{Ra} = 9.985\times 10^3.$$

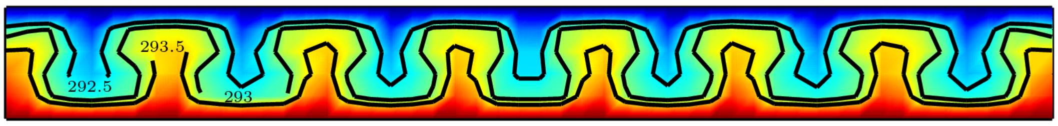
This time Griebel provides the total simulation time t_{end},

$$t_{end} = 10\,000\,\mathrm{s}.$$

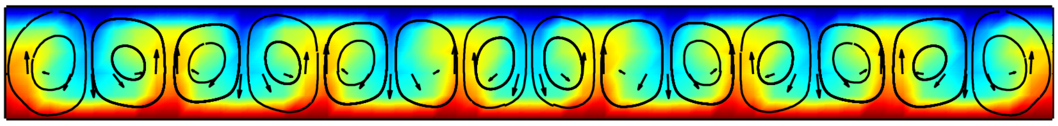
Do not be confused when reading `t_end = 2.5` $\cdot\, 10^5$ in Griebel [6], that's the dimensionless time corresponding to 10 000 s.

We use two different meshes, a 49 × 5 (cell size 7.7 mm × 8 mm) and a 227 × 21 mesh (1.7 mm × 1.9 mm). For both, non-dimensionless quantitative data is presented [6] (Section 9.7.2), which we used for comparing.

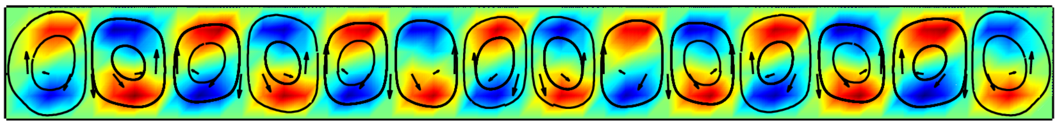
At first we take a look at the 49 × 5 mesh, leaving us only four degrees of freedoms for the temperature in the y-direction, since we have Dirichlet boundaries at the top



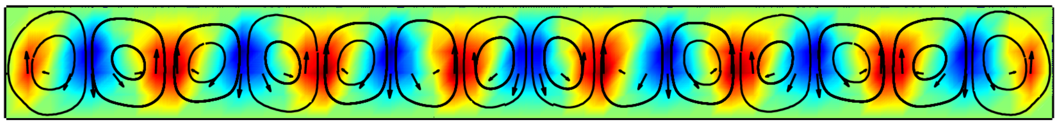
(a) Isolines of temperature, color-coded temperature.



(b) Streamlines with color-coded temperature.



(c) $u_{max} = 7.35 \times 10^{-5}\,\mathrm{m\,s^{-1}}, u_{min} = -7.35 \times 10^{-5}\,\mathrm{m\,s^{-1}}$, streamlines with color-coded u

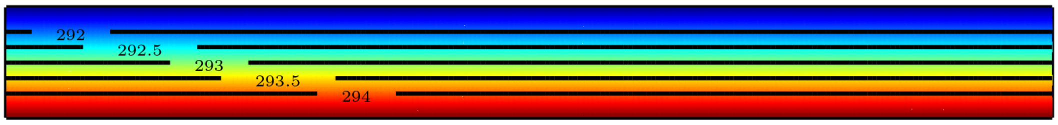


(d) $v_{max} = 1.04 \times 10^{-4}\,\mathrm{m\,s^{-1}}, v_{min} = -9.65 \times 10^{-5}\,\mathrm{m\,s^{-1}}$, streamlines with color-coded v

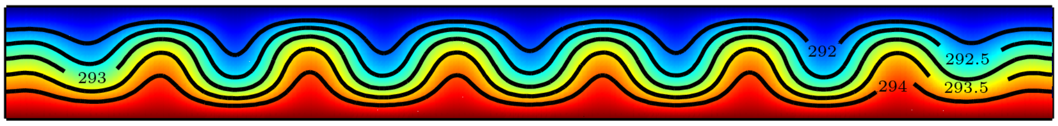
Fig. 4.15: Rayleigh-Bénard Convection, Glycerine, 49×5 mesh, $t = t_{end} = 10\,000\,\mathrm{s}$, $\tau = 0.01$, temperature and velocity.

and bottom. To resolve a real flow, this actually seems rather few. Results – using the time step size $\tau = 0.01$ – at time t_{end} are shown in Fig. 4.15. The fungi-formed isolines of the temperature in Fig. 4.15(a) show the warm fluid pushing upwards from eight spots at the bottom of the box. Lying inbetween each two of these spots cold fluid is pushing down from the top, forming Rayleigh-Bénard convection cells. Note that the rather rough isolines are due to the coarse discretization. Fig. 4.15(b) shows the color-coded temperature, just like before. Additionally, the streamlines are plotted to better see the interaction of heat transport and density differences. Fig. 4.15(c) and Fig. 4.15(d) show the same streamlines but with different color-coded data. The former color-codes u, the latter v of the flow. Since these are not absolute values, we also get negative velocities, meaning that at that point the fluid is flowing in the negative x and y-direction, respectively.

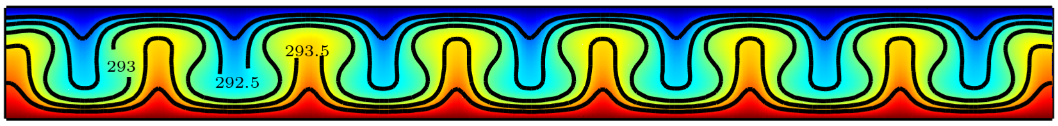
Before we go on to the finer mesh, note that there are a total of 14 Rayleigh-Bénard cells, whereas Griebel's result for the coarse discretization only shows 12. When refining the grid, both Griebel's and our results show 14 cells as well, so that he attributes



(a) $t = 2500\,\mathrm{s}$.



(b) $t = 4300\,\mathrm{s}$.



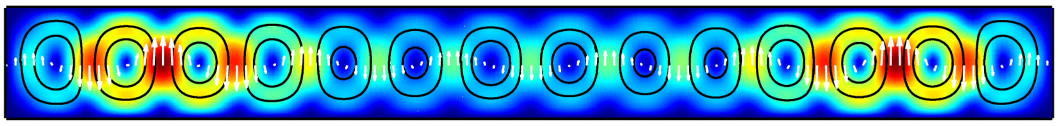
(c) $t = t_{end} = 10\,000\,\mathrm{s}$.

Fig. 4.16: Rayleigh-Bénard Convection, Glycerine, 227×21 mesh, $\tau = 0.001$, evolution of flow, isolines of temperature, color-coded temperature.

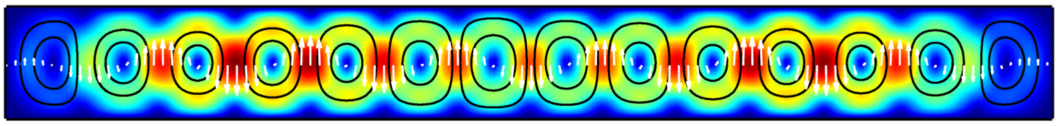
this phenomena to the discretization size. Keep in mind that Griebel uses a finite difference method whereas Peano implements a finite element method. The different number of Rayleigh-Bénard cells may explains, why the velocities given in Griebel ($u_{max} \approx 4.76 \times 10^{-5}\,\mathrm{m\,s^{-1}}$, $v_{max} \approx 7.78 \times 10^{-5}\,\mathrm{m\,s^{-1}}$) differ quite a lot from our results for the coarse grid.

Now we take a look at the 227×21 mesh, giving us a very fine cell size of $1.7\,\mathrm{mm} \times 1.9\,\mathrm{mm}$. Fig. 4.16 shows the temperature isolines at three different times $t = 2500\,\mathrm{s}, 4300\,\mathrm{s}$ and $10\,000\,\mathrm{s}$ using the time step size $\tau = 0.001$. One can see that there is no heat exchanged by convection at time $t = 2500$ (Fig. 4.16(a)) as all contour lines are still straight. Note that this is after a quarter(!) of the total simulation time elapsed. As can be seen in Fig. 4.16(b), the temperature is transported – by convection – into the Rayleigh-Bénard cells afterwards. Fig. 4.16(c) then shows the steady state at time $t_{end} = 10\,000\,\mathrm{s}$, presenting the same structure as the 49×5 mesh.

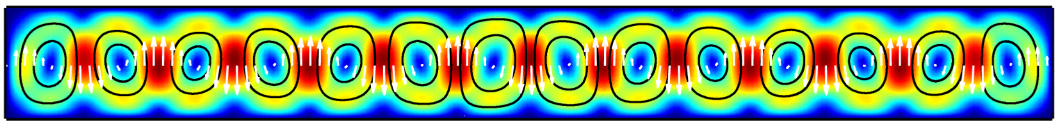
The streamlines and their chronological progress can be seen in Fig. 4.17. One can see the forming of the eddies – starting to form from the left and right wall into to middle of the box in Fig. 4.17(a). Note that the flow is very slow ($\|\vec{u}_{max}\| = 1.15 \times 10^{-9}\,\mathrm{m\,s^{-1}}$ at time $t = 2500\,\mathrm{s}$) and has no significant influence on the temperature transport (compare Fig. 4.17(a) and Fig. 4.16(a)). Time $t = 4000$ shows that the eddies have reached the center, having a maximal velocity of ($\|\vec{u}_{max}\| = 1.52 \times 10^{-5}\,\mathrm{m\,s^{-1}}$). This shows that the



(a) $t = 2500\,\mathrm{s}$, $\|\vec{u}_{max}\| = 1.15 \times 10^{-9}\,\mathrm{m\,s^{-1}}$.



(b) $t = 4000\,\mathrm{s}$, $\|\vec{u}_{max}\| = 1.52 \times 10^{-5}\,\mathrm{m\,s^{-1}}$.



(c) $t = t_{end} = 10\,000\,\mathrm{s}$, $\|\vec{u}_{max}\| = 8.05 \times 10^{-5}\,\mathrm{m\,s^{-1}}$.

Fig. 4.17: Rayleigh-Bénard Convection, Glycerine, 227×21 mesh, $\tau = 0.001$, evolution of flow, streamlines with color-coded velocity magnitude.

convective heat transport itself has a great impact on the speed of the flow. For the final time t_{end} in Fig. 4.17(c) we have:

$$\begin{aligned} \|\vec{u}_{max}\| &= 8.05 \times 10^{-5}\,\tfrac{\mathrm{m}}{\mathrm{s}}, \\ \|u_{max}\| = 5.28 \times 10^{-5}\,\tfrac{\mathrm{m}}{\mathrm{s}}, &\quad \|v_{max}\| = 8.05 \times 10^{-5}\,\tfrac{\mathrm{m}}{\mathrm{s}}, \end{aligned}$$

which is very close to Griebel's result ($u_{max} \approx 5.45 \times 10^{-5}\,\mathrm{m\,s^{-1}}$, $v_{max} \approx 8.07 \times 10^{-5}\,\mathrm{m\,s^{-1}}$). Bearing in mind that our results were better for the coarser mesh, our results are probably more accurate than Griebel's for the fine mesh as well.

4.2.2 Air

The second fluid we take a look at is air. The given Griebel parameters are:

$$\begin{aligned} \mathrm{Pr} &= 0.72, \quad \mathrm{Re} = 4365, \quad \mathrm{Ra} \approx 30\,000, \\ \beta &= 3.4 \times 10^{-3}\,\tfrac{1}{\mathrm{K}}, \quad u_\infty = 1\,\mathrm{m\,s^{-1}}, \\ T_H &= 293.5\,\mathrm{K}, \quad T_C = 292.5\,\mathrm{K}, \quad T_I = 293\,\mathrm{K}. \end{aligned} \tag{4.11}$$

Griebel sets the dynamic viscosity $\eta^G = 1.81 \times 10^{-4}\,\mathrm{Pa \cdot s}$, whereas the real viscosity of air is:

$$\eta = 1.81 \times 10^{-5}\,\mathrm{Pa \cdot s},$$

so that we guess that this is a typo. We use the real viscosity η in our simulations. Therefore we have

$$\begin{aligned} \rho &= 1.205\,\frac{\mathrm{kg}}{\mathrm{m}^3}, \text{ and} \\ \nu &= \frac{\eta}{\rho} = 1.502\times10^{-5}\,\frac{\mathrm{m}^2}{\mathrm{s}}, \end{aligned}$$

which leads to

$$\alpha = \frac{\nu}{\mathrm{Pr}} = 2.086\times10^{-5}\,\frac{\mathrm{m}^2}{\mathrm{s}}.$$

Just like for glycerine we can fix $c_p = 1\,\frac{\mathrm{J}}{\mathrm{kg}\cdot\mathrm{K}}$ and get:

$$\kappa = \alpha\rho c_p = 2.514\times10^{-5}\,\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}},$$

since the real properties of air for κ and c_p are not important for Peano, as long as α has the correct value.

The characteristic length L is the height of the box and chosen to fit the given Rayleigh number,

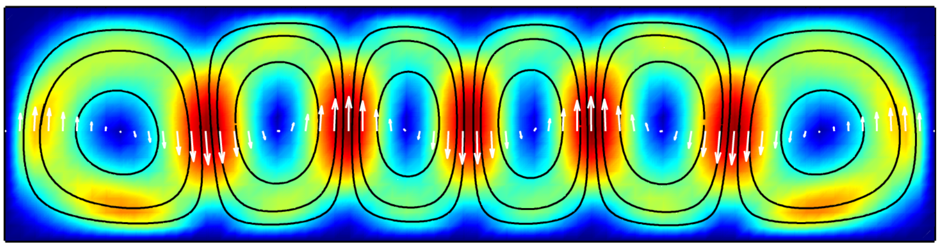
$$L = \left(\frac{\mathrm{Ra}\,\nu\alpha}{\|g\|\,\beta\,(T_H - T_C)}\right)^{\frac{1}{3}} = 6.56\times10^{-2}\,\mathrm{m}.$$

For the domain Griebel uses a box with aspect ratio 4:1. Therefore we have a domain size of 26.24 cm $\times$ 6.56 cm. As final time we set:

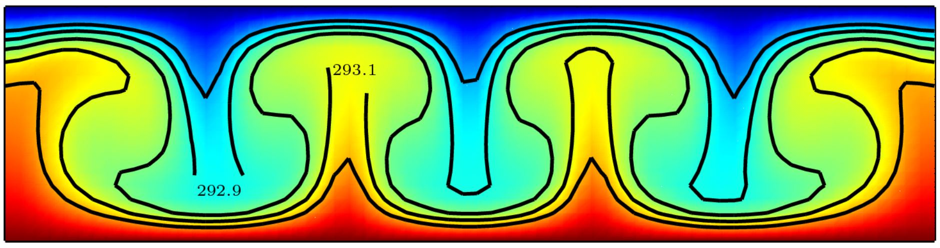
$$t_{end} = 1000\,\mathrm{s}.$$

Summarizing, we have a highly non-viscous fluid with better heat conductivity than water.

In Fig. 4.18 we see the results using a 65×17 mesh (approx. cell size 4 mm $\times$ 3.9 mm) with $\tau = 0.01$. When comparing the streamlines in Fig. 4.18(a) to Griebel's results (Fig. 4.19(a)), we do not see much difference, since six Rayleigh-Bénard cells have formed in both cases. But when looking at the temperature isolines in Fig. 4.18(b) and Fig. 4.19(b), it is obvious that our results do not match Griebel's. Instead of three spots at the bottom where the heat rises to the top, we get four of those at the bottom and three at the top where the cold fluid is pushing down into the box. Griebel's results are the other way around – three warm spots at the bottom and four cold spots at the top. After all, 65×17 cells are too coarse for such a scenario and to resolve this flow appropriately. In this case we used a fixed time step size of $\tau = 0.01$, which is below the time stepping criteria that were presented in Section 3.6. The results do not differ, when using $\tau = 0.1$ and $\tau = 0.15$ instead of $\tau = 0.01$.

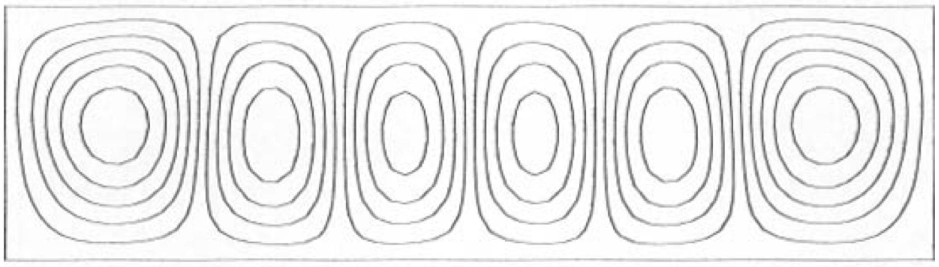


(a) Streamlines with color-coded velocity magnitude, $\|\vec{u}_{max}\| = 2.34 \times 10^{-2}\,\mathrm{m\,s^{-1}}$.

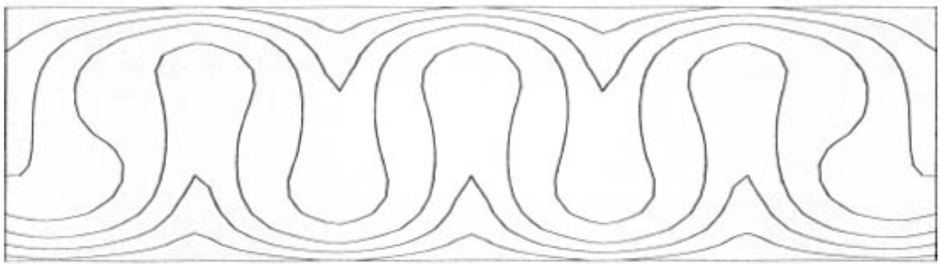


(b) Isolines of temperature with color-coded temperature

Fig. 4.18: Rayleigh-Bénard Convection, Air, 65×17 mesh, $t = t_{end} = 1000\,\mathrm{s}$, $\tau = 0.01$, streamlines and isolines of temperature.

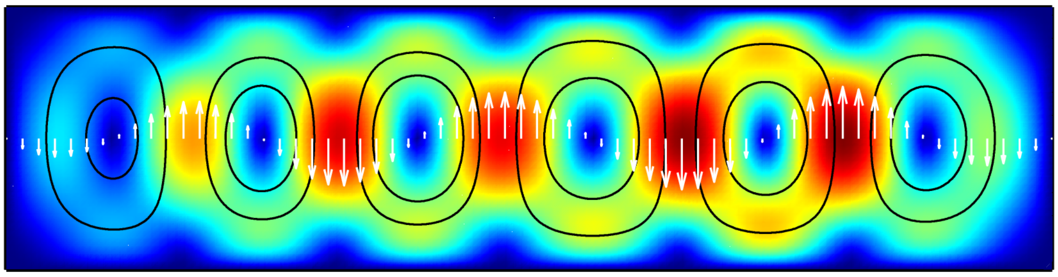


(a) Streamlines, $u^G_{max} \approx 1.3 \times 10^{-2}\,\mathrm{m\,s^{-1}}$, $v^G_{max} \approx 1.8 \times 10^{-2}\,\mathrm{m\,s^{-1}}$.

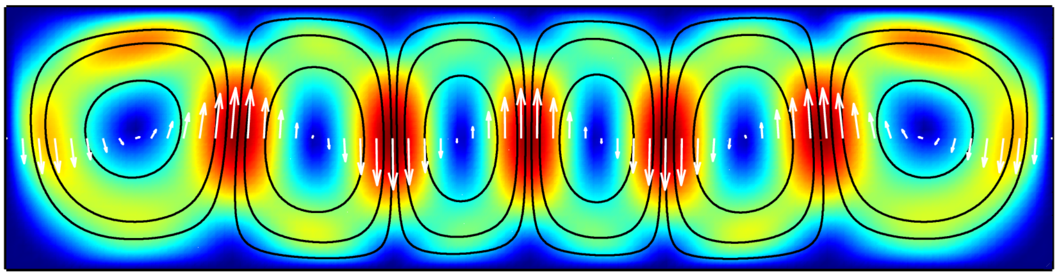


(b) Isolines of temperature.

Fig. 4.19: Griebel results, Rayleigh-Bénard Convection, Air, 65×17 mesh, $t = t_{end} = 1000\,\mathrm{s}$, streamlines and isolines of temperature, [6].



(a) $t = 50\,\text{s}$, $\|\vec{u}_{max}\| = 1.82 \times 10^{-4}\,\text{m}\,\text{s}^{-1}$.



(b) $t = 1000\,\text{s}$, $\|\vec{u}_{max}\| = 2.276 \times 10^{-2}\,\text{m}\,\text{s}^{-1}$.

Fig. 4.20: Rayleigh-Bénard Convection, Air, 130×34, $\tau = 0.005$, evolution of flow, streamlines with color-coded velocity magnitude.

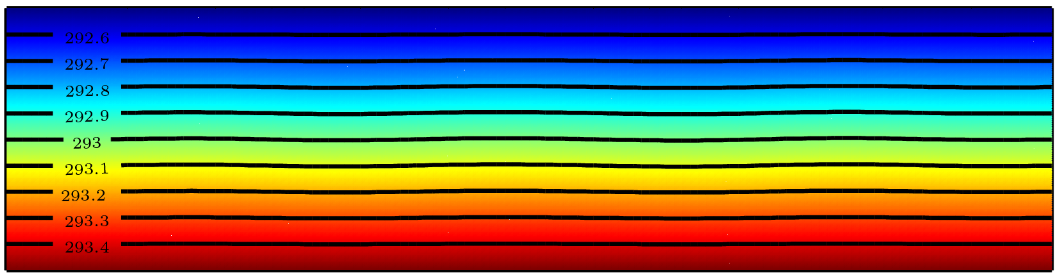
After refining the mesh to 130×34 cells (approx. cell size $2\,\text{mm} \times 1.9\,\text{mm}$) we get the results we expected (Fig. 4.20). In Fig. 4.20(a) we see the flow at time $t = 50\,\text{s}$. The maximal velocity is $\|\vec{u}_{max}\| = 1.82 \times 10^{-4}\,\text{m}\,\text{s}^{-1}$, which is slow compared to the maximal velocity $\|\vec{u}_{max}\| = 2.33 \times 10^{-2}\,\text{m}\,\text{s}^{-1}$ at time $t = 100\,\text{s}$. For the steady state at time $t = t_{end} = 1000\,\text{s}$ we have:

$$\begin{aligned} \|\vec{u}_{max}\| &= 2.276 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}, \\ u_{max} = 1.71 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}, &\quad v_{max} = 2.23 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}, \end{aligned}$$

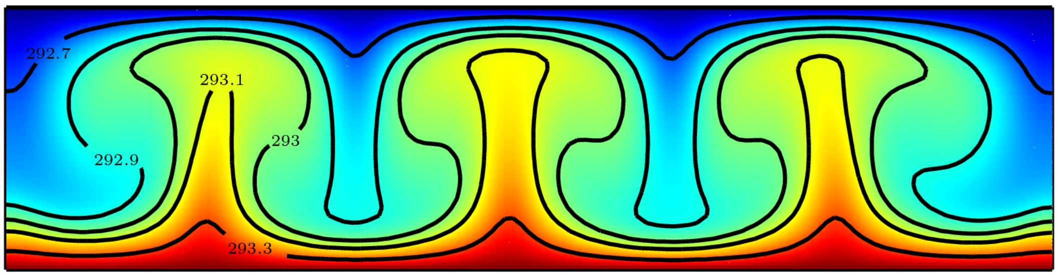
which is above the velocities that are given by Griebel ($u^G_{max} \approx 1.3 \times 10^{-2}\,\text{m}\,\text{s}^{-1}$, $v^G_{max} \approx 1.8 \times 10^{-2}\,\text{m}\,\text{s}^{-1}$). But then we have to keep in mind that we do not really know which parameters Griebel used for his calculations. If(!) Griebel really used $\eta^G = 10\eta$ for his calculations, this would explain, why our velocities are higher than his, since lower viscosity typically results in faster flows in natural convection scenarios.

When refining the grid to 260×68 cells (approx. cell size $1\,\text{mm} \times 1\,\text{mm}$), the flow does not change significantly and we get the following velocities:

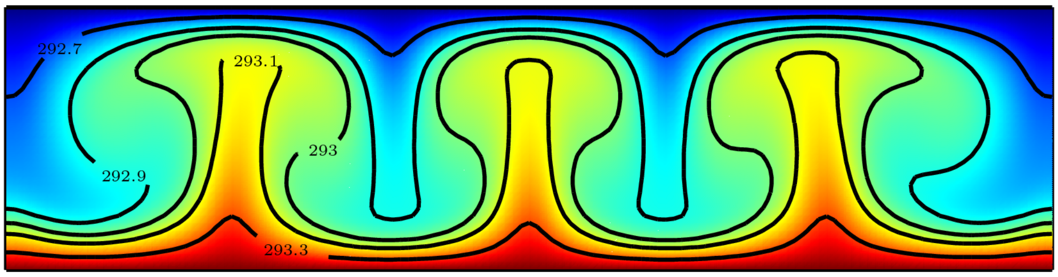
$$\begin{aligned} \|\vec{u}_{max}\| &= 2.26 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}, \\ u_{max} = 1.72 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}, &\quad v_{max} = 2.22 \times 10^{-2}\,\tfrac{\text{m}}{\text{s}}. \end{aligned}$$



(a) $t = 50\,\mathrm{s}$.



(b) $t = 100\,\mathrm{s}$.



(c) $t = 1000\,\mathrm{s}$.

Fig. 4.21: Rayleigh-Bénard Convection, Air, 130×34 mesh, $\tau = 0.005$, evolution of flow, isolines of temperature with color-coded temperature.

Finally, we show the temperature isolines for the 130×34 mesh using the time step size $\tau = 0.005$ at different times. For $t = 50\,\mathrm{s}$ (Fig. 4.21(a)) we see that convective heat transport is about to begin, since the isolines are not completely straight any more. Fig. 4.21(b) shows the almost fully developed flow, where the main heat transport happens by convection. Remember that it took more than a quarter of the total simulation time for that process to start for glycerine. This can be explained by looking at the fluids properties. Air is much less viscous than glycerine, so that the flow is more sensitive – in a convective kind of sense – to temperature differences. This can be seen by comparing the velocities of Fig. 4.17(a) ($\|\vec{u}_{max}\| = 1.15 \times 10^{-9}\,\mathrm{m\,s^{-1}}$) and Fig. 4.20(a)

($\|\vec{u}_{max}\| = 1.82 \times 10^{-4}\,\mathrm{m\,s^{-1}}$). In both cases the convective heat transport has not yet begun (Fig. 4.16(a) and Fig. 4.21(a)), but air is a lot faster than glycerine.

As was mentioned before, there is a big velocity difference from time $t = 50\,\mathrm{s}$ to time $t = 100\,\mathrm{s}$, indicating that convective heat transport has a greater influence on the speed of the flow, than diffusive heat transport. This can be seen in Fig. 4.21(b), where the steady state is almost reached. Fig. 4.21(c) then shows the steady state at time $t = 1000\,\mathrm{s}$. There is not much change to time $t = 100\,\mathrm{s}$, only that the shapes slightly move towards the center.

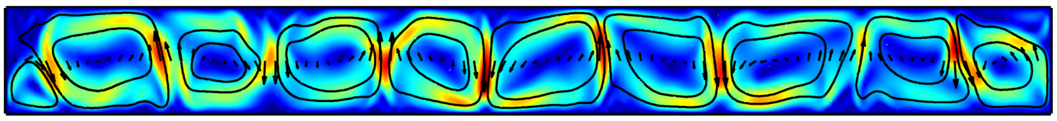
All in all we can say that this scenario works quite well with Peano. For glycerine we have slightly better results than Griebel [6], whereas air needs a finer discretization for the domain, but still procudes the results we expected.

4.2.3 Water

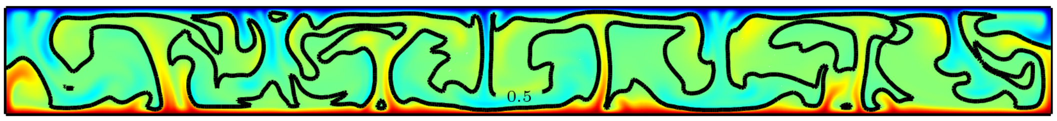
We now turn to water – parameters can be found in Section 4.1.4. Griebel does not provide a box ratio for water, so that we have to choose our own. Since this thesis is not a parameter analysis for the Rayleigh-Bénard Convection scenario, we fixed the boxes size at $0.5\,\mathrm{m} \times 0.05\,\mathrm{m}$ using $L = 0.05\,\mathrm{m}$ as the characteristic length. The Rayleigh number (4.1) thus is:

$$\mathrm{Ra} \;=\; 1.84 \times 10^{6}.$$

We ran a simulation on a 500×50 (cell size $1\,\mathrm{mm} \times 1\,\mathrm{mm}$) with the time step size $\tau = 0.005$. In contrast to the flows of glycerine and air, the flow of water in this setup does *not* have a steady state. The streamlines in Fig. 4.22(a) at time $t = 10\,000\,\mathrm{s}$ show that Rayleigh-Bénard cells still form. But the cells are not stable and deform from step



(a) Streamlines with color-coded velocity magnitude.



(b) Isoline for temperature $T = 0.5$ with color-coded temperature.

Fig. 4.22: Rayleigh-Bénard Convection, Water, 500×50 mesh (cell size $1\,\mathrm{mm} \times 1\,\mathrm{mm}$), $t = 10\,000\,\mathrm{s}$, $\tau = 0.005$.

to step, forming new cells. Fig. 4.22(b) shows the temperature isoline at $T = 0.5$. We can see the spots at the bottom and top of the wall, where the warm and cold fluid accumulates before it rises and drops, respectively.

As before, Peano does not yet handle effects of turbulence the `chemical` component. These could play a vital role, but there is no real data that we could have used for comparing. Note again that we set the box ratio almost arbitrarily and thus an other domain might give different results.

4.3 Flat Plate in Parallel Flow

The last validation scenario is the first one to involve external (driven) flow and is called *Flat Plate in Parallel Flow*. We have an isothermal plate which is placed in the middle of a global flow, that is flowing at constant speed u_∞ from the left to the right.

In Fig. 4.23 one can see the general setup for this scenario. The box in the Figure just represents our simulation domain, meaning that the outer flow is not limited to the box. The plate is heated at temperature $T_P = T_H$, whereas the inflow has an initial temperature $T_\infty = T_C$ for all times. So, it is obvious that Dirichlet boundary conditions have to be imposed on the left and bottom boundary of the scenario (Γ_D^T). For the velocity we have a constant inflow u_∞ at the left and zero-velocities for the wall at the bottom, leading to Dirichlet boundaries for the velocities ($\Gamma_D^{\vec{u}}$) as well. Note that the lower left scenario corner, which is the very edge of the plate at the left side, belongs to the plate. The vertex that is located there technically gets the boundary conditions that are used for the plate. The upper left corner vertex gets the conditions that are used for the inflow.

Also note that there is no actual wall at the top of the general scenario, but we still need to limit our computational domain. The idea is to choose the height of the box according to the fluids parameters, so that the flow at the top does not distinguish itself from the flow above of it, so that there should be a flow at constant speed u_∞ from the left to the right at the top. But this cannot be accomplished by any incompressible code, since we have conservation of mass and the code assures that the total outflow matches the total inflow. Since the fluid is slowed down along the plate, it gets faster at the top of the box to assure the same outflow, reaching a velocity $\tilde{u}_\infty$.

Nevertheless, we need to impose a boundary condition for the upper domain boundary and implemented free-slip condition in y-direction there. This means that there is neither in- nor outflow in y-direction, by setting a no-slip Dirichlet condition for the y-component (normal) of the velocity and a Neumann condition for the x-component (tangential) (see Section 2.2). If pure Neumann conditions would be used for the upper boundary, the fluid would be pushing into the domain from the top due to gravity. For the temperature we set Neumann boundaries, since free-slip conditions are only interesting when having at least two-dimensional degrees of freedom.

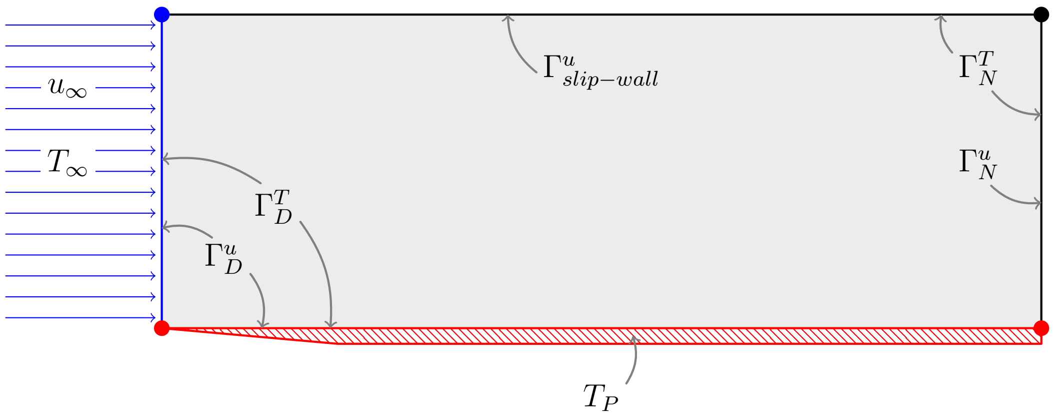


Fig. 4.23: Flat Plate in Parallel Flow, scenario setup.

Further note that the inflow does *not* have a parabolic profile, meaning that the flow is not influenced in any way before the first contact with the plate.

In Peano the scenario name for the fluid-scenario is `flat-plate-in-parallel-flow`, whereas the chemical-scenario is called `heated-plate-in-cooled-flow`.

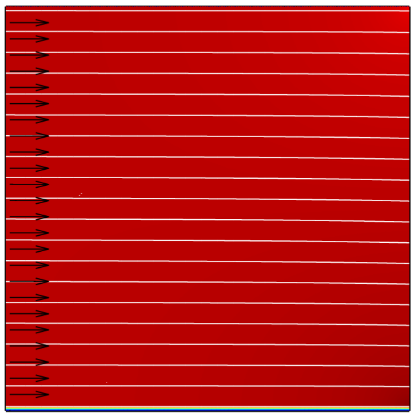
4.3.1 Hydrostatic Pressure

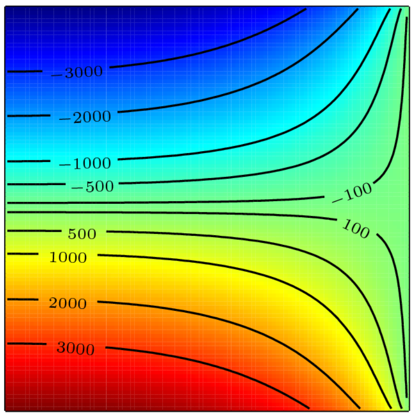
As mentioned before, this is the first scenario where we have external flow. Therefore, we have to deal with the pressure at the outflow boundary, which is on the right. Until now Peano automatically imposed zero pressure boundary conditions at the outflow, when the inflow was specified as a velocity. We modified this behavior, so that we are now able to set inflow velocity independently from the outflow pressure using the configuration tags `velocity-mean-value` and `outlet-pressure` (see Section 3.9).

It also has been pointed out that the Boussinesq approximation requires the gravity to be non-zero in order to take density differences of the fluid into account. Now, when using water (parameters can be found in Section 4.1.4 on Page 52) with an inflow velocity of $u_\infty = 1\,\mathrm{m\,s^{-1}}$ a quite strange phenomena occurs.

In Fig. 4.24 we see the very beginning ($t = 0.002\,\mathrm{s}$) of a simulation with time step size $\tau = 0.001$ on a 100×100 mesh for a $1\,\mathrm{m} \times 1\,\mathrm{m}$ box (cell size $1\,\mathrm{cm} \times 1\,\mathrm{cm}$). The fluid flows through the domain with almost constant speed u_∞ (slightly pushed downwards due to gravity, Fig. 4.24(a)). The pressure ($p^C_{max} = 4534, p^C_{min} = -4481$) in Fig. 4.24(b) clearly shows the influence of the gravity on the left side of the domain, whereas the gravity is neglected at the outflow since we have zero pressure boundary conditions.

It is obvious that the hydrostatic pressure is not taken into account at the outlet on the right side, so that we get a rather non-physical pressure distribution for our simulation.


(a) Streamlines with color-coded velocity magnitude.



(b) Isolines of pressure with color-coded pressure.

Fig. 4.24: Flat Plate in Parallel Flow, water, 100×100 mesh, $u_\infty = 1\,\mathrm{m\,s^{-1}}$, $t = 0.002$, $\tau = 0.001$.

The hydrostatic pressure is the pressure that is exerted by a fluid when it is at rest. So, when assuming constant density ρ and gravity g, the hydrostatic pressure is:

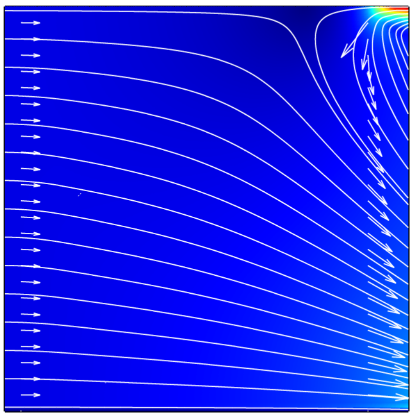
$$p_{hydrostatic} = \rho \, \|g\| \, h, \tag{4.12}$$

where h is the height of the fluid column to a reference point (with respect to the gravity) – which in our case is the top of the domain.

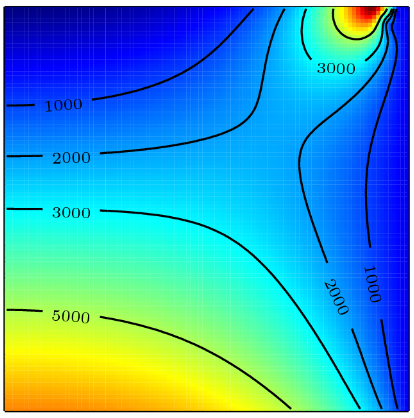
What happens is that the inflowing fluid experiences gravity forces, is pushed downwards and accelerates. This can be seen in Fig. 4.25(a), showing the streamlines and the velocities at time $t = 0.15\,\mathrm{s}$. The right border is considered to be the outflow, but since we imposed Neumann boundaries there, nothing prevents the fluid from flowing *into* the domain. Due to conservation of mass this is exactly what happens – otherwise there would be more fluid flowing out of the domain than into it.

Nevertheless, since we only have one outflow, the *right* inflow needs to flow out on the *right* side as well. This explains the high pressure area ($p^C_{max} = 10\,100$) at the top right side in Fig. 4.25(b), which pushes the fluid downwards and finally out of the domain. The gravitation acts as acceleration on the fluid, so that the velocities rise and rise until Peano crashes due to overflow.

The solution to this problem is to overlay the hydrodynamic pressure with the hydrostatic pressure at the right boundary. The result of this can be seen in Fig. 4.26(b), where we have a pressure difference of 9614 Pa between a cell center at the top and a cell center at the bottom. Fig. 4.26(a) shows the streamlines that are not bend down any more (compare to Fig. 4.24(a)). Note that this functionality has been implemented after the runs have been performed.

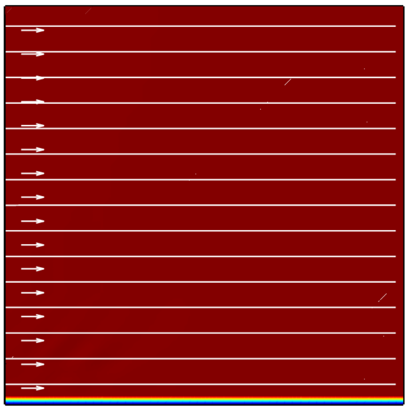


(a) Streamlines with color-coded velocity magnitude, $\|\vec{u}_{max}\| = 10.07\,\mathrm{m\,s^{-1}}$.

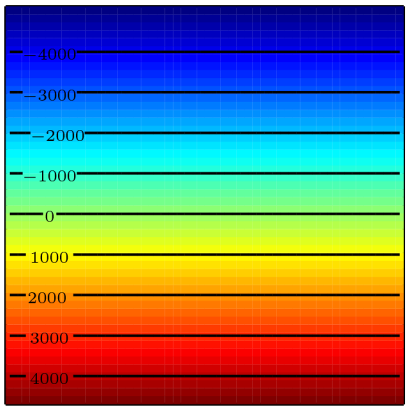


(b) Isolines of pressure with color-coded pressure, $p^C_{max} = 10\,100$, $p^C_{min} = -1134$.

Fig. 4.25: Flat Plate in Parallel Flow, water, 100×100 mesh, $u_\infty = 1\,\mathrm{m\,s^{-1}}$, $t = 0.15\,\mathrm{s}$, $\tau = 0.001$.



(a) Streamlines with color-coded velocity magnitude, $\|\vec{u}_{max}\| = 1\,\mathrm{m\,s^{-1}}$.



(b) Isolines of pressure with color-coded pressure, $p^C_{max} = 4593$, $p^C_{min} = -5021$.

Fig. 4.26: Flat Plate in Parallel Flow, Water, 100×100 mesh, $u_\infty = 1\,\mathrm{m\,s^{-1}}$, $\tau = 0.01$.

4.3.2 Boundary Layers

As was mentioned before, the inflow does *not* have a parabolic profile. This is important because we investigate the two boundary layers, the velocity and the thermal boundary layer (Fig. 4.27).

The concept of the velocity boundary is layer is easily explained. At the surface of the plate the fluid experiences zero velocities, so that there is no movement at all. The fluid then transfers momentum to the next layer through the action of viscosity, so that the fluid is retarded at this next layer. This transfer happens from layer to layer and its influence vanishes with increasing distance to the surface. The thickness $\delta(x)$ of this layer is of course dependent on the length x that the fluid has been exposed to the boundary.

The same holds for the thermal boundary layer principally. Energy is transferred from the plate's surface into the first fluid layer, which exchanges energy with the next layer, and so on. The thickness of this layer – until no significant amount of energy is transferred any more – is called $\delta_T(x)$, which again is dependent on the length x, that the fluid is flowing along the wall. A good introduction to boundary layers can be found in [10].

For the thickness of the boundary layer of the velocity one can use the formula:

$$\delta(x) = 5\frac{x}{\sqrt{\mathrm{Re}_x}}, \tag{4.13}$$

given in [12] in Section 8.2.3, where Re_x is the Reynolds number for the characteristic length x:

$$\mathrm{Re}_x = \frac{u_\infty x}{\nu}. \tag{4.14}$$

For the ratio of thermal to velocity boundary layer one can find:

$$\frac{\delta}{\delta_T} \approx \mathrm{Pr}^{\frac{1}{3}}, \tag{4.15}$$

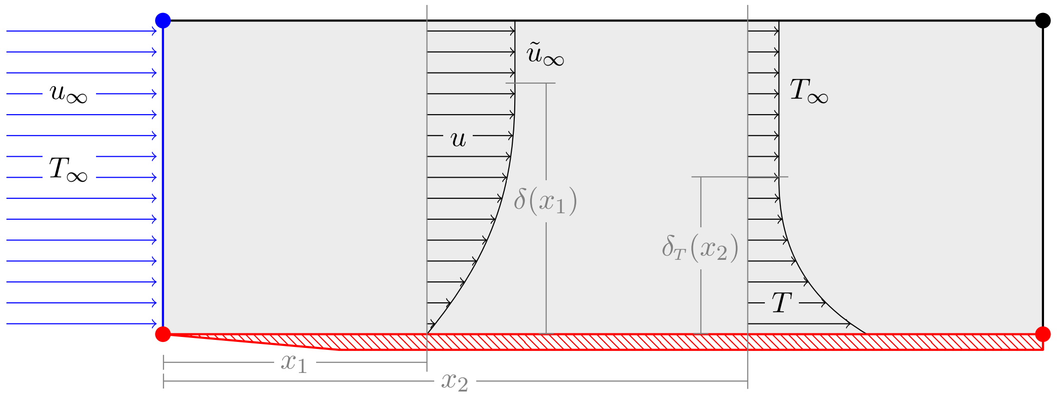


Fig. 4.27: Flat Plate in Parallel Flow, scheme of velocity δ and thermal boundary layer δ_T.

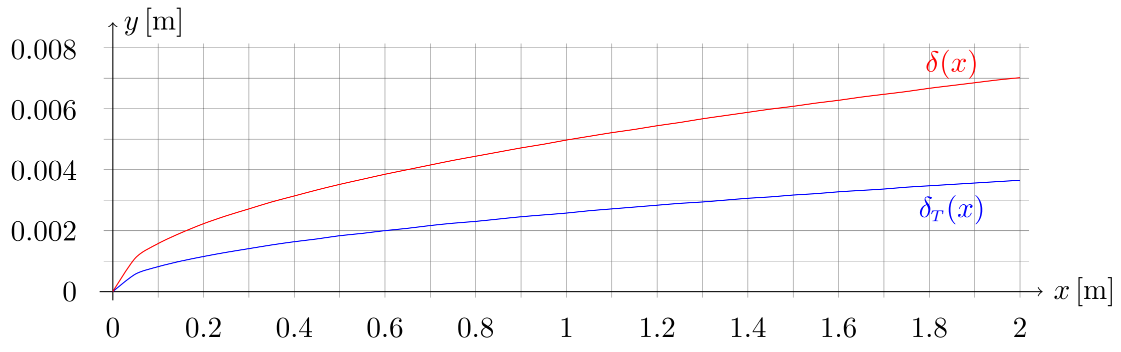


Fig. 4.28: Flat Plate in Parallel Flow, boundary layer widths for water, $\delta(x)$ velocity, $\delta_T(x)$ thermal boundary layer.

in [10], where $\mathrm{Pr} = 7$ for water. As inflow velocity we still have $u_\infty = 1\,\mathrm{m\,s^{-1}}$.

Fig. 4.28 shows $\delta(x)$ and $\delta_T(x)$ for water on a length of 2 m. Note that $\mathrm{Re}_{2\,\mathrm{m}} = 2 \times 10^6$ and the flow would be turbulent in the real world at the end of the box (more precisely the flow would get turbulent at $x = 0.5$, since $\mathrm{Re}_{0.5\,\mathrm{m}} = 5 \times 10^5$). But since our code does not resolve effects of turbulence or turbulence itself, this is not considered any further.

Also note that we have $\delta(2\,\mathrm{m}) = 7\,\mathrm{mm}$(!) for water. The thermal boundary layer is even smaller, since $\mathrm{Pr} > 1$. Therefore we choose a computational domain of size $2\,\mathrm{m} \times 0.01\,\mathrm{m}$, discretized by 1000×10 cells, so that each cell has size $2\,\mathrm{mm} \times 1\,\mathrm{mm}$.

Due to the problems we've seen above gravity is set to zero for this scenario, leading to an unrealistic thermal boundary layer. This is because the density differences in the momentum equation (Eq. (2.42) on Page 11) are neglected, when using $g = (0,0)^T\,\mathrm{m\,s^{-2}}$. Furthermore, the Peano configuration flag `initialize-everywhere` is set, so that the whole domain is initialized with velocity u_∞ (except for the vertices on the plate of course). As time step size we have:

$$\tau \;=\; 1 \times 10^{-6},$$

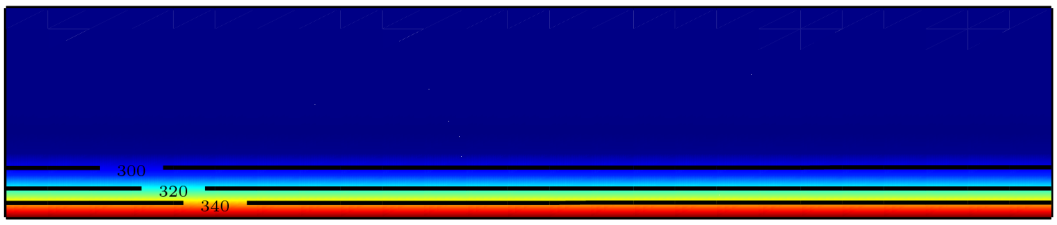
with a total simulation time of $t_{end} = 10\,\mathrm{s}$. The temperatures are:

$$T_H \;=\; 363\,\mathrm{K}, \quad T_C \;=\; T_I \;=\; 293\,\mathrm{K},$$

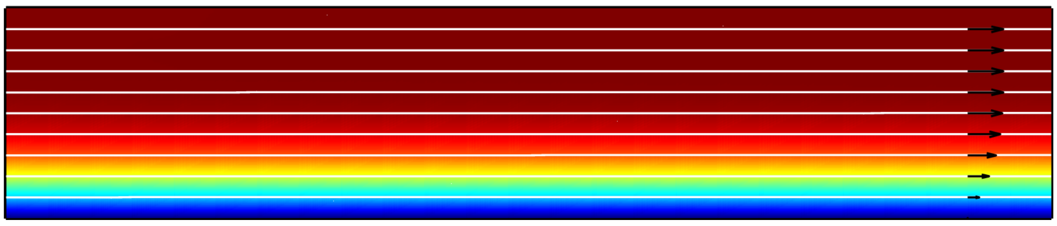
so that we have a temperature difference of 70 K.

In Fig. 4.29 the most right 5 cm of the domain are shown, so that one can see the extract $[1.95\,\mathrm{m}, 2\,\mathrm{m}] \times [0\,\mathrm{cm}, 1\,\mathrm{cm}]$. The arrows in Fig. 4.29(b) indicate the velocity boundary layer, which seems to be nicely curved. Due to the lack of gravity the thermal boundary layer only shows the diffusive heat transport.

Finally, we have Fig. 4.30, where the temperature and the velocities are plotted over vertical cuts of the domain. For the thermal boundary layers in Fig. 4.30(a) we already



(a) Isolines of temperature with color-coded temperature in K.



(b) Streamlines with color-coded velocity magnitude, arrows to indicate boundary layer $\delta(x)$, $\|\vec{u}_{max}\| = 1.17\,\mathrm{m\,s^{-1}}$.

Fig. 4.29: Flat Plate in Parallel Flow, Water, 1000×10 mesh, $t = 10\,\mathrm{s}$, $\tau = 1 \times 10^{-6}$, showing domain $[1.95\,\mathrm{m}, 2\,\mathrm{m}] \times [0\,\mathrm{cm}, 1\,\mathrm{cm}]$.

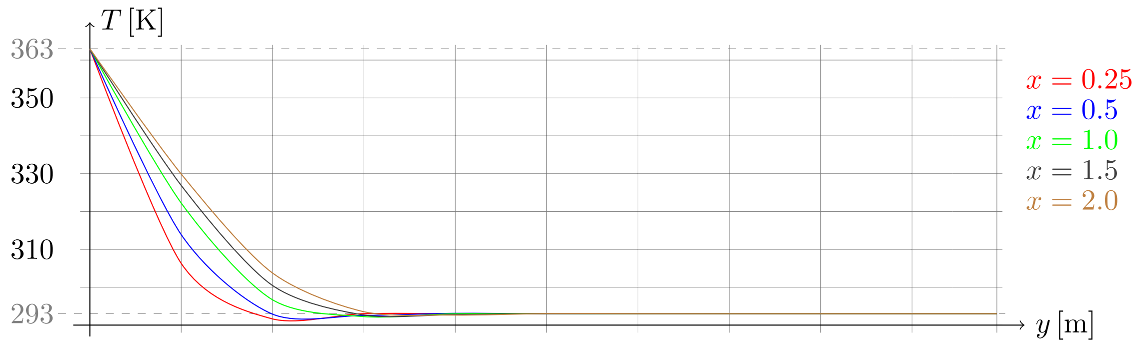
mentioned that buoyancy forces due to density differences are neglected. Therefore, we only see temperature changes that happened diffusively over one or several layers, explaining why these look quite alike. Another thing to mention is that the temperatures are below T_C for $x = 0.25$. This is a phenomena that occurs when using non-square cells and thus can be considered as discretization error [Zenger, personal communication].

The mark, that is plotted for each x in Fig. 4.30(b), is the value of the velocity boundary layer width when using Eq. (4.13). They all almost perfectly fit the boundary layer widths that are computed by Peano, as can be seen in the magnification. The latter also shows horizontal dashed lines for the height of each $\delta(x)$ for easier comparison to Peano's boundary layer widths.

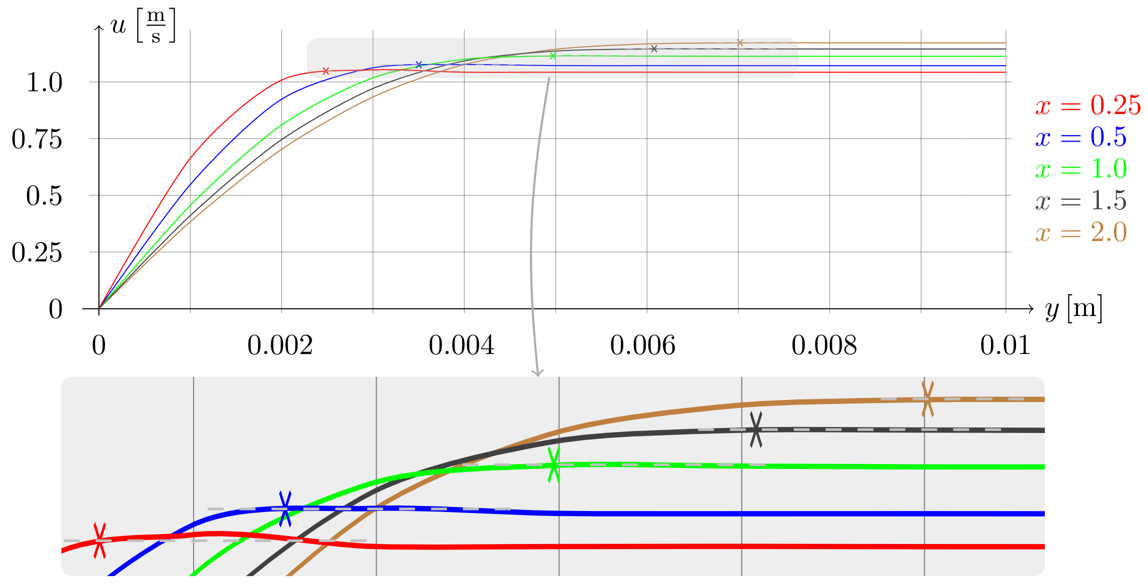
This shows that Peano is able to resolve the phenomena of boundary layers accurately. However, in reality one has much larger domains and cannot afford to refine the mesh on a mm-scale, so that these effects are typically neglected for water. Note that the velocities are higher than u_∞ due to conservation of mass, an effect that has been explained before.

4.4 Adaptive Grid

So far we have only been using the `trivialgrid` component of Peano for our `chemical` simulations. In Section 3.7 the implementation of the (adaptive) `grid` for the `chemical` component is described. The `grid` allows to refine regions with high velocities, high



(a) Thermal boundary layers.

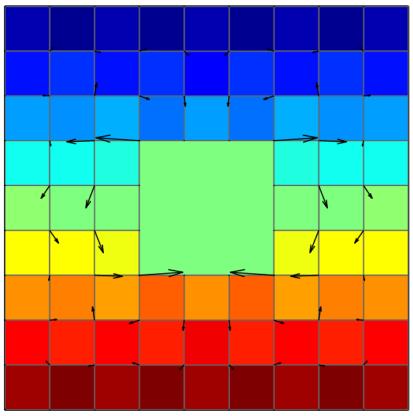


(b) Velocity boundary layers, marks using Eq. (4.13).

Fig. 4.30: Flat Plate in Parallel Flow, Water, 1000×10 mesh, $t = 10\,\mathrm{s}$, $\tau = 1 \times 10^{-6}$, boundary layers ((a) thermal, (b) velocity) over vertical cuts ($x = 0.25\,\mathrm{m}, 0.5\,\mathrm{m}, 1\,\mathrm{m}, 1.5\,\mathrm{m}$, and $2\,\mathrm{m}$) of the domain.

temperature gradients or other regions of interest manually by defining refinement boxes in the domain via the configuration file (see Section 3.9). At boundaries this refinement is done automatically until the finest level is reached.

We now discuss a phenomena that occurs when having different levels of refinement (e.g., Fig. 3.3(b)) and the gravity is non-zero. A scenario that has no external flow, such as a natural convection scenario, typically produces very small velocities. Fig. 4.31 compares a scenario that has no external flow (`gravity-box`


(a) Dipole field in a `grid` simulation without external flow (`fluid` scenario `gravity-box`), $\|\vec{u}_{max}\| = 1.22 \times 10^{-3}\,\mathrm{m\,s^{-1}}$, $T_H = T_C = T_I = 0$.

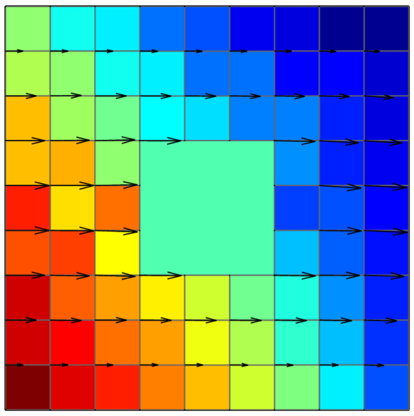

(b) No visible dipole field due to higher velocities in scenario with inflow (`obstacle-in-channel`), parabolic velocity profile, $\|\vec{u}_{max}\| = 1.48\,\mathrm{m\,s^{-1}}$.

Fig. 4.31: Comparison of scenario without (a) and with (b) external flow on same mesh ($h = h_{min} = 1/9$, $H = h_{max} = 1/3$), color-coded pressure, chemical scenario for both `rayleigh-benard-convection`, $u_\infty = 1\,\mathrm{m\,s^{-1}}$.

with `rayleigh-benard-convection`, Fig. 4.31(a)) and a scenario with external flow (`obstacle-in-channel` with `rayleigh-benard-convection`, Fig. 4.31(b)).

Both have the same mesh with one coarse cell at the center ($H = 1/3$) and refined cells ($h = 1/9$) at the domain boundary. Since there is no temperature gradient ($T_H - T_C = 0$) there should be no flow at all in the natural convection scenario (Fig. 4.31(a)). Nevertheless, we see a dipole field with a maximal velocity of $\|\vec{u}_{max}\| = 1.22 \times 10^{-3}\,\mathrm{m\,s^{-1}}$. The driven flow (Fig. 4.31(b)) does not show this dipole field, *falsely* suggesting that there is none.

The reason for this dipole field is a discretization error that is independent of the `chemical` component. The pressure is a cell degree of freedom, so that it is located in the center of a cell. The ansatz functions for the pressure are piecewise constant (see Section 3.2). Since the velocities are located at the vertices and we need the pressure – or better the force – for the calculations of the update in the momentum equation, this force has to be spread onto the vertices (Fig. 4.32). The hydrostatic pressure levels $p_0 < p_1 < p_2$ are dependent on the gravity g of course (Eq. (4.12)). Note that cell A and C both have value p_1 because their cell centers are located on the same height (with respect to g).

On stepping up to the coarser grid, node N accumulates not only the values of cells A and B directly but also the values of the hanging nodes are restricted. This is what causes problems, since the force on node N of cell A is larger than the accumulated forces of cells B, C and D. This leads to an outward pointing velocity contribution – seen from the

Table 4.4: Maximal velocity in dipole field at time $t = 1\,\mathrm{s}$, error in order of $\mathcal{O}(h^2)$, grids with maximal mesh size H and minimal mesh size h.

H	h	$\|\vec{u}_{max}\|$ $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$ at $t = 1\,\mathrm{s}$
$1/3$	$1/9$	1.64×10^{-2}
$1/9$	$1/27$	2.26×10^{-3}
$1/27$	$1/81$	2.54×10^{-4}
$1/81$	$1/243$	2.82×10^{-5}

coarser cell. At the bottom of cell A the same happens in the opposite direction, since the force pushing from cells B, C and D is larger than the force pushing from cell A.

In the box scenario (Fig. 4.31(a)) these are the only velocity contributions that exist, resulting in a dipole field. Note that the error is in order of $\mathcal{O}(h^2)$, since the we use bi-linear FEM elements (Table 4.4). Also note that this error occurs for all FEM approaches, where using different elements might eliminate the error in this scenario but results in the occurrence an error elsewhere.

This phenomena has not been seen in Peano before, because there was no scenario without external flow. Natural convection scenarios thus should not use the adaptive `grid`, when the velocities are expected to be slow – which is dependent on the fluids parameters of course. As soon as an external flow is involved it should not play a vital role and the `grid` can safely be used. But note that the discretization error is still there and typically just not visible in the output, because it is too small.

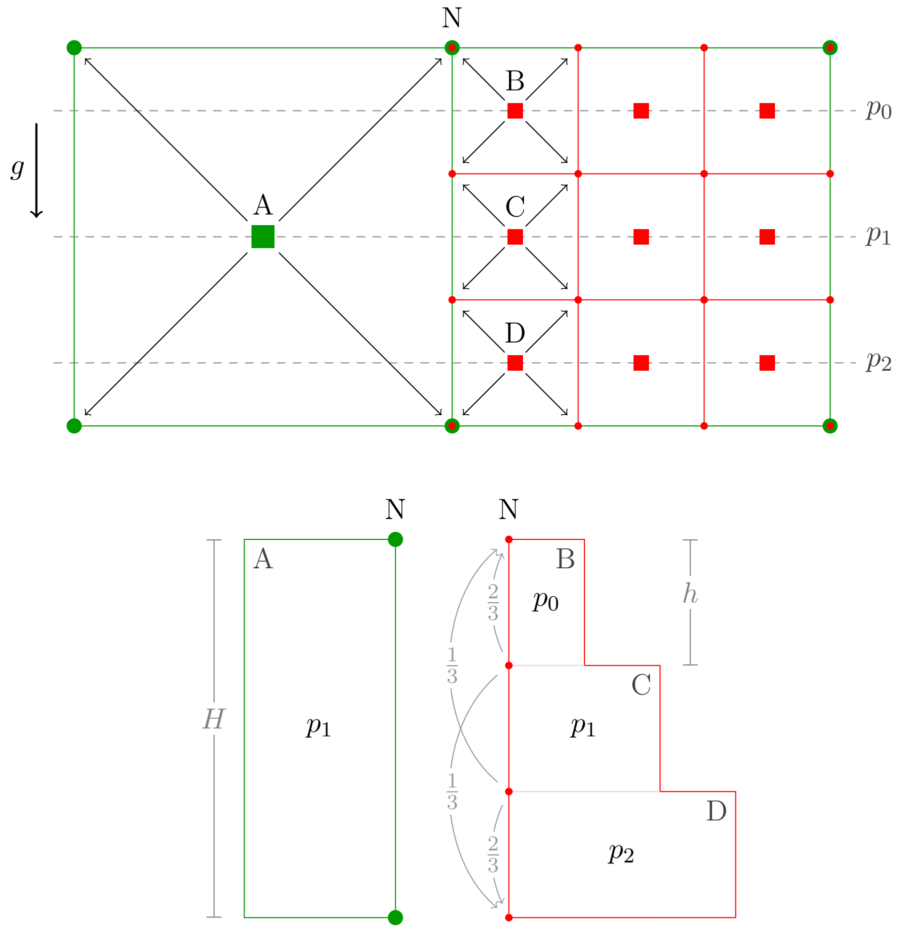


Fig. 4.32: Peano distribution of pressure in a cell with hydrostatic pressure levels p_0, p_1 and p_2 and cell sizes H and $h = {}^{H}/3$ with restriction of hanging nodes.

5 Application to Reactor Safety

In a nuclear power plant (NPP) power is produced by controlled nuclear fission. The released heat is used to heat a fluid (typically water) for generation of electricity. Different types of NPPs exist that use water as coolant and moderator (for neutron deceleration), such as heavy water reactors (HWR) and light water reactors (LWR). The latter can be subdivided into two different types of reactors:

- In a boiling water reactor (BWR) steam is produced in the reactor core (approx. 70 bar, 285 °C).The steam drives a turbine before it is cooled down in a condenser.
- In the primary cooling circuit of a pressurized water reactor (PWR) water is heated but – due to the pressure (approx. 155 bar) – does *not* boil (approx. 310 °C). In the steam generator heat is transferred to a secondary coolant, which also is water. The second coolant boils and drives a turbine to generate electricity.

On normal operation the primary coolant in a PWR is heated in the reactor pressure vessel (RPV) and flows through a pipe – the so called *Hot Leg* – into the steam generator (Fig. 5.1). The steam generator connects the primary and the secondary cooling circuit, transferring the heat to the secondary coolant. The primary coolant is then pushed through the *Cold Leg* back into the RPV by the reactor coolant pump (RCP).

A loss-of-coolant accident (LOCA) requires the injection of emergency core coolant (ECC) for the cooling of the reactor core. The ECC is pumped into the cold leg through a safety injection nozzle. For reactor safety analysis it is important to describe the mixing of the ECC (30 °C) and the hot primary coolant in the cooling circuit accurately, since the high temperature gradients may lead to a thermal shock related fracture of the reactor pressure vessel.

5.1 Upper Plenum Test Facility (UPTF)

The mixing of the ECC with the primary coolant was experimentally investigated in the frame of the UPTF-*Transient and Accident Management* (TRAM) test series. The UPTF is a geometrically full-scale test facility with a (non-radioactive) core simulator that is used to simulate the primary circuit of a PWR [13].

The distribution of the temperature in the cold leg and the annulus of the RPV has been examined in the UPTF-TRAM test series C1 and provides an exhaustive data basis for the validation of Peano and the implemented energy equation. Considering UPTF-TRAM C1 Run1a01, the specifications and results can be summarized as follows:

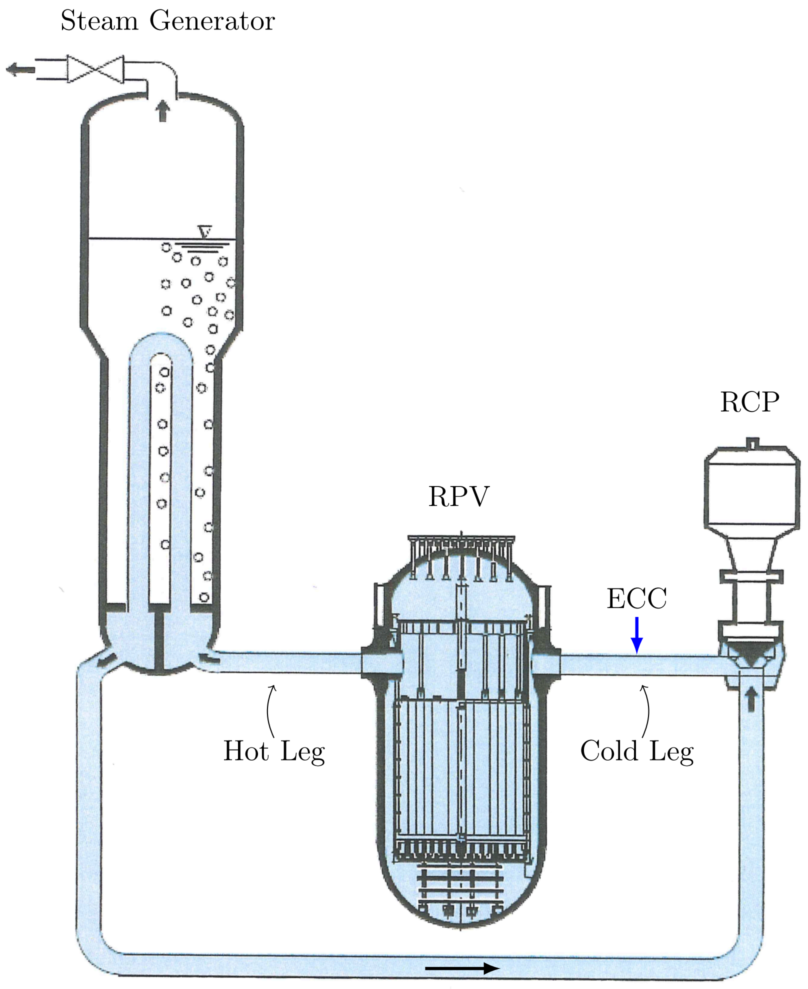


Fig. 5.1: Scheme of a pressurized water reactor with reactor pressure vessel (RPV), hot leg, steam generator, reactor coolant pump (RCP), cold leg and the injection nozzle of emergency core coolant (ECC).

- Before ECC injection the flow is stagnant in the cold leg at approx. 18 bar and 190 °C.
- The cold leg is always filled with liquid water, so only single-phase phenomena were considered.
- There is a turbulent mixing of cold and hot water close to the safety injection nozzle.
- A stratification of hot and cold water along the cold leg was obeserved.

5.2 Peano Implementation

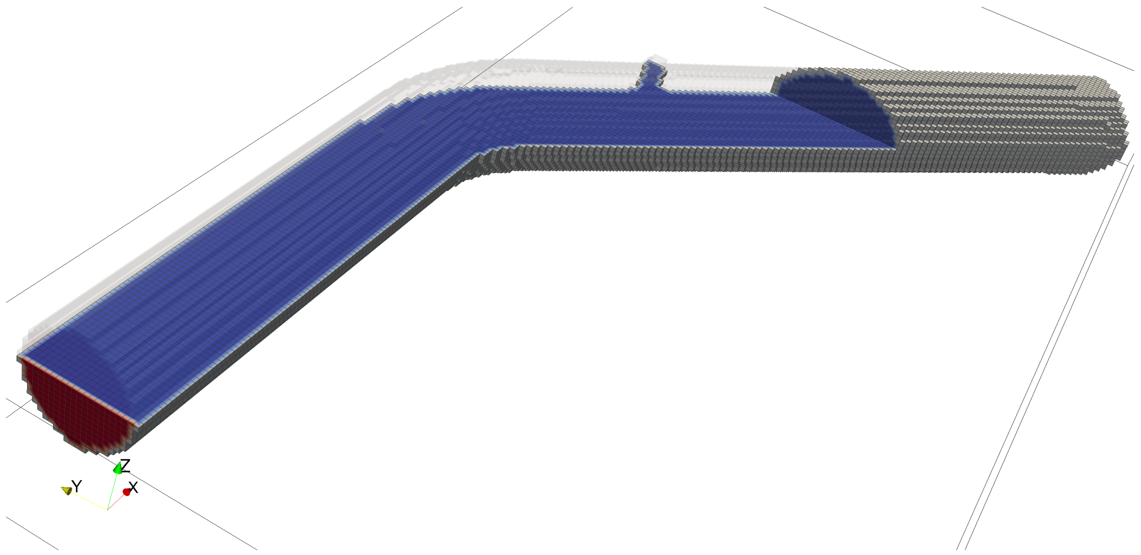
GRS provides a CAD-model of the cold leg, that is used by Peano in combination with the tool *preCICE*, which has been developed at the Technische Universität München [2]. preCICE is used to fill the gap between the CAD-model and Peano's geometry, that can tell whether or not a voxel is inside or outside of the domain. The geometry also defines boundary numbers for the safety injection nozzle, the boundary towards the RPV and the union of all other boundaries. A notable advantage of Peano is, that the mesh generation, both regularly and adaptively, happens fully automatic (see Section 3.7). In order to compile the `chemical` component of Peano with preCICE one can use the SCons target `fluid-chemical-precice` (see Section 3.9).

The `fluid` scenario `precice-cold-leg` then applies boundary conditions for the velocity $\vec{u}$: a Dirichlet boundary condition for the inlet (safety injection nozzle), a Neumann boundary condition for the outlet (open boundary towards RPV) and no-slip Dirichlet conditions (walls) for the rest of the boundaries. The temperature boundary conditions are set by the `chemical` scenario `chemical-cold-leg`: a Dirichlet boundary condition for the inlet and Neumann boundary conditions for all other boundaries (walls are assumed to be adiabatic and the outlet adapts temperature according to temperature distribution in the pipe).

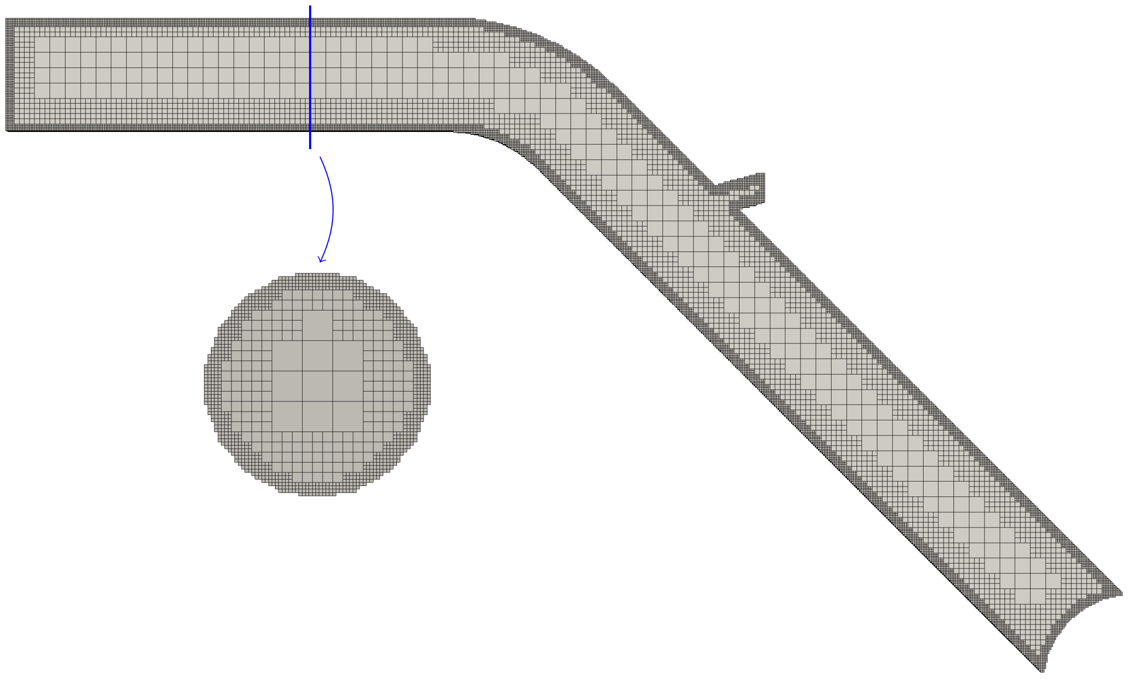
Both the `trivialgrid` and the `grid` component of Peano can be used for mesh generation. Technically, the cold leg is embedded in a rectangular box. So, when using a `trivialgrid` with $270 \times 150 \times 60$ cells (approx. cell size $3.3\,\text{cm} \times 3.3\,\text{cm}$) the complete box is partitioned into this number of cells and only some (approx. 85 000 cells) are inside of the cold leg (Fig. 5.2(a)). An adaptive `grid` using $h_{min} = {}^{1}\!/_{729}$ (fraction of edge length of embedding cube) has approximately 915 000 inner cells (see longitudinal and cross section in Fig. 5.2(b)). It is possible to use refinement boxes to refine certain areas of the scenario manually. The left end of the cold leg (marked red in 5.2(a)) represents the outlet towards the RPV.

The cold leg geometry – in contrast to channel-like scenarios – provides a challenging test setup for Peano exactly because of the embedding mentioned. Furthermore, Peano's `trivialgrid` component has been designed for evaluation and comparison purposes. Hence, current runtimes – depending, of course, on the mesh resolution and the number of time steps – are in the order of two weeks for purely serial computations of fluid flows of about 100 seconds in simulation time scale. The streamlines of an on-going computation of a `fluid`-only scenario with time step size $\tau = 1 \times 10^{-4}$ at time $t = 0.3\,\text{s}$ convey a positive impression for further simulations (Fig. 5.3).

It remains to do a thorough analysis of the cold leg and a validation using the UPTF-TRAM test series results. Nevertheless, to do so, the work done provides the theoretical and technical base for Peano and its validation, which are all essential prerequisites for any further simulation. Future developments of Peano, both mathematical enhancements of the model and numerical improvements regarding runtime etc., will be discussed in Chapter 6.



(a) `trivialgrid`, 270 × 150 × 60 mesh of embedding box, 85 000 cells inside cold leg.



(b) `grid`, $h_{min} = 1/729$, 915 000 cells, longitudinal and cross section.

Fig. 5.2: `trivialgrid` and `grid` mesh of cold leg, outlet to RPV on left side (red in (a)).

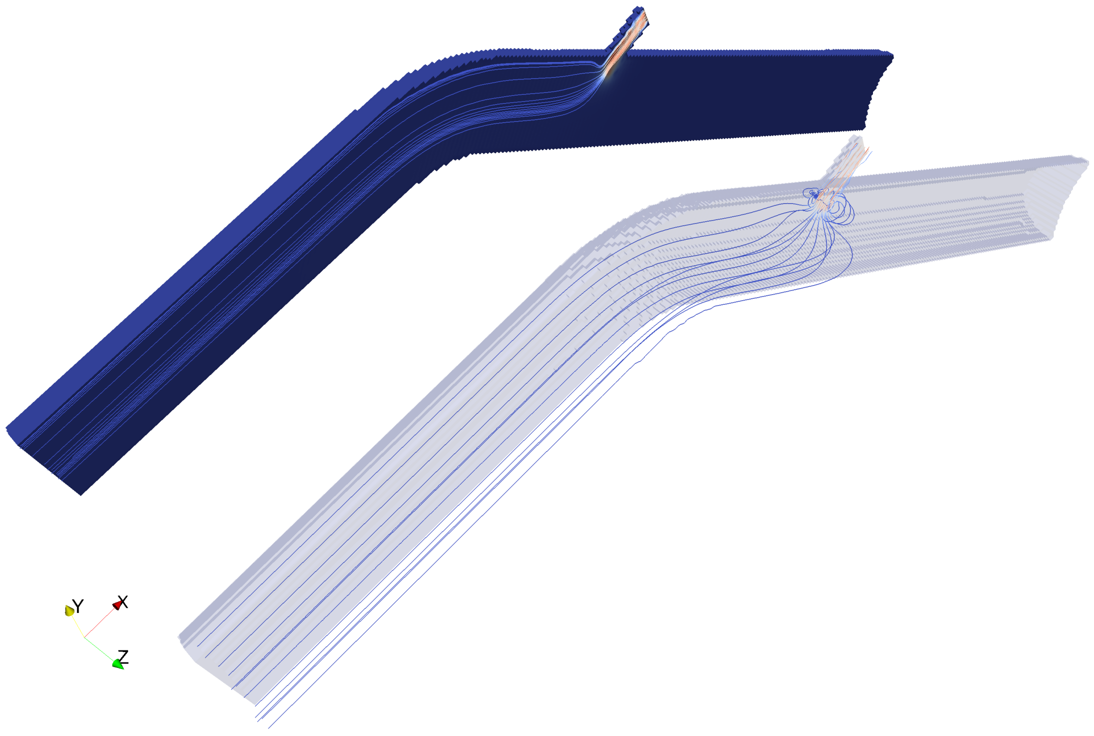


Fig. 5.3: Cold Leg, $270 \times 150 \times 60$ mesh, $\tau = 1 \times 10^{-4}$, $t = 0.3\,\mathrm{s}$, velocity streamlines, $\|\vec{u}_{max}\| = 2.55\,\mathrm{m\,s^{-1}}$.

6 Conclusion

This thesis is closed by giving a short summary of the results and an outlook on the current development activities of Peano and in particular possible further extensions of the CFD component.

Summary

Peano is a memory efficient `C++` framework for solving PDEs on regular and adaptively refined Cartesian grids. The mathematical model of the existing CFD solver component has been extended by the energy equation, representing a classical convection-diffusion equation for the temperature. The Boussinesq approximation is used to approximate density differences in gravitational terms and thus includes occurring buoyancy forces.

The cell-wise operator evaluation approach of Peano allowed to easily extend the existing code by the energy equation. The general concept of Peano as well as implementation details like the *Decorator Pattern*, used to couple a `HeatScenario` and a `FluidScenario`, have been explained thoroughly (see Section 3). The in-house tools DaStGen and `preCICE` are used to support the modular implementation of Peano and allow faster development.

We have demonstrated that Peano is able to cope validation scenarios, that have been used for both qualitative and quantitative analysis. Most of the simulations have been two-dimensional since simulating a three-dimensional flow not only leads to longer computational times but there is also less data to validate it with. Nevertheless, the concrete implementation of the Peano's operators is independent of the dimension and thus three-dimensional flows have been investigated as well. A summary of the validation scenarios and the mesh sizes that have been used for our simulations are shown to give a general overview of the validation process (Table 6.1). Natural convection scenarios like *Natural Convection with Heated Lateral Walls* and *Rayleigh-Bénard Convection* show that the current implementation of the `chemical` component simulates flows with convective and diffusive heat transport accurately. Nevertheless, since there is not (yet) a model for turbulence implemented for this component, the simulation of water in such scenarios requires a fine spatial discretization.

Additional features like the hydrostatic pressure correction at open boundaries have been added to the existing implementation of the `fluid` component. Boundary layers are examined in the *Flat Plate in Parallel Flow* scenario and especially the velocity boundary layer is resolved accurately. Furthermore, we explained a phenomenon related to the

Table 6.1: Summary of computed scenarios with fluids and mesh sizes. Asterisk * for stretched meshes (see Section 4.1.3).

Scenario	Fluid	Meshes used for Simulations
Natural Convection with Heated Lateral Walls Section 4.1	Griebel 1	20×20, 50×50, 100×100
	Griebel 2	25×25, 50×50, 100×100, 200×200, 400×400 $50 \times 50 \times 50$
	Artificial Fluid	50×50 Ψ, 100×100, 250×250, 400×400, $50 \times 50^*$, $100 \times 100^*$, $200 \times 200^*$
	Water	$200 \times 200^*$ Ψ, $400 \times 400^*$
Rayleigh-Bénard Convection Section 4.2	Glycerine	49×5, 227×21
	Air	65×17 Ψ, 130×34, 260×68
	Water	500×50
Flat Plate in Parallel Flow Section 4.3	Water	1000×10

hydrostatic pressure arising due to the choice of FEM elements (bi-linear for the velocity, piecewise constant pressure), which has not been seen in Peano before (see Section 4.4).

The coupling of preCICE and Peano allows the application of Peano on a reactor safety scenario, the temperature mixing in the *Cold Leg* during ECC injection. We provided the technical base that is necessary for further validation of this scenario, meaning the possibility to create a mesh using the CAD-model that is provided by GRS, and performing first `fluid` simulations on it.

Outlook

The current implementation of the `chemical` component of Peano provides a solid base for simulations involving temperature. Possible enhancements as the inclusion of a turbulence model (e.g., a $k - \epsilon$-model or the computation of turbulence with a Large-Eddy Simulation (LES)), will allow the simulation of turbulent flows. As the possibility to calculate the time step size adaptively, dynamic mesh refinement of regions in the domain with high velocity or temperature gradients could provide more accurate results.

At the same time this feature could be used to coarsen regions with small gradients, optimizing computational costs.

Another enhancement that is currently being integrated in Peano is the modeling of free surface flow and dispersed two phase flows, e.g., simulation of bubbles in water.

The `parallel` component of Peano is available for the `fluid` but not for the `chemical` component, yet. The same holds for an implicit time integration method to solve the pressure Poisson equation. Both can be implemented to allow finer meshes and reduce the computational time for a simulation.

The core of Peano is under strong development and recently Peano 2 has been released. The `fluid` component is not yet ported to this new version, but once it has been, new features and, in particular, parallelization can be integrated even more easily and faster.

Bibliography

[1] D. Braess. *Finite elements – Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, 3 edition, 2007.

[2] H.-J. Bungartz, J. Benk, B. Gatzhammer, M. Mehl, and T. Neckel. *Fluid-Structure Interaction – Modelling, Simulation, Optimisation, Part II*, volume 73 of *LNCSE*, chapter Partitioned Simulation of Fluid-Structure Interaction on Cartesian Grids, pages 255–284. Springer, Berlin, Heidelberg, Oct. 2010.

[3] H.-J. Bungartz, W. Eckhardt, M. Mehl, and T. Weinzierl. *ICCS 2008: Advancing Science through Computation, Part III*, volume 5103 of *Lecture Notes in Computer Science*, chapter DaStGen - A Data Structure Generator for Parallel C++ HPC Software, pages 213–222. Springer-Verlag, Heidelberg, Berlin, June 2008.

[4] J. H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, 3 edition, 2001.

[5] P. M. Gresho and R. L. Sani. *Incompresible Flow And The Finite Element Method*. John Wiley & Sons, 1998.

[6] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics – A Practical Introduction*. SIAM Monographs on Mathematical Modeling and Computation, 1998.

[7] H. Herwig. *Strömungsmechanik – Eine Einfürung in die Physik und die mathematische Modellierung von Strömungen*. Springer-Verlag, 2006.

[8] C. Hirsch. *Numerical Computations of Internal and External Flows, Computational Methods for Inviscid and Viscous Flows*, volume 2. John Wiley & Sons, 1990.

[9] C. Hirsch. *Numerical Computations of Internal and External Flows, Fundamentals of Numerical Discretization*, volume 1. John Wiley & Sons, 2001.

[10] F. P. Incropera and D. P. D. Witt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 3 edition, 1990.

[11] T. Neckel. *The PDE Framework Peano: An Environment for Efficient Flow Simulations*. Dissertation, Institut für Informatik, Technische Universität München, June 2009.

[12] W. Polifke. *Wärmetransportphänomene*. Lehrstuhl für Thermodynamik, Technische Universität München, 2003. Skriptum zur Vorlesung.

[13] Siemens AG, Erlangen. *Versuch C1/C2 Strähnen- und Streifenkühlung der RDB-Wand*, Mar. 1996. NT31/96/17.

[14] M. F. Tome and S. McKee. GENSMAC: A Computational Marker and Cell Method for Free Surface Flows in General Domains. *Journal of Computational Physics*, 110:171–186, 1994.

[15] T. Weinzierl. *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids.* Dissertation, Institut für Informatik, Technische Universität München, 2009.

[16] J. E. Welch, F. H. Harlow, J. P. Shannon, and B. J. Daly. The MAC Method – A Computing Technique for Solving Viscous, Incompressible, Transient Fluid-Flow Problems Involving Free Surfaces. *Los Alamos Scientific Laboratory of the University of California*, March 1966.