NTNU
Norwegian University of
Science and Technology

Numerical Simulation of Friction Losses in Tunnels with Strip Roughness

Erik Mølmann

Civil and Environmental Engineering
Submission date: May 2018
Supervisor: Nils Reidar Bøe Olsen, IBM
Co-supervisor: Pierre-Yves T. Henry, IBM

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

Abstract

To estimate friction introduced by plastering strips inside of laboratory tunnels, an equation named after Reidar Birkeland is used at NTNU. In this thesis, four different geometries with three different strip distances are simulated numerically using OpenFOAM to test the accuracy of the equation. The calculated friction factors from 36 runs are analyzed. Other factors, including how length to development and secondary currents interact with the strip roughness, are also looked at.

With a 24 percent average overestimation of the friction factor λ, the Birkeland equation is decidedly out of its domain of accuracy in these cases. To ensure reliable head loss measurements, length to development was investigated. Starting out with a uniform velocity profile gave values between 2 and 3.5 of $C = \lambda L/D$, with L being length to development and D being four times the hydraulic radius, for most geometries and flow velocities. An implicit assumption made by Birkeland was that the strips act as isolated roughness elements, with one not being affected by the former. This was found to be true for all cases tested.

Preface

This thesis marks the end of my Master's degree in Civil and Environmental Engineering at the Norwegian University of Science and Technology, NTNU.

I would like to thank my supervisors, Nils Reidar B. Olsen and Pierre-Yves T. Henry, for their contributions and feedback during the thesis work. The simulations in this thesis were done on the supercomputer Fram, whose processors have served me well.

Erik Mølmann
Trondheim, 30.05.2018

Contents

Acronyms

CC Center-to-Centre (distance)

CFD Computational Fluid Dynamics

CFL Courant-Friedrichs-Lewy (condition)

CLI Command-Line Interface

DES Detatched Eddy Simulation

DNS Direct Numerical Simulation

GUI Graphical User Interface

LES Large Eddy Simulation

LRR Launder, Reece and Rodi (model)

PDE Partial Differential Equation

PDF Probability Density Function

PIV Particle Image Velocimetry

RANS Reynolds-Averaged Navier-Stokes

SIMPLE Semi-Implicit Method for Pressure-Linked Equations

Chapter 1

Introduction

Real tunnels have real roughness, leading to head loss as water flows through them. When doing experiments in the laboratory, this roughness must somehow be replicated. In the hydraulic laboratory at NTNU, the most common method is to glue strips on the inside of otherwise smooth acrylic glass tunnels. Examples of this from 2017 include several master theses; Brøste [2017], Gjerde [2017], Perzyna [2017] and Ekeade [2017]. The equation used by all of them to estimate the friction introduced by these strips is called the Birkeland equation.

1.1 Preexisting methods for quantifying strip roughness

To estimate the roughness introduced by discrete elements in a channel, a few different methods exist. Morris [1963] gives formulas for the drag experienced in pipes with different spacing of roughness elements. The formula used depends on if one element affects the flow situation at the next, and he distinguishes between quasi-smooth, hyper-turbulent and isolated roughness flow. In quasi-smooth flow the depressions between elements are short, and the recirculation happening there does not affect the macroscopic flow in the main channel. Hyper-turbulent flow is distinguished by the wake structures of each element affecting the next one. Isolated roughness flow occurs when each element can be treated as a separate source of form drag, with no influence of the previous one. For the rough part of the friction with isolated roughness, Morris [1963] postulates this scaling:

$$\lambda_{rough} \propto \lambda_{smooth} C_D \frac{L_r}{P} \frac{h}{r_i} \frac{r_i}{L}. \tag{1.1}$$

Here L_r is the peripheral length of the roughness element, P is wetted perimeter of the pipe, r_i is the pipe radius inside the elements, h is height of the roughness elements and L is the distance between them. C_D is the drag coefficient of the discrete elements' surfaces.

1.2 Development of the Birkeland equation

Due to shortcomings of the formula of Morris [1963] in predicting the friction factor in his tests, Birkeland [2008] decided to make a new one. The goal was that a new equation would give better results for the type of tunnels used in the laboratory. He used the same concept of splitting the friction contributions of the smooth and rough parts, and the variables used are loosely based on equation 1.1. Based on the geometry of his case, isolated roughness flow was assumed to be present. This implies that the wake structure of an element is not affecting the drag introduced by ones downstream, setting a lower bound on strip spacing used with the resulting equation. Birkeland [2008] uses this scaling:

$$\lambda_{rough} \propto C_0 \, (\frac{L}{4R})^{C_1} \, (\frac{h}{4R})^{C_2} \, (\frac{L_r}{4R})^{C_3}, \tag{1.2}$$

with R being the hydraulic radius of the tunnel and L_r being the length of the roughness element. L_r equals the wetted perimeter, P, if the strip is plastered around the whole periphery of the tunnel. The differences between the equations of Morris [1963] and Birkeland [2008] is that the former scales the rough friction with the smooth one, uses the calculated drag coefficient for the roughness shape and that it has the ratio of constriction to intra-element distance as a factor.

To tune the coefficients, several experiments where conducted using the setup showed in figure 1.1. Head losses over a 1.61 meter stretch in a laboratory tunnel were recorded using two pressure measurements. A filter was installed at the inlet to avoid coherent flow structures from the inlet pipe convecting into the studied domain. The resulting friction factors were calculated using the Darcy-Weisbach equation:

$$Head\ loss\ [mH_2O] = \lambda \, \frac{x_2 - x_1}{4R} \, \frac{u^2}{2g} \, . \tag{1.3}$$

The investigated variables are shown in table 1.1. Including the base case without strips, a total of 20 configurations were ran three times each. Based on a least-square regression of

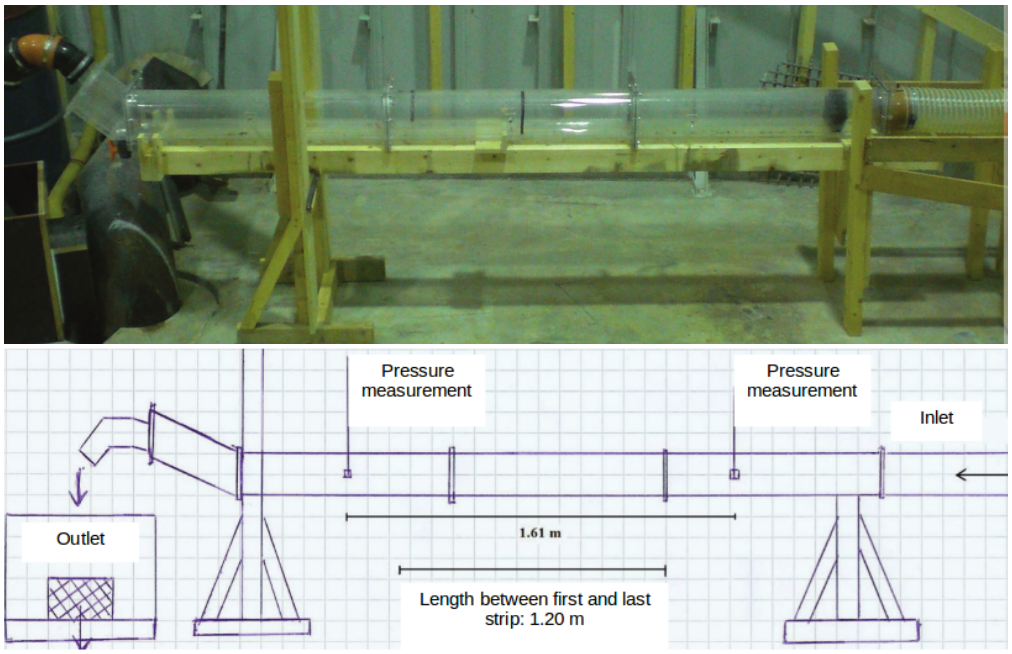

Figure 1.1: Experimental setup used by Birkeland [2008].

Variable	Configurations
L [m]	0.1 (not for h=5mm or h=9mm), 0.2 (not for h=9mm), 0.4, 0.6
Q [l/s]	19.1, 28.1
h [mm]	3, 5, 9

Table 1.1: Configurations used in tuning the Birkeland equation.

the powers and a constant, the resulting equation reads:

$$\lambda_{rough} = 8.43(\frac{L}{4R})^{-0.796}(\frac{h}{4R})^{1.655}(\frac{L_r}{4R})^{1}. \tag{1.4}$$

1.3 Critique of the equation

To put this section in context, the Birkeland equation was not meant to be a universal law of nature. It was highly tuned for the work done in the master thesis it is a part of, and is not a result of rigorous testing by Reidar Birkeland. It has, however, been put to great use in the laboratory at NTNU, and is thus deserving of a closer look.

The isloated roughness flow assumption

In structuring his equation after Morris [1963] and equation 1.1, Birkeland [2008] implicitly assumes isolated roughness flow. This is a priori only valid over a certain range of velocities and strip distances for each geometry, with an unknown relationship between the variables. When this bound is overstepped, strips are hit by the wake of the one prior, rendering rough surface vortices as the main source of drag (Morris [1963]). For the strip distances and velocities subjected to study by Birkeland, this was not reported to be the case.

Strip placement

All of Birkelands tests seem to have been done without strip roughness on the floor of the tunnel. Although not mentioned explicitly in his thesis, all pictures show the same method of application. Today, several of the tunnels in the lab have strips glued around the whole perimeter. Figure 1.2 shows two pictures of Birkelands usage of the strips, paired with two examples of how it is used in the lab today. The difference is in theory being accounted for by the factor L_r, giving the length of the roughness elements. As the flow was not constricted in all directions by Birkeland [2008], one might however experience different effects when it is.

Multiphase flow

All of Birkelands experiments were done with the tunnel full of water. Both Gjerde [2017] and Brøste [2017] have a water-air mixture running through their tunnels. This comes with an increase in head loss as a result of the energy expended in mixing the two (Nestmann [2017]).

Flow development

As seen in figure 1.1, the velocity profile of the inflow pipe is broken up by a filter. This kills coherent structures, and is regarded as good practice. There can however be traces of the inflow conditions far downstream, even with several turbulence generating trips in the form of strips underway. In Garcia [2017], the inflow pipe had a sideways bend right before the inlet to the tunnel. 5.11 meters downstream of the inlet, equalling 32 times the

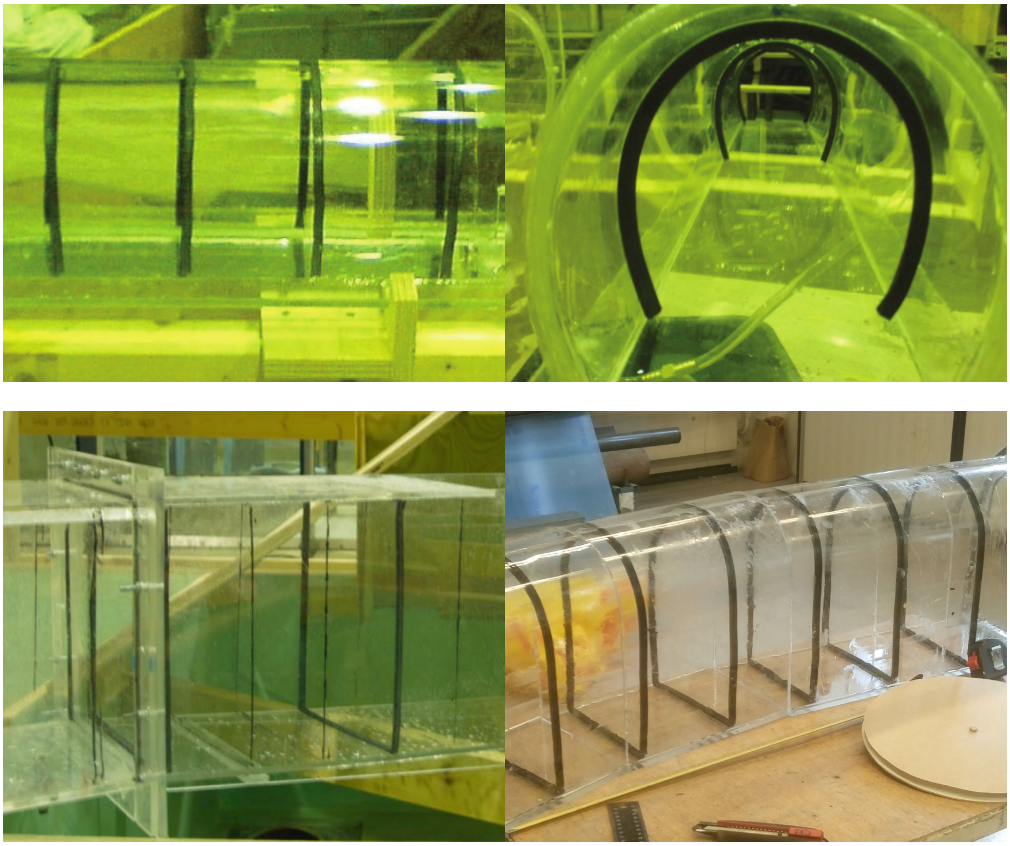

Figure 1.2: Applications of strip roughness. Top: Two pictures of the setup used by Birkeland [2008]. Bottom: Setups used by Brøste [2017] to the left and Gjerde [2017] to the right.

hydraulic diameter, the horizontal velocity profile was still skewed. No length of development is prescribed when using the equation, and only a global friction factor is calculated. The effective friction might change downstream until the flow is fully developed, at which point the pressure loss is uniform between sections. Length to development is therefore a factor to consider when setting up experiments using strip roughness, but one that is not taken into account in the equation or its use. All experiments were performed with the same distance between measurement points, and were thus unable to see any downstream change in friction.

Shape effects

The Birkeland equation scales all variables with hydraulic diameter, four times the hydraulic radius. When using the Darcy-Weisbach equation to calculate head loss, velocity squared comes in as an additional factor. Shape effects are however neglected. Secondary flow patterns are strongly affected by the shape of the cross section (Miller [1990]), and this can

thus alter how large the effect of the strips is on head losses. All experiments done by Reidar Birkeland were on the same cross section shape and size.

Inflow effects

Depending on the experimental configuration, the flow might come in swirling, accelerated or decelerated. This has the possibility of affecting energy loss (Pope [2000]) and length to development (Zagarola and Smits [1998]) but can be very hard to quantify a priori. As mentioned above, swirl was detected over 30 hydraulic diameters downstream of the inlet by Garcia [2017], even with the use of a filter at the inflow.

1.4 Contents and structure of this thesis

Besides the precision of the equation itself, all of the aforementioned factors are possible topics of study. In this thesis, four different cross section shapes are simulated numerically to look at the isolated roughness flow assumption, length to development, strength of secondary flow and strip induced roughness. This is done in chapter 6 after a walk through of the theory, numerical methods and software used. The results are discussed in chapter 7.

Chapter 2

Theory of fluid motion

To facilitate the discussions about the methods used in this thesis, the central theory needed is presented in this chapter. This includes the assumptions made in calling a fluid continuous, via the Navier-Stokes equations to the models used to solve them and the numerical methods employed in doing so.

2.1 General assumptions

2.1.1 Continuum hypothesis

At a microscopic level, a fluid consists of molecules in constant motion. The ratio of the mean free path between these particles, l, and the smallest macroscopic flow structure, L, is the Knudsen number $Kn \equiv l/L$. To treat the fluid as a continuum, $Kn \ll 1$ is a constraint. One can examine air at atmospheric pressure as an example. The mean free path is 10^{-8} m, and the average time between collisions for a particle is 10^{-10} s (Pope [2000]). When dealing with a system of small length scales, say 10^{-4} m, and high velocities, 10^{2} m/s, the time scale is 10^{-6} s. Our Knudsen number is then $Kn = 10^{-8}\ m/10^{-4}\ m = 10^{-4}$, and satisfies the criteria for a continuum. When dealing with water, which is denser than air, and systems with larger scales and lower velocities, the Knudsen number will be orders of magnitude smaller, and it can safely be treated as a continuum.

2.1.2 Newtonian fluids

All the fluids used here are assumed to be Newtonian, defined as fluids where the viscous stresses scale linearly with the local strain rate.

2.1.3 Incompressibility

The assumption of incompressibility is made in the making of the subsequent equations. This excludes flows at high Mach numbers, flows of mixed fluids and flows with large internal differences in density or temperature.

2.2 Conservation laws

2.2.1 Conservation of mass

The equations of fluid motion are based on the notion of a control volume with a surface $\mathcal{S}$, a volume $\mathcal{V}$ and a unit normal vector $\mathbf{n}$. The mass of the control volume is then:

$$M = \int_{\mathcal{V}} \rho \, dV \tag{2.1}$$

The net flow across the surface is given by

$$\int_{\mathcal{S}} \rho(\mathbf{u} \cdot \mathbf{n}) dS, \tag{2.2}$$

and without internal sources, mass conservation implies that it equals the change in mass:

$$\frac{d}{dt} \int_{\mathcal{V}} \rho \, dV = - \int_{\mathcal{S}} \rho(\mathbf{u} \cdot \mathbf{n}) dS. \tag{2.3}$$

The minus sign entails that mass is lost when it flows out of the control volume. Pulling the time derivative inside of the integral can be done due to the control volume being constant in time. Gauss' theorem is applied to the surface integral, leading to

$$\int_{\mathcal{V}} \frac{\partial \rho}{\partial t}\, dV = -\int_{\mathcal{V}} \nabla(\rho \mathbf{u}) dV, \tag{2.4}$$

after which collecting the terms on the left side gives

$$\int_{\mathcal{V}} (\frac{\partial \rho}{\partial t} + \nabla(\rho \mathbf{u})) dV = 0. \tag{2.5}$$

As this result is true however big or small the control volume is, the term inside the integral must be zero. Using index notation in place of ∇ gives

$$\int_{\mathcal{V}} (\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i}) dV = 0. \tag{2.6}$$

In incompressible flow, the density is constant along fluid paths. Introducing the material derivative:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u_i \frac{\partial}{\partial x_i} \tag{2.7}$$

For the density, the material derivative becomes

$$\partial \rho / \partial t + u_i \partial \rho / \partial x_i \ . \tag{2.8}$$

This is zero due to the incompressibility, leaving

$$\frac{\partial u_i}{\partial x_i} = 0. \tag{2.9}$$

This shows that the continuity equation for incompressible flows describes a divergence free velocity field.

2.2.2 Conservation of momentum

The formulation of conservation of momentum is based on Newton's second law, stating that the change of momentum on an element equals the sum of forces acting on it. The material derivative $D(\rho \mathbf{u})/Dt$ is the momentum change, and the forces consist of body forces $\rho \mathbf{f}$ and surface forces $\mathbf{P}$. Our starting point becomes:

$$\rho \frac{D\mathbf{u}}{Dt} = \rho \mathbf{f} + \mathbf{P}. \tag{2.10}$$

The only body force considered here is gravity, **g**. Other body forces, not accounted for, include magnetic and electric fields. Our surface forces are pressure and viscous forces. To derive the momentum equation a differential element, shown in figure 2.1,is used. Only two spatial directions are considered.

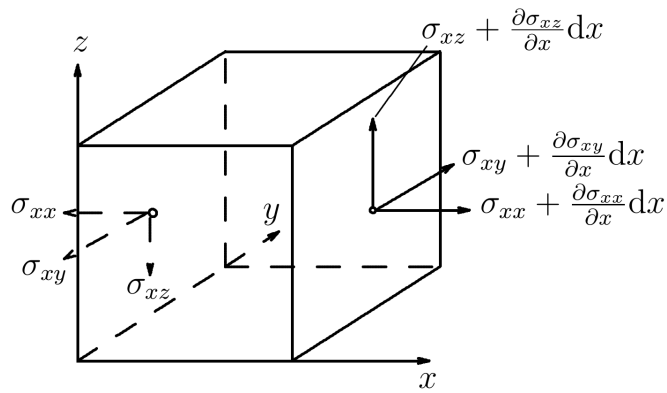

Figure 2.1: A differential element of dimension dx, dy, dz. Shown are stresses acting on the (y,z) planes.

$\boldsymbol{\sigma}$ is the total stress tensor. When written σ_{ij}, i is the normal direction of the plane, and j refers to the direction of the stress component. Tallying up all forces acting in the x-direction gives:

$$\begin{aligned} F_x &= (\sigma_{xx} + \frac{\partial \sigma_{xx}}{\partial x} dx - \sigma_{xx})\ dy + (\sigma_{yx} + \frac{\partial \sigma_{yx}}{\partial y} dy - \sigma_{yx})\ dx \\ &= (\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y})\ dxdy, \end{aligned} \tag{2.11}$$

with equivalent systems in the y and z-directions. The general expression is $\mathbf{F}/dV = \nabla\boldsymbol{\sigma}$. Splitting $\boldsymbol{\sigma}$ into pressure and viscous forces:

$$\boldsymbol{\sigma} = \boldsymbol{\tau} - \mathbf{I}p, \tag{2.12}$$

with **I** being the identity matrix. The viscous tensor is given by the assumption of the fluid being Newtonian, and reads:

$$\tau_{ij} = \mu\ (\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})\ . \tag{2.13}$$

The surface forces in 2.10 can now be written as:

$$\mathbf{P} = \frac{\mathbf{F}}{dV} = \nabla \cdot \boldsymbol{\tau} - \nabla p. \tag{2.14}$$

By combining the results of 2.10, 2.13, 2.14 and the divergence free criterion of 2.9, the momentum equation can be constructed:

$$\rho \frac{D\mathbf{u}}{Dt} = \rho \mathbf{g} - \nabla p + \mu \nabla^2 \mathbf{u}. \tag{2.15}$$

If the fluid density is constant, the gravity term can be incorporated into a modified pressure (Pope [2000]). To do this, one starts by defining a potential $\psi \equiv |\mathbf{g}|z$. This leads to the formulation:

$$\rho \mathbf{g} = -\rho \nabla \psi = -p|\mathbf{g}|\mathbf{e}_3 \tag{2.16}$$

making sure that gravity acts in the negative z-direction and letting $\mathbf{e}_3$ be a unit vector in the z-direction. Defining a modified pressure $\tilde{p}$:

$$\tilde{p} \equiv p + \rho|\mathbf{g}|z, \tag{2.17}$$

having the property

$$\nabla \tilde{p} = \nabla p + \rho \nabla \psi. \tag{2.18}$$

Substituting 2.16 and 2.18 into the momentum equation, the gravity term is eliminated:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla \tilde{p} + \mu \nabla^2 \mathbf{u}. \tag{2.19}$$

The tilde is omitted in the following, but the equation used is 2.19, with modified pressure.

Arriving finally at the complete Navier-Stokes equations, here written together for completeness:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.20}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j x_j} \tag{2.21}$$

For small domains with moderate Reynolds numbers, the Navier-Stokes equations can be solved directly with numerical methods, meaning that all the turbulent frequencies and features of the flow are resolved. When the need for accuracy is lower or the available resources are limited, one has to resort to modelling of some kind. The methods of solving the equations are usually split into three categories:

Direct numerical simulation (DNS)

In DNS, the continuity and momentum equations are solved directly. All relevant scales are resolved, and the results can be used to gain insight into flow configurations and verify other models. The downside comes with the large amounts of computing power needed; the number of grid points scale as $Re^{9/4}$, while the time steps grow as $\sqrt{Re}$ (Rogallo and Moin [1984]). As a consequence, DNS is mostly used for research purposes.

Large eddy simulation (LES)

For practical and engineering applications, LES finds more use than DNS. Here the large scales are simulated directly, while the viscous scales are modelled. The cut off between direct simulation and modelling is set so as to resolve a certain percentage of the flow energy, while the dissipation happens in the sub-grid scales. LES is computationally cheaper than DNS. It has problems with wall-bounded flows, where it needs to tend to DNS levels of resolution to capture the relevant scales. To circumvent this, a subcategory called Detached eddy simulation (DES) uses RANS close to the wall and LES away from it.

Reynolds-averaged Navier-Stokes equations (RANS)

The last category of models are under the umbrella of RANS. Here another set of equations, shown in section 2.3, are solved in place of the Navier-Stokes equations. This reduces the computational requirements at the cost of decreased accuracy. It sees wide use in engineering applications due to the body of testing the models have been through Pope [2000].

2.3 Reynolds-averaged Navier-Stokes equations

The idea behind the RANS equations is to treat the flow statistically. This starts by splitting the velocity field into an average and a fluctuating part:

$$\mathbf{u}(\mathbf{x},t) = <\mathbf{u}(\mathbf{x},t)> + \mathbf{u}'(\mathbf{x},t) \tag{2.22}$$

The important concept here is the statistical mean, denoted $<>$. It is defined as the integral over the sample space, weighted by the probability density function (Pope [2000]). As the PDF is often not known a priori, the statistical average can be approximated by ensemble averaging of N experiments:

$$<u(t)>_N = \frac{1}{N}\sum_{n=1}^{N} u_n(t) \ . \tag{2.23}$$

In the case of statistically stationary flow, one can perform a time average instead:

$$<u(t)>_T = \frac{1}{T_2 - T_1}\int_{t=T_1}^{T_2} u(t)\ dt \ . \tag{2.24}$$

If the flow is spatially homogeneous, one can use spatial averaging:

$$<u(t)>_L = \frac{1}{L^3}\iiint_{L_i} u(x_i,t)\ dx_i \ . \tag{2.25}$$

In either case the velocity profile becomes smoother, as shown in figure 2.2.

Applying the statistical average to Navier-Stokes, noting that the equations are divergence free and that $<u'> = 0$, gives the Reynolds averaged Navier-Stokes equations. The notation used here is $u_{i,j} \equiv \partial u_i/\partial x_j$.

$$<u_i>, i = 0 \tag{2.26}$$

$$\partial_t <u_i> + (<u_i><u_j>)_{,j} = -\frac{1}{\rho}<p>_{,i} + \nu <u_i>_{,jj} - \underline{<u_i' u_j'>_{,j}} \tag{2.27}$$

The act of averaging brings up a new unknown. It is underlined above, and is called Reynolds stress: $\mathbf{R} \equiv <u_i' u_j'>$. To see why it is deemed a stress, the RANS equations can be rewritten

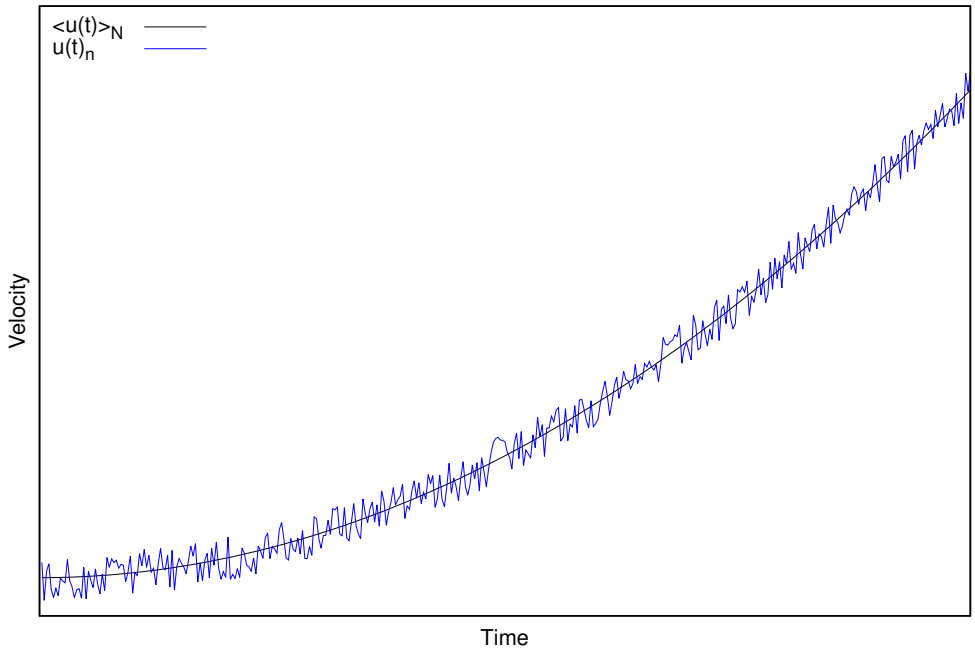

Figure 2.2: Instantaneous and ensemble averaged velocities.

like this:

$$\underbrace{\rho\partial t<u_i> + \rho(<u_i><u_j>)_{,j}}_{material\ derivative\ of\ average\ velocity} = \underbrace{(\mu<u_i>_{,j} - <p>\delta_{ij} - \rho<u_i'u_j'>)_{,j}}_{divergence\ of\ stress} \tag{2.28}$$

Using a divergence operator on the right hand side, these terms can be transformed into surface integrals by Gauss' theorem when using the integral form of the equations. Now the stress term makes sense, as they are indeed acting on the surface of the fluid element.

The closure problem

By introducing the statistical averaging, a new variable containing higher order moments of the original variables appears. This is called the closure problem, and is as consequence of the Navier-Stokes equations being non-linear. To close the equations, some level of modelling is required. As the unknown disappears in laminar flow (the fluctuations are zero), the models are called turbulence models. Which level of closure one decides to go to determines the number and complexity of the equations to be solved.

2.4 Derived quantities

In addition to the four unknowns in the Navier-Stokes equations (three velocity components and pressure), other variables and equations can be derived through algebra.

2.4.1 The kinetic energy equation

Kinetic energy is defined as:

$$E_k = \frac{1}{2} u_i u_i \tag{2.29}$$

To derive a transport equation for kinetic energy, the operator $\boldsymbol{u} \cdot ()$ is applied to the Navier-Stokes equations. All the derivations can be found in chapter five of Pope [2000]. The resulting equation reads:

$$\frac{\partial E_k}{\partial t} + u_j \frac{\partial E_k}{\partial x_j} + \frac{1}{\rho} \frac{\partial u_i p}{\partial x_i} = 2\nu \frac{\partial u_i Sij}{\partial x_j} - 2\nu S_{ij} S_{ij} \tag{2.30}$$

Another term appearing here is the rate of strain tensor $\mathbf{S}$:

$$S_{ij} \equiv \frac{1}{2} (\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}). \tag{2.31}$$

The RANS kinetic energy equations

By looking at the individual contributions of the mean and fluctuating field, the mean and turbulent kinetic energy equations can be constructed.

$$k \equiv \frac{1}{2} <u_i' u_i'> \tag{2.32}$$

is the turbulent kinetic energy, and is half the trace of $\mathbf{R}$.

$$\overline{E} \equiv \frac{1}{2} <u_i u_i> \tag{2.33}$$

is the kinetic energy of the mean field. The transport equation for k:

$$\frac{\partial k}{\partial t} + u_j \frac{\partial k}{\partial x_j} + \frac{\partial}{\partial x_j} (\frac{1}{2} <u_i' u_j' u_j'> + \frac{<u_i' p>}{\rho}) = \nu \nabla^2 k + \mathcal{P} - \widetilde{\varepsilon} \,. \tag{2.34}$$

And for $\overline{E}$:

$$\frac{\partial \overline{E}}{\partial t} + \frac{\partial}{\partial x_j}(<u_j>\overline{E_k} + <u_j><u_i u_j> + \frac{<u_j><p>}{\rho} - 2\nu<u_i>\overline{S}_{ij}) = -\mathcal{P} - \overline{\varepsilon} \ . \tag{2.35}$$

The new terms here are the production

$$\mathcal{P} \equiv -<u_i' u_j'>\frac{\partial <u_i>}{\partial x_j}, \tag{2.36}$$

the mean dissipation rate

$$\overline{\varepsilon} \equiv 2\nu \overline{S}_{ij}\overline{S}_{ij} \tag{2.37}$$

and the pseudo-dissipation rate

$$\widetilde{\varepsilon} \equiv \nu <\frac{\partial u_i'}{\partial x_j}\frac{\partial u_i'}{\partial x_j}>. \tag{2.38}$$

The latter is in most cases virtually equal to the dissipation rate, but simplifies the turbulent kinetic energy equation a lot. Note here that the production term appears with opposite signs in the two equations. As $\mathcal{P}$ is almost always positive, it transports energy from the mean field to the the fluctuating one. The dissipation term is a sink term in both equations, being definite positive and appearing with a negative sign.

2.4.2 Reynolds stress transport equations

One can also derive transport equations for the Reynolds stresses by starting with Navier-Stokes equations for the instantaneous field and the RANS equations, and subtracting one from the other. This is then multiplied by u_j, averaged and added to the result of doing the same with u_i. The end result looks like this (Durbin and Reif [2011]):

$$\frac{\overline{D}}{Dt}(\overline{u_i' u_j'}) + \frac{\partial}{\partial x_k}\mathcal{T}_{ijk} = \mathcal{P}_{ij} + \mathcal{R}_{ij} - \varepsilon_{ij}. \tag{2.39}$$

There are three new terms here. The turbulent transport:

$$\mathcal{T}_{ijk} \equiv \overline{u_i' u_j' u_k'} + \overline{p' u_j'}\delta_{ik} + \overline{p' u_i'}\delta_{jk} - \nu\frac{\partial \overline{u_i' u_j'}}{\partial x_k}. \tag{2.40}$$

The production tensor, of which half the trace ($i = j$) is equal to 2.36:

$$\mathcal{P}_{ij} \equiv -\overline{u_j' u_k}\frac{\partial \overline{u_i}}{\partial x_k} - \overline{u_i' u_k'}\frac{\partial \overline{u_j}}{\partial x_k}. \tag{2.41}$$

A very important term, the pressure-strain tensor:

$$\mathcal{R}_{ij} \equiv \frac{p'}{\rho}(\overline{\frac{\partial u'_j}{\partial x_i}} + \overline{\frac{\partial u'_i}{\partial x_j}}) \tag{2.42}$$

Lastly, the dissipation tensor, again with the property that half the trace equals the pseudo-dissipation rate 2.38:

$$\varepsilon_{ij} \equiv 2\nu \overline{\frac{\partial u'_i}{\partial x_k}\frac{\partial u'_j}{\partial x_k}} \tag{2.43}$$

Half of the trace of 2.39 is the turbulent kinetic energy equation, 2.34.

Head loss

Because a vector field can be hard to read, a bulk description of the flow energy is often used. The change in velocity height, quantified by the Bernoulli equation, over a stretch is referred to as head loss.

$$(\frac{U_1^2 - U_2^2}{2g}) + (z_1 - z_2) + (\frac{p_1 - p_2}{\rho}) = Head\ Loss_{1-2} \tag{2.44}$$

2.5 Wall effects

At a wall, the no-slip condition gives zero velocity. In the near field, viscosity plays a large role due to the lower local Reynolds number. As a result, the velocity profile looks very different than in the free stream (Pope [2000]). For the inner part of attached boundary layers in turbulent flows, there are characteristic zones based on the length scale wall units:

$$y^+ \equiv \frac{u^* y}{\nu}, \tag{2.45}$$

with

$$u^* \equiv \sqrt{\tau_w/\rho}. \tag{2.46}$$

The velocity scale reads:

$$u^+ \equiv \frac{<u>}{u^*}. \tag{2.47}$$

Figure 2.3 shows a velocity profile obtained through DNS by John Kim and Moser [1987], compared with the stipulated wall laws of Prandtl [1938] and Von Kármán [1930].

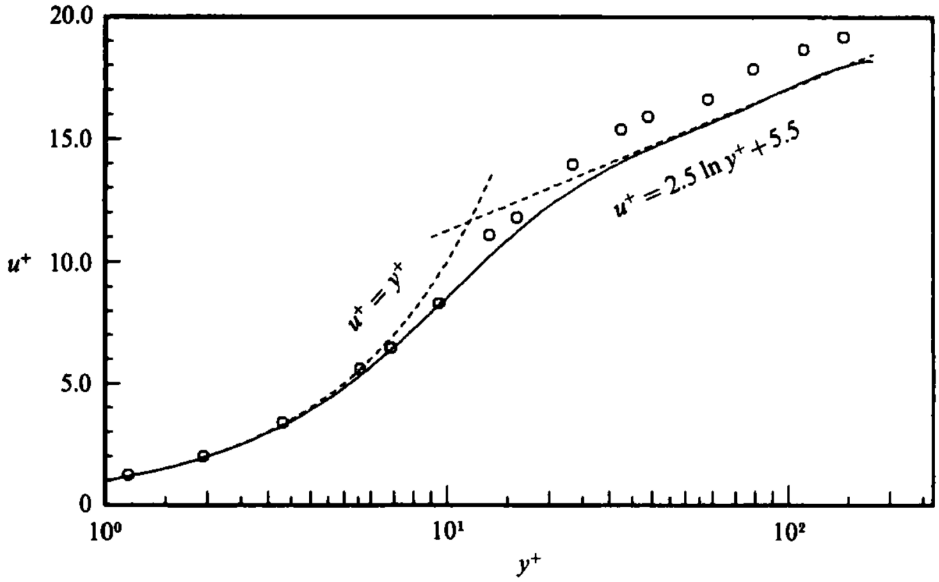

Figure 2.3: The velocity profile close to the wall. Solid line is DNS data from John Kim and Moser [1987], dotted line is the wall law.

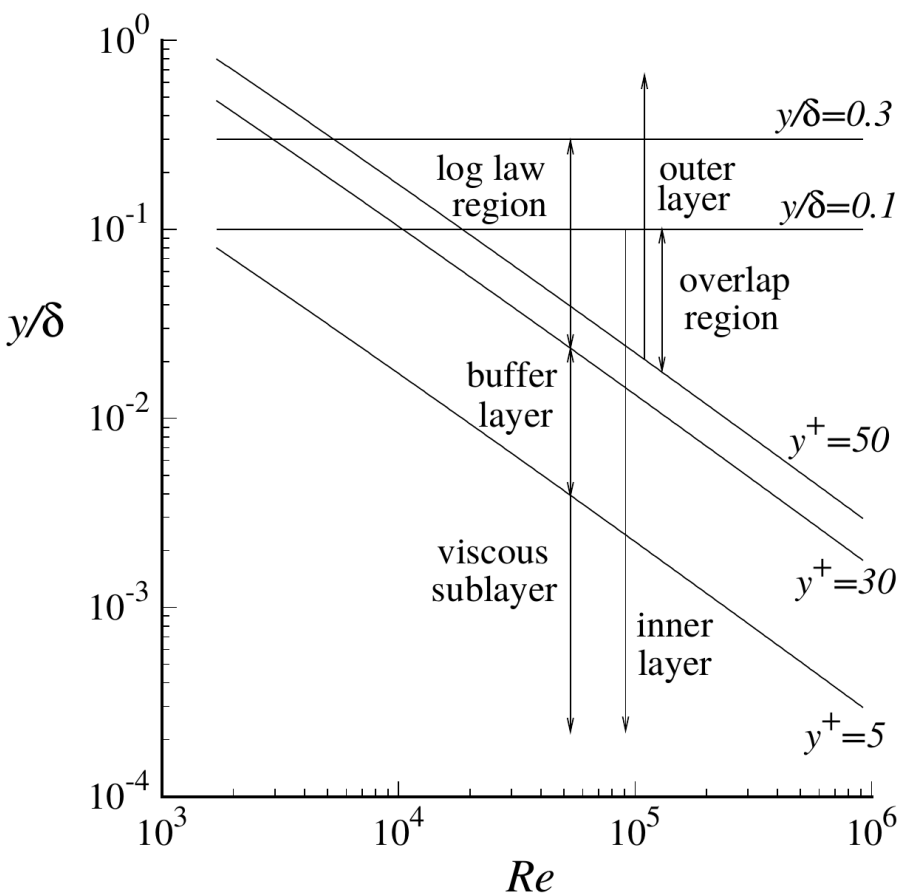

Figure 2.4: The distinct zones close to the wall as defined by Pope [2000].

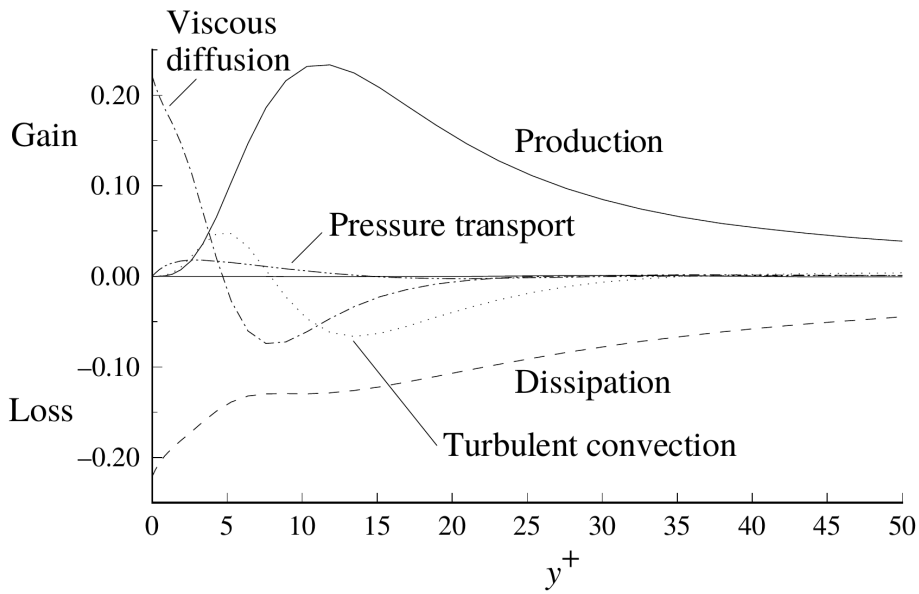

Figure 2.5: Budget of turbulent kinetic energy from John Kim and Moser [1987]. Figure taken from Pope [2000].

As the velocity profile changes close to the wall, so does the terms of the turbulent kinetic energy equation 2.34. Figure 2.5 shows the energy budget. Production, viscous diffusion and dissipation all finds their peak values at $y^+<20$. This makes modelling this part of the flow challenging in terms of resolution, discussed further in section 2.6.2.

2.6 Turbulence modelling with RANS

Almost all engineering flows are turbulent (Pope [2000]), characterized by a high Reynolds number, $Re \equiv \frac{UD}{\nu}$. Turbulent flows are distinguished by a large separation of scales, irregularity, complex vortical motion, larger dissipation of energy and increased transport of momentum, mass and heat. The first characteristic can prove problematic, seeing that the ratio of the largest to the smallest scales grow as $Re^{3/4}$ (Kolmogorov [1941]). Turbulence models remove the need to model these scales in exchange for a lower accuracy.

A family of turbulence models which sees wide use in practical applications is collectively known as Reynolds Averaged Navier-Stokes or RANS models. They are based on equations 2.26 and 2.27. To make a closed system of equations, the Reynolds stress needs to be supplied, and the models differ in how they do this.

2.6.1 First order closure

The first family of models are those that use some scalar quantity to connect the Reynolds stress to the other, known quantities of the flow. This is called first order closure.

One subsection of first order models use turbulent viscosity to model the Reynolds stress. They are collectively known as turbulent viscosity models, and are based on the Boussinesq hypothesis (Boussinesq [1877]). The analogy made is based on the stress/strain relationship for a Newtonian fluid:

$$\sigma_{ij} = \rho\nu(u_{i,j} + u_{j,i}) - \delta_{ij}p, \tag{2.48}$$

with $\rho\nu$ acting as a proportionality constant. Extrapolating this to the turbulent stress gives:

$$<u_i'u_j'> = \frac{2}{3}k\delta_{ij} - 2\nu_T\overline{S}_{ij} \tag{2.49}$$

or, with the Reynolds stress anisotropy defined as $a_{ij} \equiv <u_i'u_j'> - \frac{2}{3}k\delta_{ij}$:

$$a_{ij} = -2\nu_T\overline{S}_{ij}. \tag{2.50}$$

The new variable here is ν_T, turbulent viscosity. By supplying this, the system is closed. The Boussinesq hypothesis is built on two assumptions. The first one is that the anisotropy of the Reynolds stress scales with the gradients of the mean flow, and the second is that the relationship is the one written above. A further discussion of their pros and cons are made in section A.1 in the appendix.

The k-epsilon model

The most popular of the turbulent viscosity models is the k-epsilon model. Here, the closure of the turbulent viscosity is done by way of turbulent kinetic energy and dissipation rate. The construction of the relationship is dimensional: kinematic viscosity has dimension m^2/s, kinetic energy has dimensions m^2/s^2, and dissipation has dimensions m^2/s^3. The formula then has to be on the form:

$$\nu_T = C_\mu k^2/\epsilon \tag{2.51}$$

with C_μ being constant. Given the fields of k and ϵ, the equations are closed. The transport equation for k, 2.34, can be rewritten to have one unclosed term by introducing

$\mathbf{T}'$, where $T_i = \frac{1}{2}<u_i u_j u_j> + <u_i p>/\rho - 2\nu<u_i s_{ij}>$. The full equation then reads:

$$\frac{\overline{D}k}{\overline{D}t} = -\nabla \cdot \mathbf{T}' + \mathcal{P} - \varepsilon, \tag{2.52}$$

and is closed with the specification of $\mathbf{T}'$. This is modelled with a gradient-diffusion hypothesis:

$$\mathbf{T}' = -\frac{\nu_t}{\sigma_k}\nabla k, \tag{2.53}$$

and the final equation is

$$\frac{\overline{D}k}{\overline{D}t} = \nabla \cdot (\frac{\nu_t}{\sigma_k}\nabla k) + \mathcal{P} - \varepsilon. \tag{2.54}$$

The constant, σ_k, is one of five total in the standard k-epsilon model, and is set to unity. For ϵ, the original transport equation is not used at all. Instead, a fully empirical model equation, based on that for k, is employed:

$$\frac{\overline{D}\varepsilon}{\overline{D}t} = -\nabla \cdot (\frac{\nu_t}{\sigma_\epsilon}\nabla \epsilon) + C_{\epsilon 1}\frac{\mathcal{P}\epsilon}{k} - C_{\epsilon 2}\frac{\epsilon^2}{k}. \tag{2.55}$$

The production and dissipation terms are transformed using ε/k times a constant. The full list of constants in the standard model is:

$$C_\mu = 0.09 \qquad C_{\varepsilon 1} = 1.44 \qquad C_{\epsilon 2} = 1.92$$
$$\sigma_k = 1.0 \qquad \sigma_\epsilon = 1.3$$

2.6.2 Wall modelling

The k-epsilon model is well tested in- and outside of its domain of applicability. Problems arise close to the wall, as the tuning of the model coefficients is done at high Reynolds numbers. As the local Reynolds number decreases, so does the accuracy of the model. The steep gradients of velocity and turbulent quantities normal to the wall means that a very fine grid needs to be employed here in order to capture the gradients adequately (Pope [2000]). A common estimate is 10 grid cells in the viscous region ($y^+ \leq 5$), as given by Nikitin et al. [2000]. It can be problematic if one chooses to make the cells too flat in an attempt to decrease the total amount, as a high aspect ratio can degrade the results or lead to instability (Olsen [2012]). There is also a danger of oscillations in the solution, giving negative k and ε values (Ferziger and Peric [2012]).

If the flow is somewhat parallel to the wall, wall functions can be applied. Boundary

conditions are applied at a certain distance from the wall, inside the log layer ($y^+ \approx 50$). The following relations are taken from Pope [2000], and are also found in OpenFOAMs implementation, summed up by Liu [2017]. The standard log-law reads:

$$<u> = u^*(\frac{1}{\kappa}\ ln\ y^+ + B), \tag{2.56}$$

with the von Karman constant $\kappa = 0.41$ and $B = 5.5$ (Pope [2000]). Assuming a balance of production and dissipation gives

$$\varepsilon = \frac{u^{*3}}{\kappa y}, \tag{2.57}$$

and the turbulent viscosity formulation given in section 2.6.1 allows writing:

$$-<u'v'> = u^{*2} = C_\mu^{1/2} k\ . \tag{2.58}$$

For the first grid point, where the boundary conditions are applied, the subscript y_g is used. This allows the definition of

$$u^*_{nominal} \equiv C_\mu^{1/4} k_g^{1/2}, \tag{2.59}$$

with a corresponding estimate of y_g^+:

$$y_g^+ \equiv \frac{y_g u^*_{nominal}}{\nu}. \tag{2.60}$$

The nominal mean velocity is given by the log-law:

$$<u>_{g\ nominal} = u^*_{nominal}(\frac{1}{\kappa}\ ln\ y_g^+ + B)\ . \tag{2.61}$$

To make the solution more robust, the shear stress is used as a boundary condition in the mean momentum equation:

$$-<u'v'>_g = u^{*2}_{nominal} \frac{<u>_g}{<u>_{g\ nominal}}\ . \tag{2.62}$$

This acts as a restoring force if $<u>_g$ exceeds $<u>_{g\ nominal}$. For k and the normal stresses, a zero-normal gradient boundary condition is used, while epsilon gets evaluated based on equation 2.57:

$$\varepsilon_g = \frac{u^{*3}_{nominal}}{\kappa y_g}\ . \tag{2.63}$$

Wall functions can reduce the computational requirements substantially, as they lower the number of cells necessary close to the wall. There are however cases where their accuracy is poor; separation zones and flows with strong adverse pressure gradients being two examples

(Wilcox et al. [1993]).

2.7 Flow development

A flow is said to be developed when its mean statistics do not change in the streamwise direction. The quantity to be investigated in this thesis is head loss, and its value between equally spaced measurements along the tested pipes. Different estimates for the length needed to be developed are found in the literature; Zagarola and Smits [1998] uses the result of Dean and Bradshaw [1976] to come up with their formulas. Splitting the development into a laminar to turbulent transition, a boundary layer development length and a large-eddy development length, they find different Reynolds number dependencies for the three. The large-eddy development has a minute impact on head loss, as the secondary currents generated by the strips will overshadow them by an order of magnitude or more in terms of momentum transfer to the walls. Combined with the strips acting as tripping devices for turbulence, the length to be used for an estimated length to development is the boundary layer development length. This has an inverse scaling with the skin friction (Zagarola and Smits [1998]); $L/D \approx C/\lambda$, with λ being skin friction, D is channel height and C a constant. Seeing as skin friction decreases with increasing Reynolds numbers, one will expect L to increase with the Reynolds number in a smooth pipe. Again, the strips will interfere with this in some a priori unknown way. A result giving an upper limit for what might be expected is $L = 30D$ at $Re = 10^5$ from Dean and Bradshaw [1976].

2.8 Secondary currents

Secondary currents are flows in straight channels that do not align with the stream direction. They are split into two categories, depending on their origin; Prandtl's first and second kind, after Ludwig Prandtl. Prandtl [1952] contains the theoretical basis for this section.

The mechanism behind secondary currents of Prandtl's first kind is the centrifugal force exerted as a result of mean curvature of the flow. The force balance between the centrifugal acceleration and the vertical pressure gradient leads to circulation in the plane.

The second kind is generated by the turbulence. Citing Nezu et al. [1993], the foremost driving factor is anisotropy between $\sqrt{v'v'}$ and $\sqrt{w'w'}$. These are the fluctuations in y- and z-directions respectively, in a system where the main flow direction is x. Looking at

the production term from the kinetic energy transport equation, equation 2.36, and setting $i = 2$ and $j = 3$ or vice versa explains why. An increase in anisotropy comes with increased production of turbulence at the expense of the kinetic energy of the mean field.

Figure 2.6 shows the mechanisms behind the generation of secondary currents of Prandtl's second kind, as well as their interconnections. Most relevant to the results of this thesis is the increase of momentum and energy transfer that comes with secondary currents, regardless of their origin. This leads to increased head loss in an otherwise smooth pipe, as energy is transferred from driving the flow forward to generating turbulence.

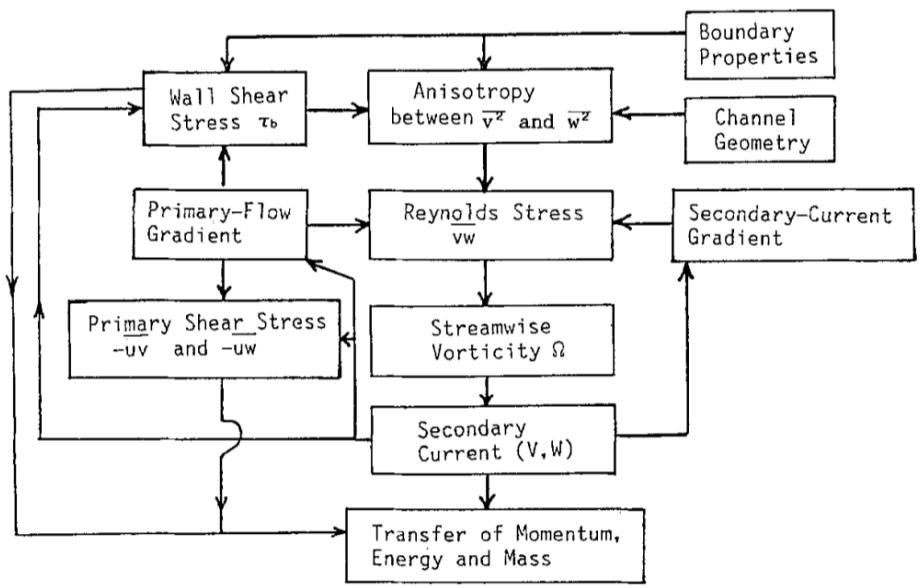

Figure 2.6: Mechanisms generating secondary currents of Prandtl's second kind, taken from Nezu et al. [1993].

Chapter 3

Numerical methods

3.1 Computational domain

To handle a domain of interest numerically, it needs to be discretized. To discretize an equation encompasses turning something continuous into something discrete. For the Navier-Stokes equations, the differential equations are turned into algebraic ones, while a discrete grid is created to represent the originally continuous solution space.

3.1.1 The finite volume approach

The finite volume method is one of the two main ways of converting partial differential equations into algebraic ones, the other one being finite differences. While the finite difference method is based on replacing derivatives directly with differences, akin to the definition of the derivative, the finite volume method is based on an idea of using the integral form of the governing equations. The computational domain gets split up into finite volumes, also called grid cells. The variables are computed as mean values inside of each cell, and these values are the unknowns. The equations are discretized from their integral form, enforced by flux balances over the cell faces. The latter fulfills the conservation properties of the governing equations by design, given that three conditions are met (taken from Uhlmann [2012]):

- **Coverage**: The whole domain needs to be covered by the grid cells.
- **Overlap**: If cells overlap, each cell face must appear an even amount of times to ensure conservation, as fluxes going out of one cell need to appear in another.

- **Flux uniqueness**: The fluxes should be computed on cell faces, not for cells as a whole, and then at the end of the loop be added to their cells. This ensures that the same volume of water is not being accounted for multiple times in one loop through the domain.

Due to its flexibility in terms of grid arrangements as well as guaranteed conservation, the finite volume approach sees wide use in CFD programs today, among them OpenFOAM.

3.1.2 Grids and truncation error

As mentioned above, the continuous computational domain is covered by discretely sized grid cells. The difference between the real result and the one obtained from a discretization is named truncation error. The size of this error is decided by the grid point spacing together with the magnitude of the gradients and choice of numerical scheme. This is covered in section 3.2.

3.1.3 Boundary conditions

For equations that are not strictly hyperbolic, where only incoming information needs to be specified (Hirsch [2007]), initial and boundary conditions are needed. Some are given by the physics of the problem, an example being the no-slip condition on the wall leading to zero velocity here. Others need to be decided, and making practical as well as physically sound choices here can make or break a calculation. Boundary conditions can in general be put into three categories:

- **Dirichlet boundary conditions**: The variable takes a set value at the boundary.
- **von Neumann boundary conditions**: The gradient of the variable takes a set value at the boundary.
- **Mixed type boundary conditions**: Some combination of the two above.

3.2 Analyzing numerical schemes

To solve the original partial differential equations on a discrete grid, a numerical scheme is needed. In order to ensure that the solution is sound, the scheme needs to have three properties (Hirsch [2007]);

- **Consistency**: The algebraic equation should go to its original PDE when grid spacing and time step goes to zero. In other words, the limit of the truncation error should be zero in the aforementioned case.
- **Convergence**: The solution of the algebraic equation should tend to the solution of its original PDE when grid spacing and time steps goes to zero. Note that a consistent scheme may not be convergent if it is not stable.
- **Stability**: Small errors introduced by machine precision or round-off error should not grow without bounds.

3.2.1 Consistency

To prove consistency means showing that the truncation error goes to zero, which starts by finding it. An example of a truncation error, adopted from Uhlmann [2012], is found by studying the second order approximation of a first derivative, $\frac{\partial u}{\partial x}$. The goal is to approximate this derivative using the discrete values on an equally spaced grid. Here u_i, u_{i-1} and u_{i-2} are used, i being the index of the grid point where the derivative is to be evaluated. At the end, the equation should have this form:

$$(u_{,x})_i = \frac{au_i + bu_{i-1} + cu_{i-2}}{\Delta x} + O(\Delta x^2), \tag{3.1}$$

achieving a second order truncation error. Starting by performing Taylor expansions around a grid point:

$$u_i = u_i \tag{3.2}$$

$$u_{i-1} = u_i - \Delta x(u_{,x})_i + \frac{\Delta x^2}{2}(u_{,xx})_i - \frac{\Delta x^3}{6}(u_{,xxx})_i + ... \tag{3.3}$$

$$u_{i-2} = u_i - 2\Delta x(u_{,x})_i + \frac{(2\Delta x)^2}{2}(u_{,xx})_i - \frac{(2\Delta x)^3}{6}(u_{,xxx})_i + ... \tag{3.4}$$

Now, the equations can be tallied up:

$$\frac{au_i + bu_{i-1} + cu_{i-2}}{\Delta x} = \frac{u_i}{\Delta x}(a+b+c) - (b+2c)(u_{,x})_i + \frac{\Delta x}{2}(b+4c)(u_{,xx})_i + O(\Delta x^2) \quad (3.5)$$

The conditions that have to be met are:

- $a + b + c = 0$, to avoid the error increasing as $\frac{1}{\Delta x}$;
- $b + 2c = -1$, to have the wanted derivative with a factor of one;
- $b + 4c = 0$, to get rid of the first order error term.

This gives the factors $a = 3/2$, $b = -2$ and $c = 1/2$. Finally, equation 3.1 with the correct weights:

$$(u_{,x})_i = \frac{3u_i - 2u_{i-1} + u_{i-2}}{2\Delta x} + \epsilon \quad (3.6)$$

The leading error term is the first term that gets omitted from the approximation:

$$\epsilon = -\frac{\Delta x^2}{6}(b+8c)(u_{,xxx})_i + O(\Delta x^3) = -\frac{\Delta x^2}{3}(u_{,xxx})_i + O(\Delta x^3) \quad (3.7)$$

Here it is a second order term, meaning the approximation is in fact of second order. Another observation is that the terms vanish with decreasing Δx, and the scheme is thus proven to be consistent.

3.2.2 Stability

Analyzing the stability can be done in several ways. Many methods are restricted to linear problems, and almost all of them run into problems when complex schemes and boundary conditions are being investigated. One of these, named the Equivalent Differential Equation method, is used underneath in section 3.3.1. By instead turning the problem into one with periodic boundary conditions, a von Neumann analysis can be performed. A von Neumann analysis is one of the most general ways of proving the stability of a scheme. Note that the boundaries need separate treatment here. The key trick to a von Neumann stability analysis is turning the solution into a Fourier series in the spatial frequency domain. To do this, the domain is mapped onto the length $(-L, 0)$, thus letting itself being mapped onto a finite Fourier series in the domain $(-L, L)$. The method then goes like this (Hirsch [2007]):

1. Replace all the terms of the form u_{i+m}^{n+k} with $u_{i+m}^{n+k} \Rightarrow V^{n+k} e^{I(i+m)\theta}$.

2. Simplify, dividing all the terms by $e^{Ii\theta}$.
3. From the resulting relation, extract an error amplification factor $G \equiv \frac{V^{n+1}}{V^n}$.
4. Check the criterion $|G| \leq 1$ for all angles $-\pi \leq \theta \geq \pi$.

Step number four ensures that introduced errors do not grow without bounds using the scheme in question. An example of this method being used is found in Konangi and Palakurthi [2016], where the stability of the SIMPLE algorithm is being analyzed.

3.2.3 Convergence

Proving convergence is hard. Often, the solution to the original PDE is not known beforehand. Instead, a theorem is used:

> Given a properly posed initial value problem (A), and a finite difference approximation to it (B) that satisfies the consistency condition, stability is a necessary and sufficient condition that B is a convergent approximation.
>
> Lax and Richtmyer [1956]

This allows taking convergence for granted if the scheme used is proven to be stable and consistent.

3.3 Other traits of numerical schemes

Besides the three properties from section 3.2, there are other important phenomenon one needs to have an idea about when dealing with numerical schemes.

3.3.1 Numerical diffusion

Turbulent flow is convection dominated, as the Reynolds number, being high, can be interpreted as the ratio of convective to diffusive motion. A lot of insight into the behaviour of numerical solutions to the Navier-Stokes equations can therefore be gained by ignoring the

diffusion, and study a simple convection equation:

$$u_{,t} + au_{,x} = 0 \qquad \text{with } a \text{ constant.} \tag{3.8}$$

An explicit, forward difference in time (using time nodes n and $n+1$) with a first order upwind scheme in space (using space nodes i and $i-1$) is employed. The scheme reads:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{a}{\Delta x}(u_i^n - u_{i-1}^n) = \epsilon \qquad \text{with } \epsilon \text{ being the truncation error.} \tag{3.9}$$

Taylor expansions around u_i^n are used to find replacements for the other grid points:

$$u_i^{n+1} = u_i^n + \Delta t(u_{,t})_i^n + \frac{\Delta t^2}{2}(u_{,tt})_i^n + \frac{\Delta t^3}{6}(u_{,ttt})_i^n + ...$$

$$u_{i-1}^n = u_i^n - \Delta x(u_{,x})_i^n + \frac{\Delta x^2}{2}(u_{,xx})_i^n - \frac{\Delta x^3}{6}(u_{,xxx})_i^n + ... \tag{3.10}$$

Inserting the Taylor expansions into 3.9:

$$\{u_{,t} + au, x\}_n^i + \frac{\Delta t}{2}(u_{,tt})_i^n + \mathcal{O}(\Delta t^2, \Delta x^2) = 0 \tag{3.11}$$

The term in curly brackets is the original equation (3.8), while the rest of the left hand side is the resulting truncation error ϵ.

Notes to be made here are that the scheme is consistent, with the error going to zero with Δt and Δx, and that the truncation error is of first order in time and space. The next step is to replace all higher order time derivatives by space derivatives. To achieve this, equation 3.8 is differentiated with respect to time, giving

$$u_{,tt} + a(u_{,x})_{,t} = 0$$

$$u_{,tt} = -a(u_{,t})_{,x} \tag{3.12}$$

where the last change can be done due to the interchangeability of time and space derivatives. Using the original equation 3.8 to replace $u_{,t}$ gives

$$u_{,tt} = a^2 u_{,xx}. \tag{3.13}$$

This is then entered into 3.11 to yield

$$\{u_{,t} + au, x\}_n^i = \frac{a\Delta x}{2}(1 - \frac{a\Delta t}{\Delta x})(u_{,xx})_i^n + \mathcal{O}(\Delta t^2, \Delta x^2). \tag{3.14}$$

The leading error term is now a second derivative in space; to the leading error term, a convection-diffusion equation is solved in place of the original convection equation. This leads to numerical diffusion, smearing out the convected quantity. The term $\frac{a\Delta x}{2}(1 - \frac{a\Delta t}{\Delta x})$ is often coined numerical viscosity.

Another note about this new term introduced by the discretization, is that it puts an important restraint on the grid spacing and time step; if the numerical viscosity is negative, the error will grow without bounds. A necessary condition for stability, and thereby convergence, is therefore

$$\frac{a\Delta t}{\Delta x} \leq 1, \tag{3.15}$$

known as the Courant–Friedrichs–Lewy condition or CFL number.

3.3.2 Boundedness

Some values are bounded by physical laws. Kinetic energy k is for example always positive. Any numerical scheme should optimally guarantee boundedness of such variables. It is difficult in practice; only some first order schemes (see 3.4.2) holds up to this wish. Higher order schemes can produce unbounded results, especially if steep gradients are not properly captured by the grid (Ferziger and Peric [2012]).

Use of limiters

To improve the boundedness of higher order schemes, flux limiters can be used. OpenFOAM use the Total Variation Diminishing method, introduced by Harten [1983], to enable switching between higher and lower order schemes at discontinuities. By using a lower order scheme with more dissipation close to large gradients, the simulation is kept stable at the cost of accuracy.

3.3.3 Iterative methods

Fluid mechanics problems are usually represented by large matrices. These can always be solved by Gauss elimination, but this method is extremely computationally expensive for most matrices of the size seen in CFD. Instead, iterative methods are usually employed. A solution is guessed, and an equation is used to improve it. Given a cheap method and a small total number of iterations needed, this method is often less costly than a direct solution

(Ferziger and Peric [2012]). Starting with a system

$$A\phi = \mathbf{Q} \ , \tag{3.16}$$

we arrive at a solution ϕ^n after n iterations. This is not an exact solution, but retains a residual

$$A\phi^n = \mathbf{Q} - \rho^n. \tag{3.17}$$

The iteration error is defined as

$$\epsilon^n \equiv \phi - \phi^n, \tag{3.18}$$

and the residual is thus

$$A\epsilon^n = \rho^n. \tag{3.19}$$

The goal of any iterative method is to drive this residual down, and with it the iteration error.

3.3.4 Accuracy

All numerical solutions are only accurate to a certain extent. Common error sources are (Casey and Wintergerste [2000]):

- Modelling errors, introduced by the equations themselves not perfectly picturing the actual physics.
- Numerical errors, also known as discretization errors. Discussed in section 3.2.
- Convergence errors or iteration errors. Discussed in section 3.2 and 3.3.3.
- Round-off errors, due to machine precision being finite.
- Boundary- and initial condition errors.
- Human errors.
- Bugs in the software.

3.4 Schemes used in this thesis

3.4.1 SIMPLE

SIMPLE, or Semi Implicit Method for Pressure Linked Equations, is a classical method for solving a problem with the Navier-Stokes equations: for incompressible flow, the pressure and velocity are tightly linked. There are however no explicit way to compute the pressure field from the momentum equation (Moukalled et al. [2016]). A solution to this was proposed by Patankar and Spalding [1983], and is now widely used in commercial CFD codes.

The method employed by Patankar and Spalding [1983] for solving the momentum and continuity equations is known as a segregated approach. The solution, where both the pressure and velocity field should satisfy both equations, is found by letting each of the fields satisfy them separately and then correcting them iteratively, using a pressure correction equation. The intermediate solutions might violate mass or energy conservation, as only one equation is enforced at a time. The procedure is summarized as following by Moukalled et al. [2016]:

1. Guess a velocity field, u^n, and a pressure field, p^n.
2. Solve the momentum equation for the new velocity field, u_f^*.
3. Update the mass flow rates in line with u_f^* to get a mass flow field, m_f^*.
4. Use m_f^* to get the pressure correction field, p', through the pressure correction equation.
5. Update the pressure and velocity fields to satisfy the continuity equation using the pressure correction field.
6. Set u^n and p^n to equal the updated fields.
7. Return to step two and repeat until convergence.

Stability

Konangi and Palakurthi [2016] did an extended von Neumann stability analysis of the scheme, verified by numerical trials. The stability region for a 1D case is shown in figure 3.1. For the range of flow velocities that occur in fluid dynamics, the SIMPLE algorithm is stable. This is the far left of the graph, at very low Mach numbers.

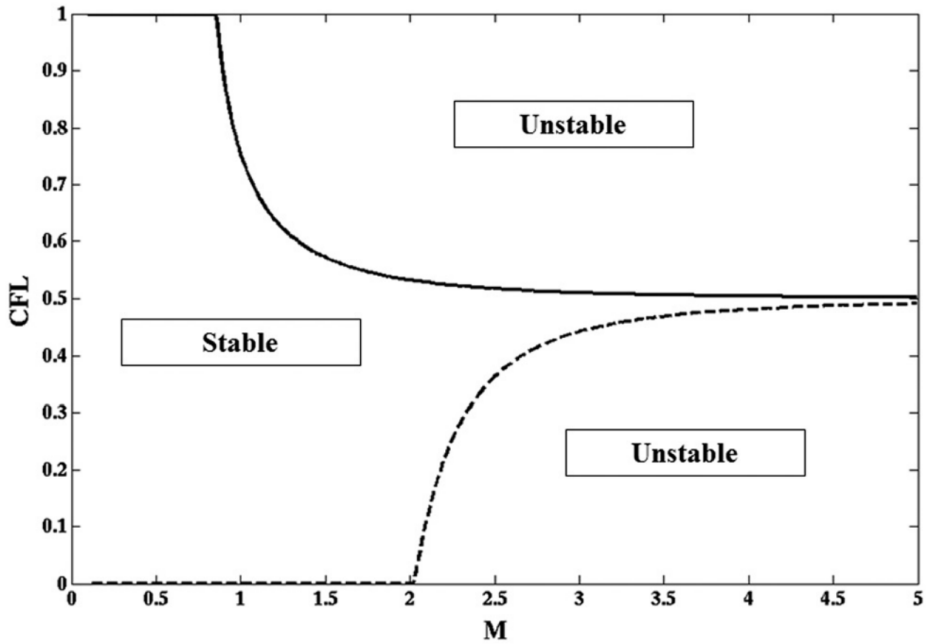

Figure 3.1: The stability region for a 1D SIMPLE scheme. Shown are CFL and Mach numbers. Konangi and Palakurthi [2016].

3.4.2 Upwind

Going by the name of **Gauss upwind**, the upwind scheme is a first order scheme. A glance into the OpenFOAM tutorials using foamSearch reveals that it is the most popular scheme for discretizing k and epsilon. This is due to the focus being on boundedness rather than the accuracy of these variables.

A thorough testing of the accuracy of this scheme, as well as the ones below, is available at Holzmann [2018]. He used OpenFOAM and several grids, differing in non-orthogonality and structure, to look at how the schemes compared to a analyitcal solution of a pure convection equation. For the upwind scheme, he comments that the scheme is very sensitive to flows at an angle to the grid, when the diffusive character of the scheme becomes overwhelming. The upwind scheme is used for k and epsilon in the simulations.

Stability

The stability of this scheme is discussed in section 3.3.1, with the resulting stability limit set by the Courant number; $Co < 1$.

3.4.3 Linear

The linear scheme is a second order scheme, named **Gauss linear** in OpenFOAM. It is known for being unstable, and the limited version **Gauss limitedLinear** is normally used. Holzmann [2018] comments that it leads to errors growing without bounds if left unchecked on a nonorthogonal grid. The limited linear scheme is used for ν_t in the simulations.

Stability

The stability of the central, second order scheme that the linear scheme is based on can be checked by way of von Neumann analysis.

Using grid points i, $i-1$ and $i+1$ on a uniform grid with an explicit forward difference in time, the scheme looks like this:

$$u_i^{n+1} = u_i^n - \frac{\sigma}{2}(u_{i+1}^n - u_{i-1}^n), \quad \text{defining} \quad \sigma \equiv \Delta ta/\Delta x. \tag{3.20}$$

Adding the Fourier modes as described in section 3.2.2 yields:

$$\hat{u}^{n+1}e^{Ii\theta} = \hat{u}^n e^{Ii\theta} - \frac{\sigma}{2}(\hat{u}^n e^{I(i+1)\theta} - \hat{u}^n e^{I(i-1)\theta}). \tag{3.21}$$

Dividing by $e^{Ii\theta}$:

$$\hat{u}^{n+1} = \hat{u}^n(1 - \frac{\sigma}{2}(e^{I\theta} - e^{-I\theta})). \tag{3.22}$$

Using the fact that $e^{Ia} = cos(a) + Isin(a)$ and that cosine and sine are even and odd, respectively, G becomes:

$$G = \frac{\hat{u}^{n+1}}{\hat{u}^n} = 1 - \sigma\ Isin(\theta) \quad \text{with} \quad 0 \leq \theta \leq \pi. \tag{3.23}$$

The condition for stability is $|G| \leq 1$. To make matters easier, $|G|^2 = GG^*$ is computed, with G^* being the complex conjugate.

$$GG^* = (1 - \sigma\ Isin(\theta))(1 + \sigma\ Isin(\theta)) = 1 + \sigma^2 sin^2\theta \geq 1. \tag{3.24}$$

The scheme is thus unconditionally unstable.

3.4.4 Linear Upwind

Referenced as **Gauss linearUpwind** in the *fvSchemes* file (see 4.4), the linear upwind scheme is a second order scheme. It uses upwind interpolation weights, and uses an explicit correction based on the local cell gradient. The scheme is described in Warming and Beam [1976], which is the base of OpenFOAMs implementation. They refer to it as the MU scheme, as it is a hybrid of a first order upwind scheme and the second order MacCormack algorithm (Warming et al. [1973]). It is used here for velocity flux (denoted **(phi, U)** in *fvSchemes*) in the simulations, as it is advisable to have at least second order accuracy for this term (Ferziger and Peric [2012]). For the linear upwind scheme, Holzmann [2018] comments that it "increases the accuracy while being more stable - compared to the upwind and linear schemes."

Stability

The scheme has a larger stability bound than the separate algorithms, as it corrects to the most stable one (Warming and Beam [1976]). Knowing that the upwind scheme has $Co < 1$ as a limit (see 3.3.1), this is also sufficient when using the linear upwind scheme.

Chapter 4

OpenFOAM

First released in 2004, OpenFOAM is a free, open-source toolbox for CFD developed by OpenCFD Ltd (OpenCFD [2018b]). At the time of writing, the latest official release is OpenFOAM-v1712. The version used in this thesis, both on the personal desktop and on Fram, is version 5.0, released in July 2017.

What sets OpenFOAM apart from most commercial CFD programs is the lack of a GUI. All input is done with text files, written in C++. Apart from this, workflow is similar to that of other CFD software. It can be split into three distinct parts:

- **Preprocessing**: The user decides on the extent of the computational domain, and creates a mesh. Depending on the size, resolution and complexity of the grid, this can take anywhere from a few hours to several weeks. A lot of care should be taken here; garbage in, garbage out applies strongly.
- **Solving**: Boundary conditions, solver and solver settings, length of the simulation, available computing power and required quality of the retrieved statistics all influence the time spent in this step. A steady-state solution where one is looking for mean drag in a small domain is a lot less expensive than a DNS looking for triple correlations to tune a Reynolds stress transport model.
- **Postprocessing**: Depending on the goal, all available statistics can get extracted and made more accessible with graphs, isovels, streamlines or 2D figures using one of the vast number of available tools. OpenFOAM comes prepackaged with ParaView, but can also produce raw data or gnuplot files.

In the following sections, the file structure used by OpenFOAM will be explained, before the steps above are investigated more thoroughly, including some examples from the experience gained during the work with this thesis. All names of files and folders are written in italic (e.g. *system*), while names of commands are in bold text (e.g. **mpirun -np 128 simpleFoam**). Modules like ParaView and simpleFoam are in plain text.

4.1 File structure in OpenFOAM

OpenFOAM is not run through a GUI, but rather through file manipulation and command line arguments. There are many ways to access and run an OpenFOAM case, but only the standard method used in the tutorials is presented here. In figure 4.1 the file structure of *cavity*, a tutorial case, is shown.

```
.
├── 0
│   ├── epsilon
│   ├── k
│   ├── nut
│   ├── nuTilda
│   ├── omega
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   ├── transportProperties
│   └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    └── fvSolution
```

Figure 4.1: File structure from a tutorial case.

The *0* folder is a time folder, containing boundary conditions for the solver. During a run, several file folders with data from the flow field at different times will be made. In *constant*, information about the fluid being simulated, *transportProperties*, and about the turbulence, *turbulenceProperties*, resides. The mesh is stored in *polyMesh*. In *system* information about

the solvers is stored. This includes discretization schemes, residual tolerances, time step lengths and more.

OpenFOAM solvers can also be run in parallel. The *decomposeParDict* file seen in figure 4.1 contains information about how to decompose the domain.

4.2 Preprocessing

If a pre-decided geometry is to be studied, the choice of computational domain is mostly given. The caveat is that flow must be aptly developed before it reaches an area where statistics are extracted. This can be achieved through imposing an inlet profile, having a intro section or using cyclic boundary conditions. If instead one is looking to investigate a phenomenon, both the size and shape of the domain must be established from principles.

There are two main tools for generating meshes in OpenFOAM; blockMesh and snappyHexMesh. The former relies only on the dictionary *blockMeshDict* for input, while the latter can use information stored in .nas, .stl or .obj files as well as its standard input, *snappyHexMeshDict*.

4.2.1 Meshing with blockMesh

For geometries easily defined by blocks or cylinders, blockMesh is an accurate and reliable tool. Using the input dictionary *blockMeshDict* as shown in figure 4.2, it creates hexahedral fully structured blocks. The vertices must be defined, and the edges can be straight lines, arcs or splines. Number of grid cells and size grading in each direction can be varied. Running blockMesh generates a mesh, stored in the *constant/polyMesh* folder as files named *points*, *boundary*, *faces* and *cells*.

4.2.2 Meshing with snappyHexMesh

The main tool for meshing complex geometries to use with OpenFOAM is snappyHexMesh. To run it requires a *snappyHexMeshDict*, located in the *system* folder and a background mesh, stored in the *constant/polyMesh* folder. This is often generated using blockMesh, but any hexahedral mesh with an aspect ratio close to one will work. Additionally surface files that are to be meshed can be placed in the *constant/triSurface* folder. The main parts of the

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;


vertices
( (-0.00375 -0.11 -0.1) ... (-0.00327 0.1 0.1) );

blocks
( hex (0 1 2 3 4 5 6 7) (185 84 80) simpleGrading (1 1 1) );

edges
(
);

boundary
( outerWall { type   wall;  faces ( (0 1 5 4) ... (3 7 6 2) ); }  );
```

Figure 4.2: An example *blockMeshDict*, adopted from the grid test.

snappyHexMeshDict used in chapter 6 is shown in figure B.1. snappyHexMesh works in three, optional stages. Based on a surface file and a location in the mesh it starts by castellating the mesh. This entails removing any cells not inside the area where the background mesh cells intersect the surface file, and doing refinement in areas specified by the dictionary. Each refinement level splits each grid line in two, giving an eightfold increase in number of cells. Next, it snaps the cell surfaces to that of the surface file(s). Lastly, it adds boundary layers to the snapped mesh.

Each of these steps are not done in a random fashion; rather, the user is given large freedom to choose how the mesh is made. This has its pros and cons, as the balance between a good mesh and reasonable time to completion can be hard to strike. Refinement levels, quality parameters, snap settings and number of iterations decide the end result. Some important quality controls made by snappyHexMesh are (OpenCFD [2018b]):

- **maxNonOrtho**: Limits the non-orthogonality. Defined as the angle made by the vector between the two adjacent cell centres across the common face and the face normal.
- **maxBoundarySkewness:**: Drawing a line between cell centers, the distance from its intersection with the face to the face centre normalized by the length of the whole line.
- **maxConcave**: Concavity is set as the largest interior angle of a face.

- **maxFaceThicknessRatio**: Limits the aspect ratio of the cells. Calculated as ratio of longest to shortest edge.

For all of these, high values will often lead to convergence problems or instabilities when running simulations on the final mesh. Besides the quality parameters, there are several practical choices to be made in the dictionary. Three of the most important are:

- **nCellsBetweenLevels**: Sets the number of buffer cells between one level of refinement and the next. The number one here would signal a twofold jump in cell size, giving a large stretching ratio.
- **featureAngle**: An angle in degrees. All cells that see an angle above this will be refined to the highest level set in the dictionary.
- **tolerance**: This number times the maximum local edge length gives the radius inside of which each point searches for a surface to snap to.

The workflow of meshing with snappyHexMesh is largely one of trial and error, as it is hard to know a priori how a certain tweak to the dictionary will end up affecting the mesh. To asses the change, it is usually no other way than to mesh the geometry and look at the result. Besides a visual inspection in ParaView or a similar program, OpenFOAM has a command line utility called checkMesh. A sample output can be found in figure B.2.

4.3 Solving

OpenFOAM comes with a large number of solvers. The main categories of applications are (OpenCFD [2018d]):

- Incompressible flows
- Compressible flows
- Multiphase flows
- DNS
- Combustion

- Heat transfer and buoyancy-driven flows
- Particle-tracking flows

Within the categories, there are specialized solvers for turbulent flows, steady and transient flows as well as a bunch of other cases. In this thesis, the simpleFoam solver is used. It is developed for steady state, incompressible flows. The workflow of all solvers are somewhat similar, but depending on the variables different files and adjustments are necessary. For incompressible flows, the absolute value of pressure is irrelevant, as it is only a flow variable. For multiphase flows, a file giving the ratio of fluid in each cell is needed. The files and some adjustments needed for steady state solvers, more precisely simpleFoam, are up next.

4.3.1 The *system* folder

Holding all the information about the solution process, the files contained in the *system* folder are first in line.

controlDict

An example of a *controlDict* file can be seen in figure 4.3. Take special note of the entries regarding time; as simpleFoam is a steady state solver, it gives number of iterations instead of actual time. To control when data is written, the user can manipulate writeControl and writeInterval. The entry purgeWrite gives the number of time step folders to be kept, keeping all if set to 0. If one wishes to execute additional operations on the calculated field values during run time, these can be entered under functions. As these tools come under the umbrella of post-processing, they are discussed in section 4.4.

fvSchemes

The schemes used are given in *fvSchemes*. For the simulations in this thesis, the dictionary used is shown in figure 4.4. The entries are grouped by type (CFD Direct [2018]):

- **ddtSchemes**: First and second derivatives in time.
- **gdradSchemes**: Gradient calculation, ∇.

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10000;
deltaT          1;
writeControl    timeStep;
writeInterval   5000;
purgeWrite      1;

...

functions{      #includeFunc magSqr
                #include "headLoss"
                #include "cuttingPlane"
                #include "streamLines"
                #include "sample"

                ...
        }
```

Figure 4.3: Parts of the *controlDict* file used for the simulations.

- **divSchemes**: Divergence, $\nabla\cdot$.
- **laplacianSchemes**: Laplacian, ∇^2.
- **interpolationSchemes**: Cell to face interpolation of variables.
- **snGradSchemes**: The normal component of a gradient, seen from a face.
- **wallDist**: Distance to wall, used by the LRR model among others.

OpenFOAM comes with a vast number of schemes, but only a few are suited in each category. To see which schemes are used in the tutorials, a great command line tool is foamSearch. An output giving the schemes used for the velocity vector divergence is shown in figure 4.5.

fvSolution

The tolerances and settings of the solvers are controlled with this dictionary (CFD Direct [2018]). An example used for simpleFoam can be seen in figure B.3. Choices of smoothers,

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes              {default        Euler;}

gradSchemes             {default        Gauss linear;}

divSchemes              {default        none;
        div(phi,U)                      bounded Gauss linearUpwind grad(U);
        div(phi,k)                      bounded Gauss upwind;
        div(phi,epsilon)                bounded Gauss upwind;
        div(phi,nuTilda)                Gauss limitedLinear 1;
        div((nuEff*dev2(T(grad(U)))))   Gauss linear;
        div((nu*dev2(T(grad(U)))))      Gauss linear;}

laplacianSchemes        {default        Gauss linear corrected;}

interpolationSchemes    {default        linear;}

snGradSchemes           {default        corrected;}
```

Figure 4.4: The *fvSchemes* file used for the k-epsilon simulations.

preconditioners and fine-tuning of how the problem is solved are done here. When a time step goes through several iterations, different tolerances can be prescribed for the final step. This is also relevant for steady state solvers using the SIMPLE algorithm, as the correction step can be enforced a minimum amount of times. The most important entries are:

- **solver**: Chooses solver to use.
- **preconditioner**: Chooses preconditioner to use.
- **tolerance**: The lower threshold for the residual before the solver stops iterating.
- **relTol**: The lowest ratio of the residual to the initial residual before the solver stops iterating.
- **pRefValue**: For incompressible solvers, only relative pressure matters. This gives a reference pressure in the cell given by pRefCell.
- **relaxationFactors**: The amount of under-relaxation to apply to the fields. 0 is no change between iterations, while 1 is enforcing diagonal dominance.
- **nNonorthogonalCorrectors**: Specifies the number of times to solve the pressure correction equation, see section 3.4.1.

```
foamSearch -c $FOAM_TUTORIALS fvSchemes "divSchemes.div(phi,U)"

      3 div(phi,U)      bounded Gauss limitedLinear 0.2;
      2 div(phi,U)      bounded Gauss limitedLinearV 1;
      2 div(phi,U)      bounded Gauss linear;
      1 div(phi,U)      bounded Gauss linearUpwind grad;
      4 div(phi,U)      bounded Gauss linearUpwind grad(U);
      2 div(phi,U)      bounded Gauss linearUpwind limited;
      1 div(phi,U)      bounded Gauss linearUpwind unlimited;
      2 div(phi,U)      bounded Gauss linearUpwindV grad(U);
     21 div(phi,U)      bounded Gauss upwind;
      1 div(phi,U)      Gauss cubic;
      1 div(phi,U)      Gauss limitedLinear 1;
     29 "div\(phi.*,U.*\)" Gauss limitedLinearV 1;
     21 div(phi,U)      Gauss limitedLinearV 1;
      8 div(phi,U)      Gauss linear;
      8 div(phi,U)      Gauss linearUpwind grad(U);
      1 div(phi,U)      Gauss linearUpwind limited;
      1 div(phi,U)      Gauss linearUpwindV grad(U);
      8 div(phi,U)      Gauss LUST grad(U);
      1 div(phi,U)      Gauss LUST unlimitedGrad(U);
     23 div(phi,U)      Gauss upwind;
      1 div(phi,U)      Gauss vanLeerV;
```

Figure 4.5: Output of using foamSearch to find all the entries named div(phi,U) in the tutorials. Invoking the option -c counts the number of appearances.

4.3.2 The *0* folder

All the information about the initial fields are placed in this folder, or whichever folder is set as the start time. Only the boundary conditions used are discussed here; a full list can be found at OpenCFD [2018c]. In the same manner as for the numerical schemes, foamSearch can be used to see which boundary conditions the tutorials employ.

U

This file contains the boundary and initial conditions for the velocity field. In the simulations, the walls and inlet take Dirichlet conditions. The walls obey no-slip, while the inlet takes a fixed value equalling that of the internal field. At the outlet, a modified von Neumann condition allowing reverse flow is applied, called inletOutlet. The U file used in the simulations can be seen in figure B.4.

p

The pressure field is initiated based on this file. As can be seen in figure B.5, the walls and inlet take von Neumann zero gradient boundary conditions, while the outlet takes a Dirichlet

condition, set to zero.

k

The kinetic energy field is read from this file shown in figure B.6. A wall function based on the equations found in section 2.6.2 is applied at the walls. The inlet takes a Dirichlet condition with a fixed value, while the outlet uses the same inletOutlet condition as U.

epsilon

Here, the dissipation rate is set. A wall function is used for the walls, while the inlet and outlet gets a Dirichlet and von Neumann boundary condition, respectively. The file can be seen in figure B.7.

nut

Here, the boundary field for the turbulent viscosity is set. An important point is that if one uses a wall function for ν_t, as is done here, it should also be done in the k, *epsilon* and R files. At the inlet, the field is set based on k and epsilon with a boundary condition named calculated, see equation 2.51. The outlet is assigned a zero gradient boundary condition. The file can be seen in figure B.8.

4.3.3 The *constant* folder

The contents of this file are not subject to much change during solving, hence the name. The *polyMesh* folder and its contents are aptly explained in previous sections. That leaves two files; *transportProperties* and *turbulenceProperties*.

turbulenceProperties

This file lets the user decide how and if turbulence is modelled. The entry simulationType governs if the simulation should use a laminar, RANS or LES model. The file used for the simulations is shown in figure B.9.

transportProperties

The type of fluid that is being simulated is defined here. As the fluid here is water at room temperature, the viscosity, under the nu entry, is set to $10^{-6}\ m^2/s$. The file can be seen in figure B.10.

4.4 Post processing

Having reached a converged simulation, the question is what to do with it. Pure field data is hard to interpret, so some manipulation or data extraction is necessary. OpenFOAM comes with a lot of post-processing tools, available through both command line arguments and for use in dictionaries. An overview can be found under "Post-Processing CLI" at CFD Direct [2018]. The upside to this, versus importing the whole case into ParaView, is the comparatively minute data volume necessary to process. A large case with several million cells is very heavy to compute on, and requires much more memory than single surface files or text files.

4.4.1 Sampling

Sampling is done through the command line interface by invoking **-func sampleDict** to the post-processing command, or by adding the function to the *controlDict* file. The layout of the necessary dictionary is similar, the difference being the header. It is included in the *controlDict*, as seen in figure 4.3. The file referenced by **#include sample** is shown in figure B.11. Choices to be made in this dictionary are summarized in the following:

- **interpolationScheme**: Decides whether cell centre data is used, or if some interpolation method taking face values into account is employed.
- **setFormat**: The output format of the data. **Gnuplot** is used here, as it formats the data nicely for plotting afterwards.
- **fields**: Choose which fields to sample.
- **writeControl**: When to write the data.

- **type**: How the sampling points are decided. The alternative **uniform** coupled with a number of points samples uniformly along a line. **midPoint** is used here, sampling each time it reaches the halfway point between two faces.

The end result of the sampling as prescribed by figure B.11 is a tab separated files containing z and U values, with a header designed for gnuplot.

4.4.2 Volume field value samples

In addition to sampling along lines or curves, the average, peak or minimum values of volume fields can be sampled in three-dimensional areas. The way it is done here is by defining cell zones in snappyHexMesh, and then using the **fieldValueDelta** function to work on the values during runtime. The functions are defined in the file referenced by **#include headLoss** in figure 4.3. A part of it can be seen in figure B.12. The entries of interest are:

- **operation**: Which operation the fields should be subjected to. Includes addition, subtraction and averaging.
- **regionType**: Whether the object specified by **name** is a zone, face or something else.

The fields chosen are then specified further within the next brackets, named **region**. An example can be seen in figure B.12, where the volume average of total pressure in two regions is taken.

4.4.3 Streamlines

To visualize how the water flows through the tunnel, streamlines are a great tool. The file is referenced with **#include streamlines** in figure 4.3, and can be seen in its entirety in figure B.13. The entries include:

- **lifeTime**: Along how many steps the particles should be traced.
- **nSubCycle**: Number of steps per cell.

The output used here is VTK, which is easy to use with ParaView.

4.4.4 Cutting planes

Included by **#include cuttingPlane** in the *controlDict* file shown in figure 4.3, the cutting plane dictionary is shown in figure B.14. The only new entry here is the **pointAndNormalDict**, which tells the user a point inside the plane and the plane normal. The resulting planes are written in VTK format, and can be studied in ParaView.

4.4.5 Making plots

To visualize the sampled data from section 4.4.1, gnuplot is a great tool. Taking a file like the one in B.15 as input, it comes with great customizeability. All plots used in this thesis are made with gnuplot, unless otherwise stated.

Chapter 5

Mesh convergence study

The numerical method used here was shown to replicate laboratory results in my project work (Mølmann [2017]). The results are shown in figure 5.1, where laboratory and numerical head losses are compared over a wide range of flow rates.

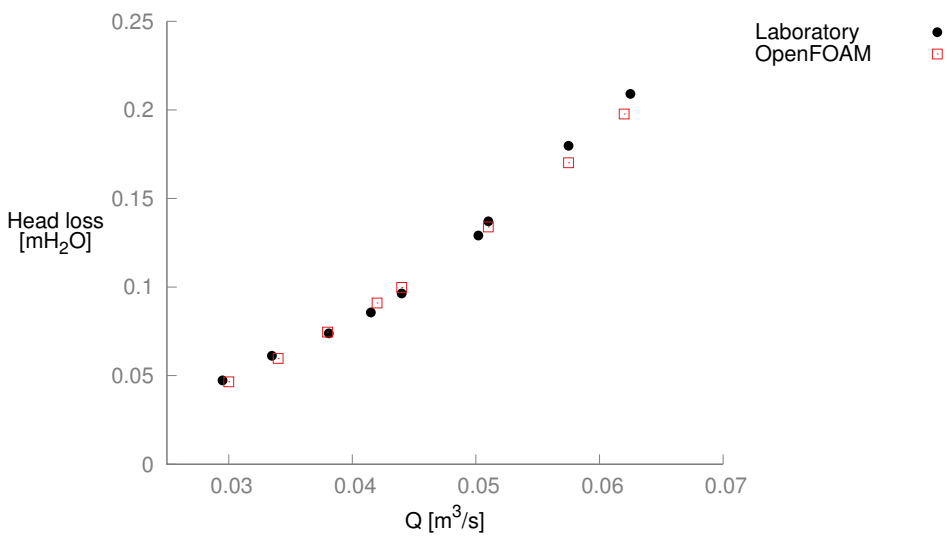

Figure 5.1: Results from the project work, showing numerical and laboratory measurements of head loss.

To test the mesh dependency of the type of problems that are to be tackled, a mesh convergence study is performed.

5.1 Geometry and mesh

For the mesh convergence study, the flow in a 45 cm long tunnel with two strips was solved with simpleFoam. Both strips were 5 mm high (into the flow) and 1 cm long (in the direction of the flow). The model can be seen in figure 5.2. Head loss between the inlet and outlet was

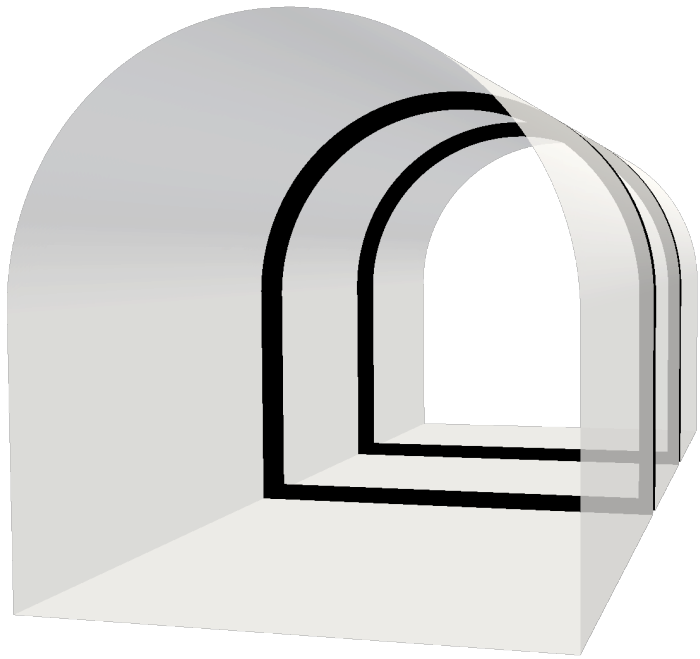

Figure 5.2: Geometry used in mesh convergence study. A=0.0208 m^2.

measured, as well as a velocity profile along the z-axis at x=33.075 cm. It is best practice when doing mesh convergence studies to start with the finest mesh, and then coarsen it by removing every second grid point to arrive at the other ones. This works well for structured grids, but lacks the flexibility needed when doing local refinement (Oberkampf and Roy [2010]). As the near field of the wall and strip needs to be refined in order to put y^+ within the log region, each mesh is made separately. For the two coarsest cases, the wall and the first 5 mm above it is refined up to the level of the second finest mesh, 2.5 mm. For the three coarsest meshes, the strips and the 5 mm above them were additionally refined to 1.25 mm to have a minimum resolution of the flow there. The finest mesh had no refinement.

Cell size	Number of cells
1 cm	164173
5 mm	361055
2.5 mm	698786
1.25 mm	4794371

Table 5.1: Base cell size and total number of cells for the four meshes

Variable	Internal	Inlet	Outlet	Walls
U $[m/s]$	2.0	*fixedValue*	*inletOutlet*	*noSlip*
k $[kgm^2/s]$	5.0×10^{-3}	*fixedValue*	*inletOutlet*	*kqRWallFunction*
ε $[m^2/s^3]$	9.7×10^{-2}	*fixedValue*	*zeroGradient*	*epsilonWallFunction*
nut $[m^2/s]$	2.3×10^{-5}	*calculated*	*zeroGradient*	*nutkWallFunction*
p $[m^2/s^2]$	0.0	*zeroGradient*	*fixedValue*	*zeroGradient*

Table 5.2: Boundary and initial conditions used for the mesh convergence study

5.1.1 Flow arrangement and boundary conditions

The boundary conditions used are shown in table 5.2. The *fixedValue* entries bear the values of the internal field. Using the hydraulic diameter of the tunnel as a length scale, the Reynolds number for the mesh convergence study is 3×10^5.

5.1.2 Velocity profile

All the meshes produced a similar profile, as seen in figure 5.3. There is some recirculation behind the strip, caught by the three finest meshes due to the refinement near the walls. In the main flow, the coarsest mesh differs substantially from the rest. The three finest meshes look very alike except some deviations by the 5 mm one.

5.1.3 Head loss

Head loss varied between the meshes, as seen in figure 5.4. The exact values are given in table 5.3. To understand the drastic jump in head loss between 5 and 2.5 mm, the size of the strip must be taken into account. Being 1 cm long in the streamwise direction, the acceleration of the flow is not captured adequately by the 5mm mesh, as only two cells cover the strip in the x-direction. The increase of cells from two to four is enough to capture this effect, increasing

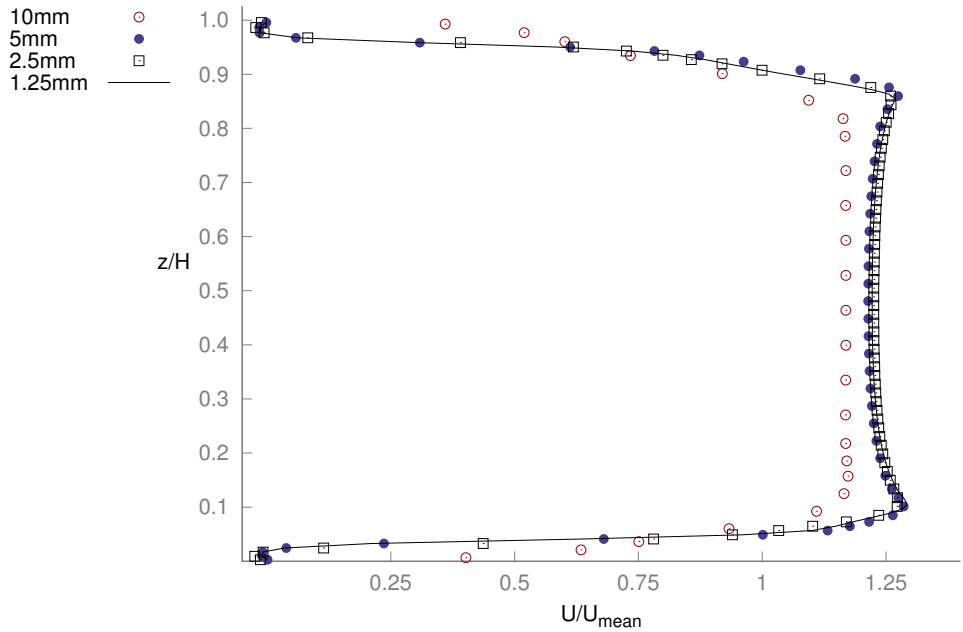

Figure 5.3: Vertical velocity profiles taken 0.75 mm behind the strip.

Cell size [mm]	Head loss [mH_2O]	Change from previous mesh [%]
10	0.06197	-
5	0.06186	-0.2
2.5	0.06582	+6.4
1.25	0.06563	-0.3

Table 5.3: Head losses for the different meshes.

the roughness introduced by the strip drastically. Between 2.5 and 1.25 mm the change is small enough that the 2.5 mm one is considered adequate.

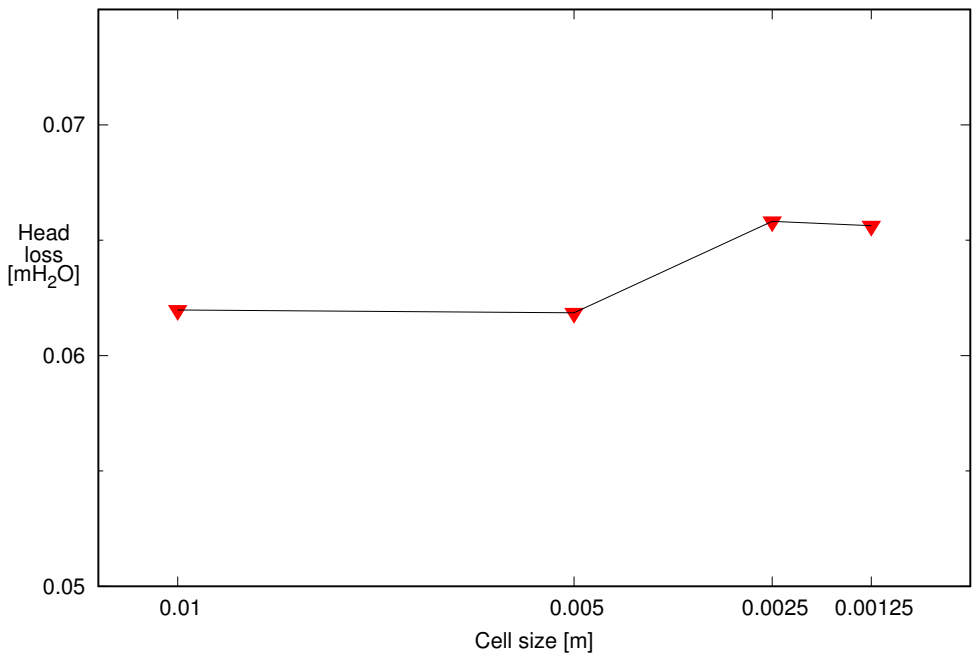

Figure 5.4: Head loss between in- and outlet for the different meshes.

Chapter 6

Shape test

To test the influence of shape on development and head loss, four different geometries are constructed. The geometries are named Circle, Normal tunnel, Birkeland and Square, and can be seen in figure 6.1. To make the size of the computational domain somewhat manageable while allowing the mesh resolution to be as fine as necessary to study the relevant phenomena, the geometries are made rather small, while still allowing fully turbulent flow to be developed. Based on estimates discussed in 2.7, $L_{max} = 30D$ gives an upper limit for the length that is expected for development. This ends up being 3 meters for all the geometries, and a total length of 3.5 meters is chosen to ensure that we get development in each case. Each geometry is equipped with strip spacings of 10, 15 and 20 centimetres, leaving 12 meshes in total. Flow velocities of 1, 2 and 4 m/s are tested for a total of 36 runs. Reynolds numbers calculated using the hydraulic diameter are 1×10^5, 2×10^5 and 4×10^5 for the three velocities tested across all geometries.

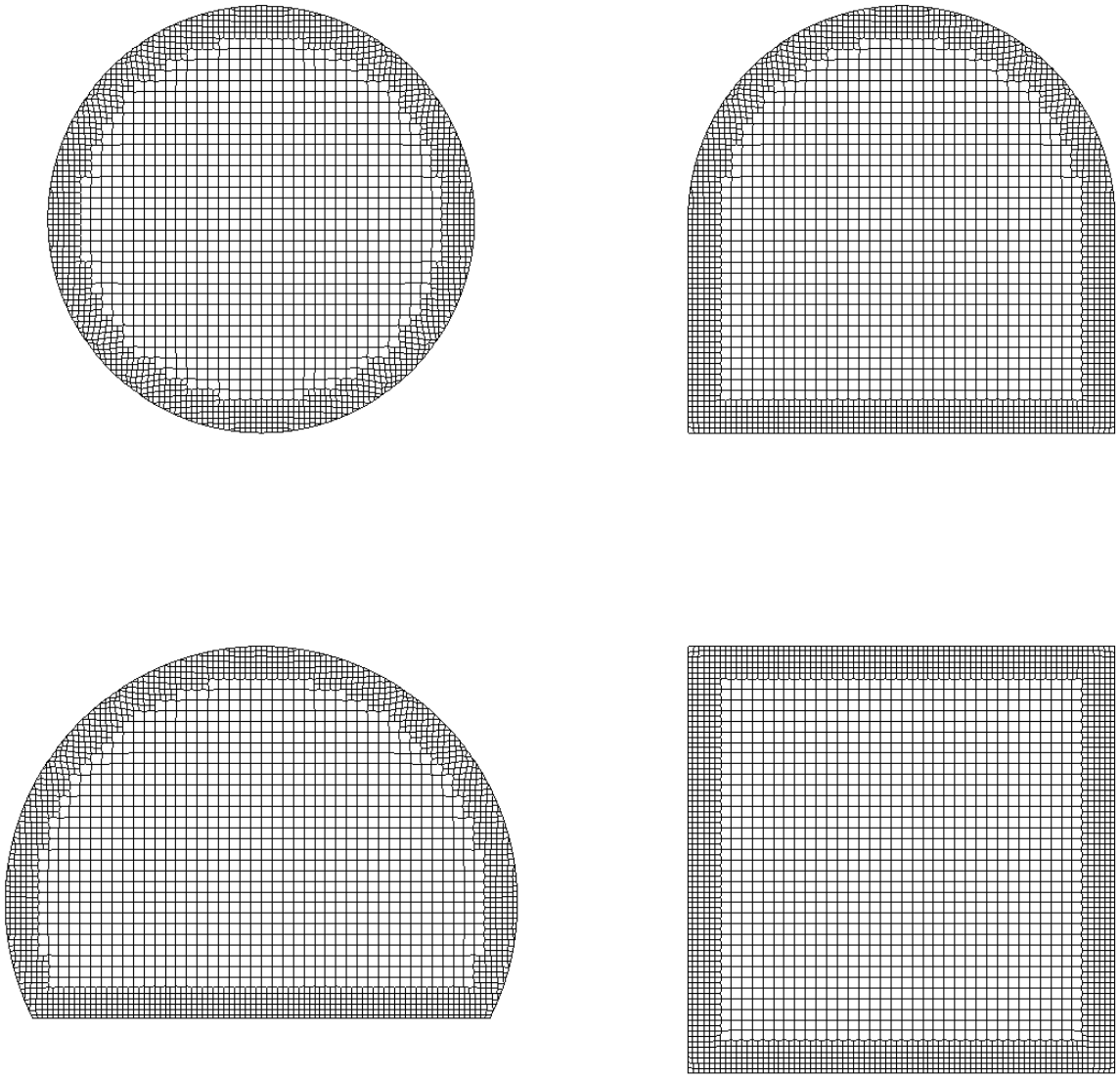

Figure 6.1: Inlet grids for the geometries used. All grids have primarily hexahedral cells with base cell size 2.5 mm. From the top left: "Circle", "Normal tunnel", "Birkeland" and "Square".

6.1 Geometries and boundary conditions

The parameter chosen for scaling is the hydraulic radius, as it is what Birkeland [2008] uses for his formula. All four of the geometries thus have the same hydraulic radius of 0.025 meters. The exact geometries are described in tables B.1 through B.4. In line with the results from section 5.1, a mesh with a base size of 2.5 mm was created with snappyHexMesh for each of the geometries. The total amount of cells, along with mesh quality stats, are shown in table 6.1.

Geometry	**Strip spacing**	**NC**	**MAR**	**MS**	**ANO**
Circle	10 cm	8.79×10^6	4.74	1.95	7.32
	15 cm	5.52×10^6	3.48	1.90	6.36
	20 cm	7.27×10^6	5.27	1.98	6.92
Birkeland	10 cm	10.27×10^6	5.29	2.52	6.91
	15 cm	6.29×10^6	3.51	1.88	5.99
	20 cm	8.29×10^6	5.29	2.51	6.52
Normal tunnel	10 cm	10.38×10^6	4.75	1.95	6.62
	15 cm	6.39×10^6	3.47	1.90	5.63
	20 cm	8.39×10^6	4.73	1.95	6.22
Square	10 cm	11.74×10^6	1.97	0.42	6.01
	15 cm	7.26×10^6	2.02	0.35	5.02
	20 cm	9.53×10^6	1.96	0.42	5.62

Table 6.1: Stats for the meshes used. Shown are number of cells (NC), max aspect ratio (MAR), max skewness (MS) and average non-orthogonality (ANO). See section 4.2.2 for definitions used.

The boundary conditions are similar to those used in section 5.1, and are summarized in tables 6.2 and 6.3. All the runs were done on Fram, utilizing simpleFoam on 128 cores. Convergence was defined as the residuals reaching 10^{-8} for the velocity field and 10^{-6} for the pressure and turbulence fields.

Variable	Inlet	Outlet	Walls
U $[m/s]$	*fixedValue*	*inletOutlet*	*noSlip*
k $[kgm^2/s]$	*fixedValue*	*inletOutlet*	*kqRWallFunction*
ε $[m^2/s^3]$	*fixedValue*	*zeroGradient*	*epsilonWallFunction*
nut $[m^2/s]$	*calculated*	*zeroGradient*	*nutkWallFunction*
p $[m^2/s^2]$	*zeroGradient*	*fixedValue*	*zeroGradient*

Table 6.2: Boundary conditions used for the shape test runs. The *fixedValue* entries take the value of the internal fields, shown in table 6.3.

	1 m/s	2 m/s	4 m/s
U $[m/s]$	1.0	2.0	4.0
k $[kgm^2/s]$	1.25×10^{-3}	5.0×10^{-3}	2.0×10^{-2}
ε $[m^2/s^3]$	5.81×10^{-2}	1.16×10^{-1}	2.32×10^{-1}
nut $[m^2/s]$	2.42×10^{-6}	1.94×10^{-5}	1.55×10^{-4}
p $[m^2/s^2]$	0.0	0.0	0.0

Table 6.3: Internal field values used for the shape tests.

6.2 Head loss and friction factor

The goal of the Birkeland equation is to predict head loss through a friction factor, λ. Using his equation, found in 1.4, to predict the rough friction of the geometries give the results shown in table 6.4. The smooth friction factors are found using Moodys diagram with $Re = 10^5$ and a smooth pipe.

	λ_{smooth}	λ_{rough}			λ_{total}		
	All	10 cm	15 cm	20 cm	10 cm	15 cm	20 cm
Circle	0.018	0.202	0.142	0.112	0.220	0.160	0.130
Square	0.018	0.258	0.181	0.142	0.276	0.199	0.160
Birkeland	0.018	0.227	0.159	0.125	0.245	0.177	0.143
Normal tunnel	0.018	0.230	0.162	0.127	0.248	0.180	0.145

Table 6.4: Friction factors λ, calculated using Birkelands equation. Smooth friction is from Moodys diagram, using $Re = 10^5$ and a smooth pipe.

To have an even comparison between geometries, measurements are made after the flow is developed. See section 6.3 for these lengths. Total pressure is measured as a volume average at two points, both centered between strips. The smallest even factor of the strip spacings (10, 15 and 20) is 60, and the points were thus picked out to be 0.6 meters apart. Using equation 1.3, the resulting head loss values are turned into friction factors. The friction factors are shown in figure 6.2, while all head loss data are shown in table B.6 in the appendix.

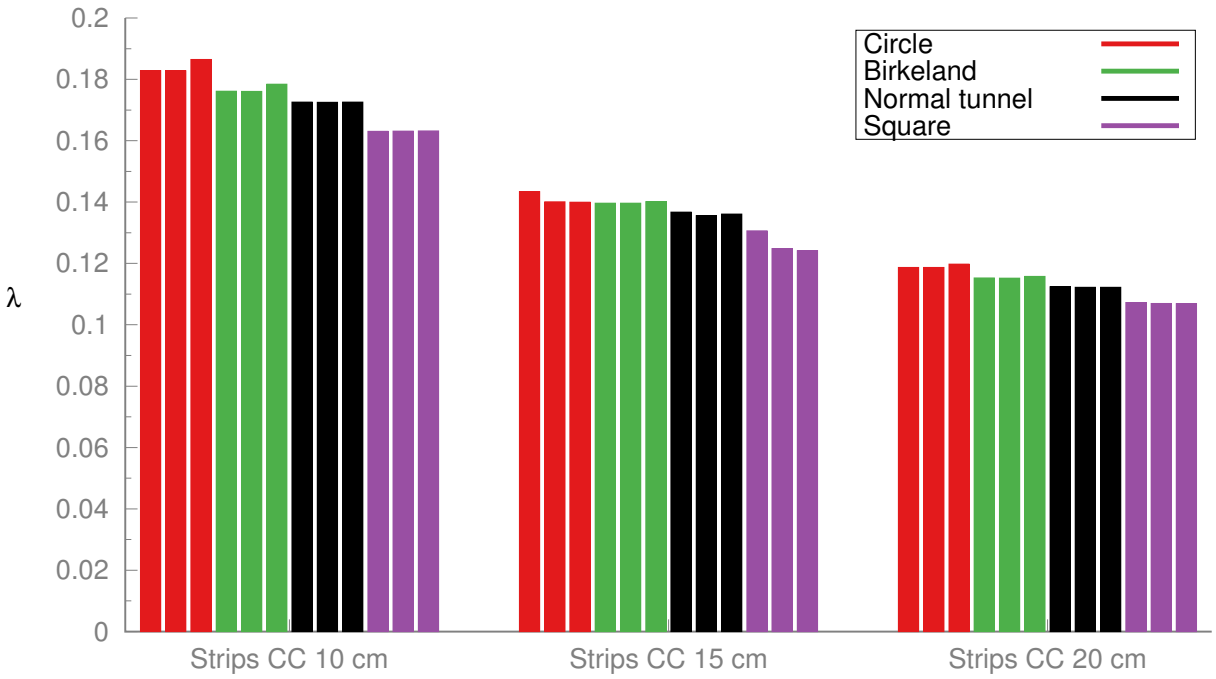

Figure 6.2: Measured friction factor λ for all geometries and velocities. For each unique geometry, the velocities are plotted in this order: 1 m/s, 2 m/s and 4 m/s.

Looking at the results for CC=10 cm, the "Circle" and "Birkeland" geometries stand out. Their calculated friction factors for U=4 m/s are quite a bit higher than those for U=1 m/s and U=2 m/s. This will be investigated in section 6.5. For the "Circle" and "Square" geometries with CC=15 cm, the results for U=1 m/s are higher than those for the higher velocities. No further enquiry has been made here.

The results differ substantially from those calculated with the Birkeland equation. Table 6.5 shows the differences in percentages. Using the closest value of the average and maximum friction factor for all geometries, the calculated values are off by between 8 and 41 percent, with an average of 24 percent.

		Birkeland equation	Mean value	Difference	Maximum value	Difference
10 cm	Circle	0.220	0.184	16%	0.186	15%
	Birkeland	0.245	0.177	28%	0.178	27%
	Normal tunnel	0.248	0.173	30%	0.173	30%
	Square	0.276	0.163	41%	0.163	41%
15 cm	Circle	0.160	0.141	12%	0.143	11%
	Birkeland	0.177	0.140	21%	0.140	21%
	Normal tunnel	0.180	0.136	24%	0.137	24%
	Square	0.199	0.127	37%	0.131	34%
20 cm	Circle	0.130	0.119	8%	0.120	8%
	Birkeland	0.143	0.115	19%	0.116	19%
	Normal tunnel	0.145	0.112	22%	0.112	22%
	Square	0.160	0.107	33%	0.107	33%

Table 6.5: Calculated friction values with the Birkeland equation and the simulation results. Percentage difference is given as fraction of the result from the Birkeland equation. Mean values are averages of the three velocities used for each geometry.

6.3 Head loss development

To see when the head loss stabilizes, the total pressure is recorded at the midpoint between each strip. This gives a resolution of the results equalling the strip distance. The percentage change between head losses over strips is calculated, and a stable value under ±0.25% is the criteria chosen for a fully developed flow. Graphs detailing the development for each case are shown in appendix B.2.

Figure 6.3 shows the results. Keep in mind the resolution; a difference of 20 cm in the latter graph can be really minute, as it is the lowest possible value it can take in the case of CC=20 cm. A closer look at the graphs in the appendix is necessary in those cases.

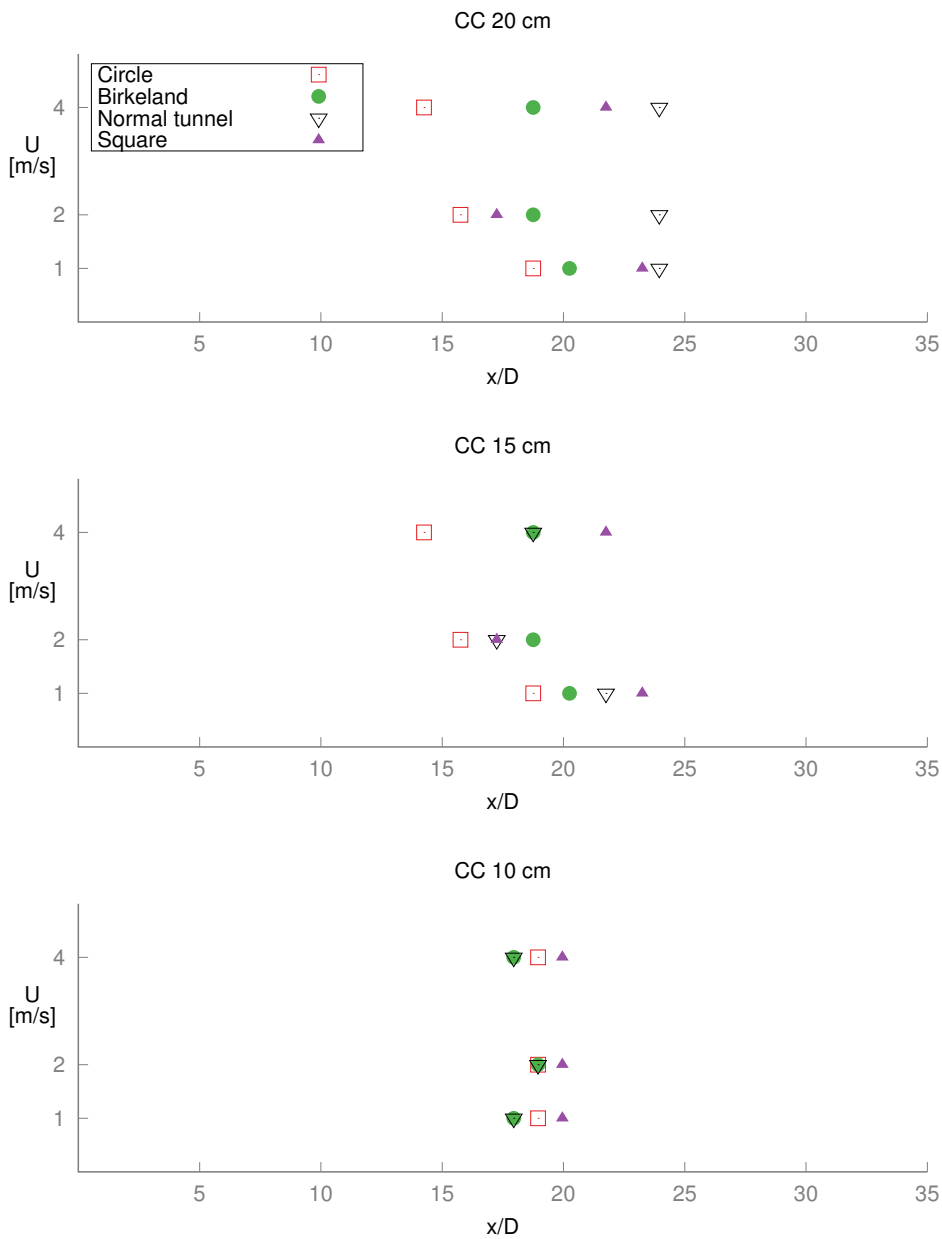

Figure 6.3: Length to development for all geometries. Distance is downstream of inlet, normalized by $D = 4 \times R_{hydraulic}$.

Some general observations can be made right away. The head loss becomes uniform fastest in the "Circle" geometry in every single case. The other three are more variable; for 10 and 15 cm strip distance, development is slower in the "Square" geometry than named "Birkeland" and "Normal tunnel". For 20 cm, however, the "Normal tunnel" is slower than the two others. In general, the difference between the shapes becomes more accentuated with greater strip distance and velocity.

The results seem to map quite neatly on the friction factor results, with higher friction geometries developing faster. One exception is the "Circle" geometry, which develops quicker with 20 cm strip spacing. The explanation can be found in figures B.22 and B.23 in the appendix. Although considered developed by the criteria chosen, the values still oscillate somewhat for another meter or so downstream, barely staying within the marked area.

The theory in 2.7 presents the correlation $L/D \approx C/\lambda$. This C is plotted in figure 6.4. The spread is extremely tight for CC 10 cm, but widens out significantly for the larger strip spacings. This is again a result of the resolution; a 20 cm difference in development length is really minute for the largest strip distance, but puts a large dent in the calculated C.

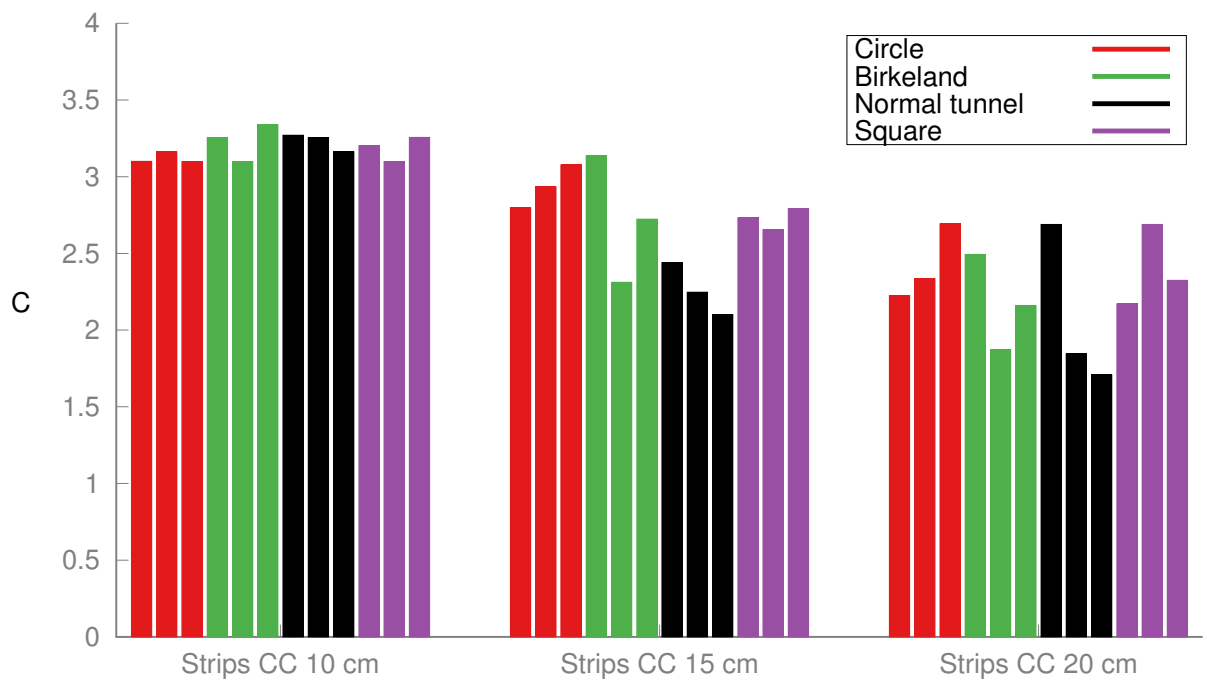

Figure 6.4: Development constant $C = \lambda L/D$ for all geometries, plotted in order of increasing velocity for each. $D = 4 \times R_{hydraulic}$, λ is the total friction factor and L is length to development in meters. Note that λ is measured after the flow is developed, and is thus a function of the geometry, not the processes guiding flow development.

The inlet profiles are all uniform, which may impact development length depending on how different the resulting profile turns out to be. Figure 6.5 show velocity profiles taken between the strips for CC = 20 cm with U = 4 m/s for all cross section shapes.

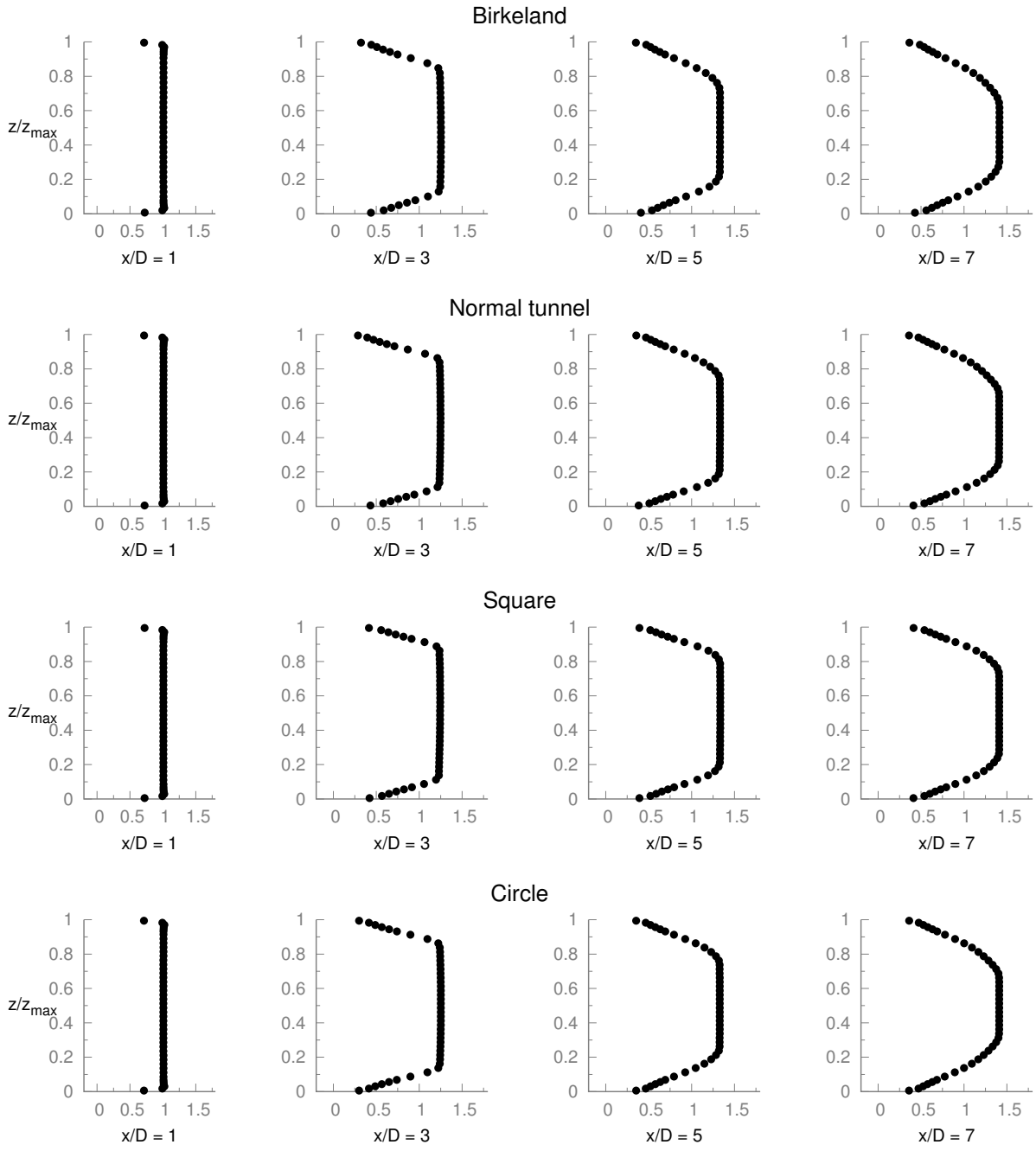

Figure 6.5: Shown are velocities, normalized by $U_{mean} = 4\ m/s$, right after the inlet for the CC = 20 cm cases. Downstream distance is normalized by $D = 4 \times R_{hydraulic}$.

The only visible difference is a slower development of the velocity profile belonging to the "Square" geometry, the rest behave very similarly. It is however clear why the head loss appears strange over the first few strips, as the velocity profiles are rather artificial in this part of the domain.

6.4 Secondary currents

The difference in length to development seems to correlate strongly with the head loss, as discussed above. Another factor which is a priori known to be different between cross section shapes, and that affects the head loss, is secondary currents.

Secondary currents take energy away from the mean flow, see section 2.8. In a smooth pipe, a streamwise increase in the magnitude of the secondary currents is expected to be accompanied by a greater pressure loss, which in turn leads to quicker development. In rough pipes, and in this case ones with discrete roughness elements, there are additional effects at play. The strips interfere with the secondary currents and the turbulence field, possibly erasing the effect of the secondary currents alone.

To measure the strength of the secondary currents, the magnitude of all velocity components is sampled. A higher value of $\frac{|U_y|+|U_z|}{|U_x|}$ indicates stronger secondary currents.

With secondary flows often being referred to as "corner flows" in the literature, the not so surprising leader of the pack is the "Square" geometry. The two tunnel shapes, "Birkeland" and "Normal tunnel" follow suit, while the "Circle" geometry has the weakest secondary currents.

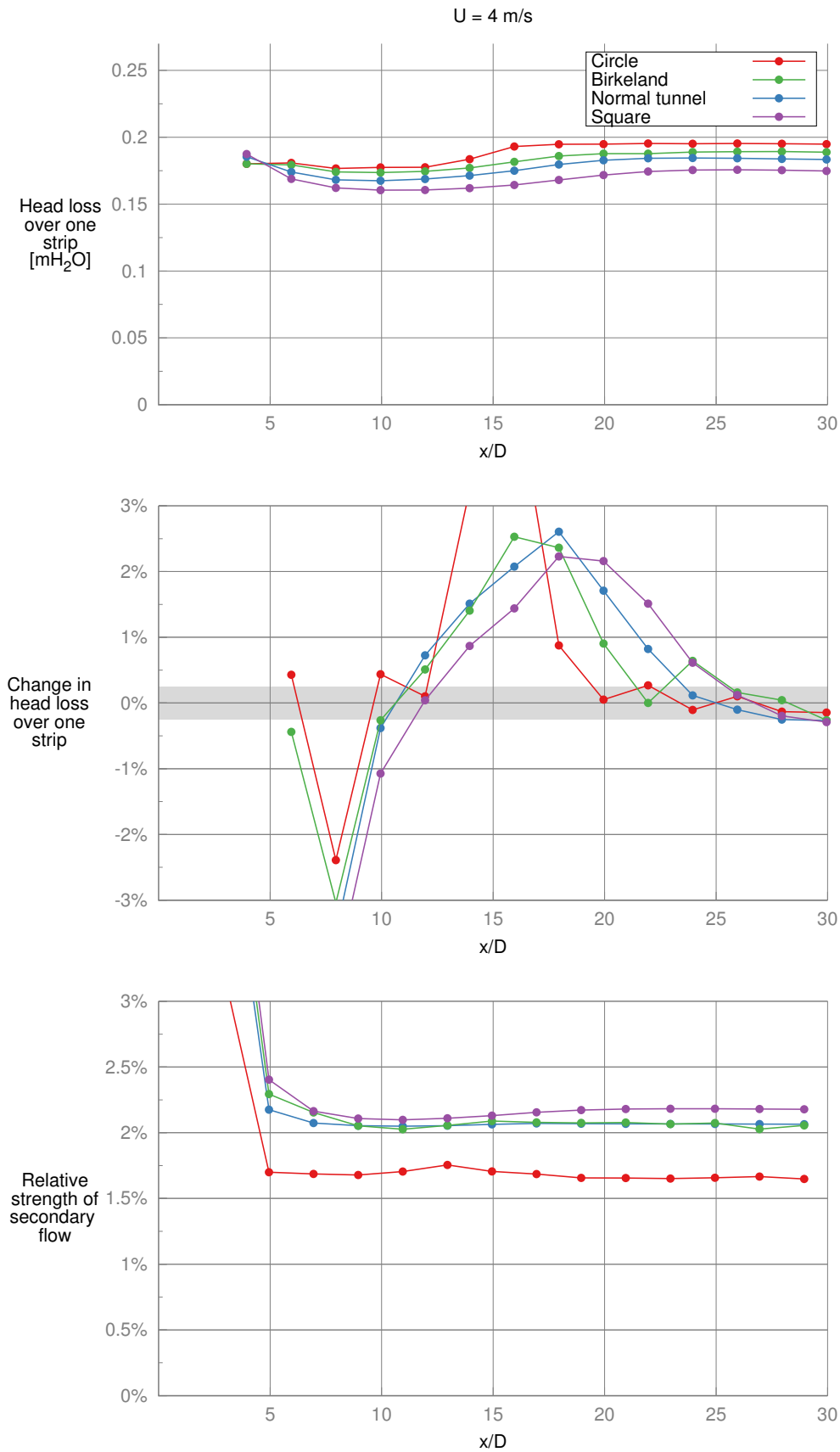

Figure 6.6: Head loss per strip, development length and strength of secondary currents for CC 20 cm and U = 4m/s. Relative strength is measured as $\frac{|U_y|+|U_z|}{|U_x|}$, summed over all cells and weighted by cell volume. Distance is downstream of inlet, normalized by $D = 4 \times R_{hydraulic}$.

6.5 Streamlines and velocity profiles

To get an idea about why the head loss and thus the friction factor is higher than expected for the case of U = 4 m/s in some geometries, streamlines are extracted. Using the "Circle" geometry with CC = 10 cm, a line is seeded some millimeters above a strip at x = 3.095 m, well into the developed region. Using a backdrop of the velocities in the x-direction, figure 6.7 shows the result. Recirculation and stagnation zones can be seen as dark blue.

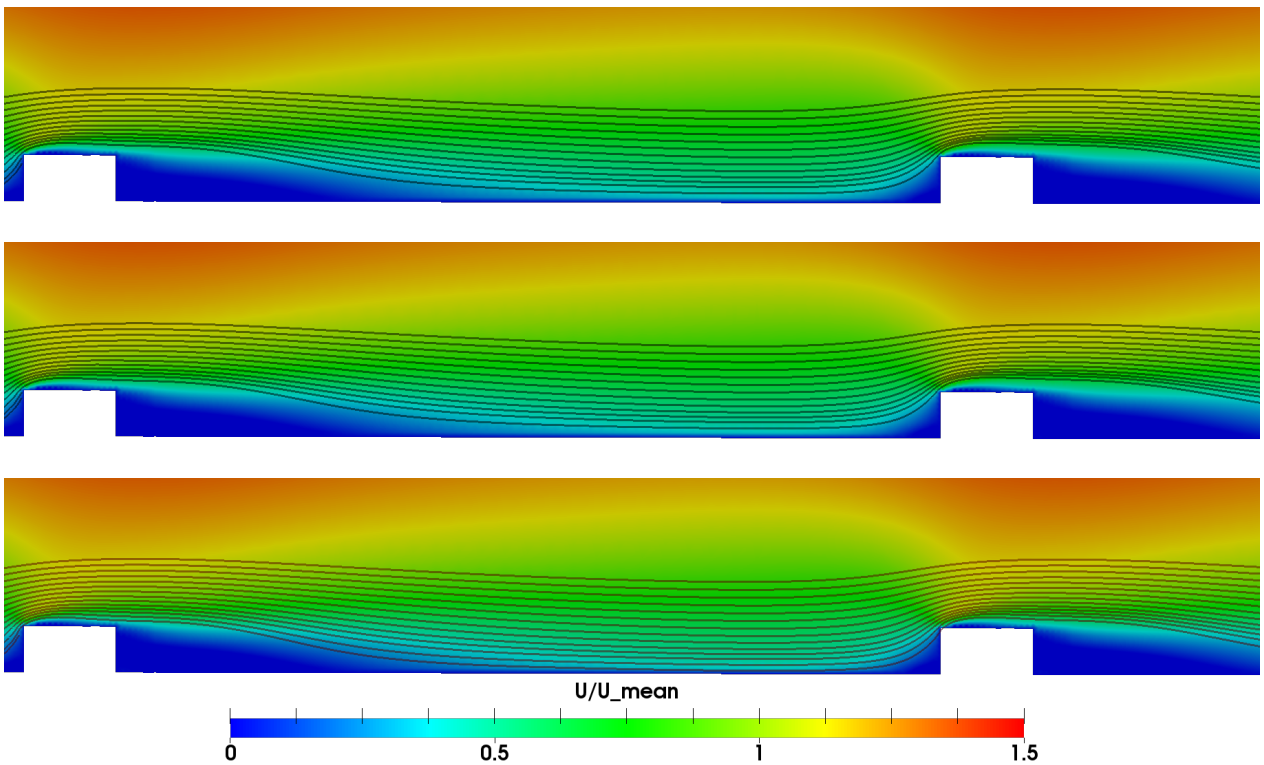

Figure 6.7: Shown here are the normalized velocity fields and streamlines over the strips at x=3.095 m and x=3.195 m for the geometry "Circle" with strip CC = 10 cm. From top to bottom: U = 1 m/s, 2 m/s and 4 m/s. Only the lower 2 centimeters of the geometry is shown.

All the streamlines look similar. A suspect for causing the extra friction was an excessive bending of the streamlines close to the strips at the higher velocities, but this seems not to be the case. The recirculation zones are also of similar size. When normalized by mean velocity, it becomes obvious that the flow fields behave the same. A closer look at the velocity profiles is given below. These span one strip CC distance from a few millimeters after one strip to a few millimeter in front of the next.

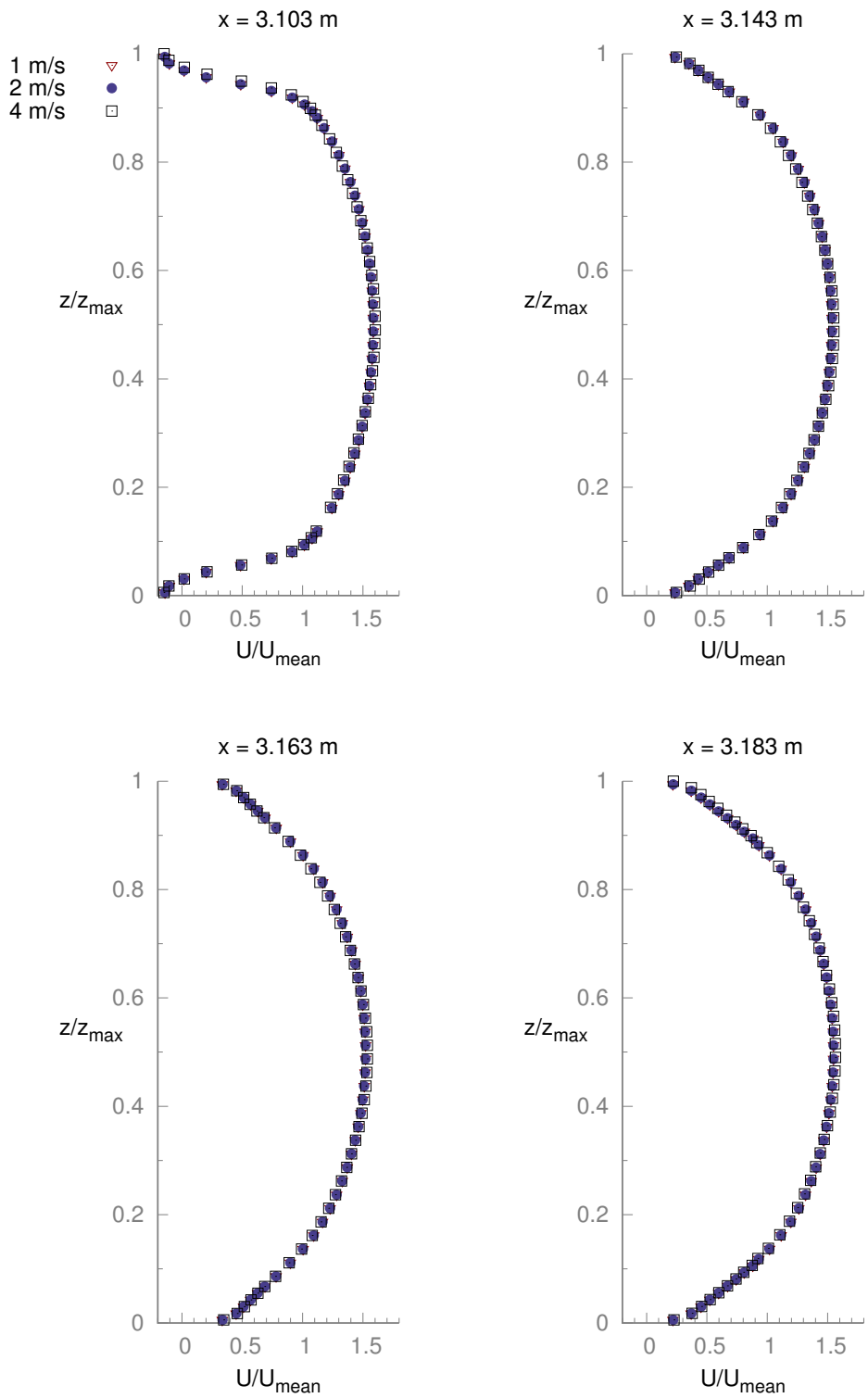

Figure 6.8: Velocity profiles, taken along the stretch marked by streamlines in figure 6.7. The strips span x = 3.09-3.10 m and x = 3.19-3.20 m. Z-values are normalized by total height, and velocities are normalized by mean velocity.

The velocity profiles seen in figure 6.8 map onto each other neatly when normalized by mean velocity. No clear indication of different effects for the highest velocity can be seen.

Chapter 7

Discussion

The Birkeland formula overestimates the friction factor for all tested geometries. As all measurements done by Birkeland [2008] were taken well inside what was found as the undeveloped region, a certain error is to be expected. Which side this errors leans to is however not given a priori; head loss values were both higher and lower before they stabilized across the geometries and velocities tested.

Among the variables tested, length to development had the starkest difference between the geometries. Far from all laboratory tunnels using strip roughness are as long as the development length, and many will not experience uniform head loss across strips in their models before the outlet. Looking at the secondary currents, the Birkeland equation missed by the most for the geometries where secondary currents were at their strongest. If this is a direct correlation is not known. The secondary currents may alter the impact of the strips on the velocity and pressure field, but in what way this would change head loss is hard to see from the formulas alone. Without any further parameters being analyzed, nothing can be concluded about the impact of the secondary currents. A relation between velocity and size of the recirculation zone was not evident in these results. Larger strips or higher velocities might be necessary to lengthen the zone enough that the wake of one strip goes as far downstream as 10 cm. The assumption of isolated roughness flow, introduced by Morris [1963], seems to hold true here.

When it comes to the accuracy of the numerical method, a few factors must be mentioned. The wall modelling employed is questionable. Its domain of accuracy, flat plane boundary layers at high Reynolds numbers, is at best approximated in parts of the wall with strip CC set at 20 cm. For the remainder of the domain, high pressure gradients and mean streamline

curvature are likely to hinder the accuracy of the wall modelling. The separation induced by the strips also leads to the failure of the assumptions behind the wall laws (Pope [2000]). Even if the logarithmic law were to hold for all parts of the wall, the largest y^+-values found in table B.5 are outside of the logarithmic area ($30 < y^+ < 200$). How this impacts the results is hard to ascertain. The boundary conditions are also an issue. As discussed at the end of section 6.3, the velocity profiles are not as expected of a physical system at the start of the domain. It does however seem like the length to development receives a fair treatment by letting all the profiles start out equal. An alternative would be to map them from either a point downstream or from a previous case. This has the advantage of shortening the part of the domain which can not be used for study, but with the downside of leaving the impact of the inlet conditions very hard to quantify.

As for the equation and its merits, it must be made clear that no factor outside of the friction investigated in this thesis was meant to be resolved by Birkeland. His equation serves as a tool for first estimation of friction, not as an alternative to pressure measurements in the laboratory. The burden is on the user to be aware of the equations limitations, especially when straying too far from the conditions present at its inception.

Chapter 8

Conclusions

Through numerical simulation of four different cross sections, different aspects of the Birkeland equation for estimating friction have been tested. Across the 36 total tests, the friction factors were overestimated by an average of 24%. Length to development was also investigated: Starting out with a uniform velocity profile gave values between 2 and 3.5 of $C = \lambda L/D$, with L being length to development and D being four times the hydraulic radius, for most geometries and flow velocities. Looking at the strength of secondary flows, a not significant correlation to both head loss and length to development was found. No evidence was found for failure of the assumption that the flow of the cases was in the isolated roughness regime.

8.1 Further work

Several factors mentioned in section 1.3 are not tested in this thesis. Especially interesting are the effect of accelerating inflow and multiphase flow on the head loss induced by the strips. As hydropower tunnels are running well under full capacity a lot of the time, an accurate model for friction in the case of an air-water mixture could be helpful in the laboratory. As for techniques not used in this thesis, utilizing particle image velocimetry (PIV) in conjunction with DES modelling of the flow around the strips can deepen the knowledge of their effect on the flow field while also serve to tune the DES models employed.

Bibliography

J. H. Bell and R. D. Mehta. Development of a two-stream mixing layer from tripped and untripped boundary layers. *AIAA journal*, 28(12):2034–2042, 1990. doi: 10.2514/3.10519.

R. Birkeland. Modellstudie av flomløpet på dam Sysenvatn. Master thesis, NTNU, June 2008.

J. Boussinesq. *Essai sur la théorie des eaux courantes*. Impr. nationale, 1877.

K. M. Brøste. Testing av luftmedrivning på flomløp Follsjødammen. Master thesis, NTNU, June 2017.

M. Casey and T. Wintergerste. *ERCOFTAC Special Interest Group on Quality and Trust in Industrial CFD - Best Practice Guidelines*. European research community on flow, turbulence and combustion, 2000.

CFD Direct. OpenFOAM User Guide. https://cfd.direct/openfoam/user-guide/, 2018. Accessed: 2018-03-01.

R. Dean and P. Bradshaw. Measurements of interacting turbulent shear layers in a duct. *Journal of fluid mechanics*, 78(4):641–676, 1976. doi: 10.1017/S002211207600267X.

J. C. Dudek and J.-R. Carlson. Evaluation of full reynolds stress turbulence models in fun3d. *AIAA SciTech 2017 Forum*, 2017. doi: 10.2514/6.2017-0541.

P. A. Durbin and B. P. Reif. *Statistical theory and modeling for turbulent flows*. John Wiley & Sons, 2011.

O. B. Ekeade. Scale model measurements of headloss in unlined tunnels. Master thesis, NTNU, June 2017.

J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.

A. Garcia. Scale model measurements of head loss in unlined tunnels. Bachelor thesis, NTNU, June 2017.

B. A. U. Gjerde. Dam Storlivatnet flomløp - modellforsøk for nytt flomløp. Master thesis, NTNU, June 2017.

A. Harten. High resolution schemes for hyperbolic conservation laws. *Journal of computational physics*, 49(3):357–393, 1983. doi: 10.1016/0021-9991(83)90136-5.

C. Hirsch. *Numerical Computation of Internal and External Flows: Fundamentals of Computational Fluid Dynamics*. Butterworth-Heinemann. Elsevier/Butterworth-Heinemann, 2007.

T. Holzmann. Numerical schemes. https://holzmann-cfd.de/numerical-schemes/case-description/the-equation-and-conditions, 2018. Accessed: 2018-05-01.

P. M. John Kim and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of fluid mechanics*, 177, 1987. doi: 10.1017/S0022112087000892.

A. Karvinen and H. Ahlstedt. Comparison of turbulence models in case of three-dimensional diffuser. In *Proceedings of Open Source CFD International Conference*, volume 2008, pages 1–17, 2008.

A. N. Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Dokl. Akad. Nauk SSSR*, 30(4):299–303, 1941. doi: 10.1098/rspa.1991.0075.

S. Konangi and N. K. Palakurthi. von Neumann Stability Analysis of a Segregated Pressure-Based Solution Scheme for One-Dimensional and Two-Dimensional Flow Equations. *Journal of Fluids Engineering*, 138, 2016. doi: 10.1115/1.4033958.

B. E. Launder, G. J. Reece, and W. Rodi. Progress in the development of a reynolds-stress turbulence closure. *Journal of Fluid Mechanics*, 68(3):537–566, 1975. doi: 10.1017/S0022112075001814.

P. D. Lax and R. D. Richtmyer. Survey of the Stability of Linear Finite Difference Equations. *Communications on Pure and Applied Mathematics*, 9:267–293, 1956. doi: 10.1002/cpa.3160090206.

F. Liu. A Thorough Description Of How Wall Functions Are Implemented In OpenFOAM. Project work, Chalmers University of Technology, January 2017.

D. Miller. *Internal Flow Systems*. BHRA (Information Services), 1990.

H. M. Morris. Applied hydraulics in engineering. Technical report, 1963.

F. Moukalled, L. Mangani, M. Darwish, et al. *The finite volume method in computational fluid dynamics*. Springer, 2016.

E. Mølmann. 3D Numerical Modelling of a Lab Tunnel. Project work, NTNU, December 2017.

F. Nestmann. *Multiphase Flow in Hydraulic Engineering*. Lecture notes, 2017.

I. Nezu, A. Tominaga, and H. Nakagawa. Field measurements of secondary currents in straight rivers. *Journal of Hydraulic Engineering*, 119(5):598–614, 1993. doi: 10.1061/(ASCE)0733-9429(1993)119:5(598).

N. Nikitin, F. Nicoud, B. Wasistho, K. Squires, and P. Spalart. An approach to wall modeling in large-eddy simulations. *Physics of fluids*, 12(7):1629–1632, 2000. doi: 10.1063/1.870414.

W. L. Oberkampf and C. J. Roy. *Verification and validation in scientific computing*. Cambridge University Press, 2010.

N. R. B. Olsen. *Numerical Modelling and Hydraulics*. NTNU, 2012.

OpenCFD. OpenFOAM LRR model. https://www.openfoam.com/documentation/cpp-guide/html/classFoam_1_1RASModels_1_1LRR.html, 2018a. Accessed: 2018-03-01.

OpenCFD. Homepage of OpenCFD. https://www.openfoam.com/about, 2018b. Accessed: 2018-04-05.

OpenCFD. Boundary conditions. https://www.openfoam.com/documentation/user-guide/standard-boundaryconditions.php, 2018c. Accessed: 2018-04-05.

OpenCFD. Standard solvers. https://www.openfoam.com/documentation/user-guide/standard-solvers.php, 2018d. Accessed: 2018-04-05.

S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. In *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion*, pages 54–73. Elsevier, 1983. doi: 10.1016/B978-0-08-030937-8.50013-1.

S. Perzyna. Air return in brook intake shafts. Master thesis, NTNU, July 2017.

S. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

L. Prandtl. Zur berechnung der grenzschichten. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 18(1):77–82, 1938. doi: 10.1002/zamm.19380180111.

L. Prandtl. *Essentials of fluid dynamics: With applications to hydraulics aeronautics, meteorology, and other subjects*. Hafner Pub. Co., 1952.

R. S. Rogallo and P. Moin. Numerical simulation of turbulent flows. *Annual Review of Fluid Mechanics*, 16(1):99–137, 1984. doi: 10.1146/annurev.fl.16.010184.000531.

J. Rotta. Statistische theorie nichthomogener turbulenz. *Zeitschrift für Physik*, 129(6):547–572, 1951. doi: 10.1007/BF01330059.

A. Shabbir and T.-H. Shih. Critical assessment of reynolds stress turbulence models using homogeneous flows. *Aerospace Sciences Meeting*, 1992. doi: 10.2514/6.1993-82.

P. R. Spalart. Direct simulation of a turbulent boundary layer up to R_θ= 1410. *Journal of fluid mechanics*, 187:61–98, 1988. doi: 10.1017/S0022112088000345.

H. J. Tucker and A. Reynolds. The distortion of turbulence by irrotational plane strain. *Journal of fluid mechanics*, 32(4):657–673, 1968. doi: 10.1017/S0022112068000947.

M. Uhlmann. *Numerical Fluid Mechanics 1*. Lecture notes, 2012.

T. Von Kármán. *Mechanische Ähnlichkeit und Turbulenz*. Sonderdrucke aus den Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen : Mathematisch-physische Klasse. Weidmannsche Buchh., 1930.

Z. Warhaft. An experimental study of the effect of uniform strain on thermal fluctuations in grid-generated turbulence. *Journal of Fluid Mechanics*, 99(3):545–573, 1980. doi: 10.1017/S0022112080000766.

R. Warming and R. M. Beam. Upwind Second-Order Difference Schemes and Applications in Aerodynamic Flows. *AIAA Journal*, 14:1241–1249, 1976. doi: 10.2514/3.61457.

R. Warming, P. Kutler, and H. Lomax. Second-and third-order noncentered difference schemes for nonlinear hyperbolic equations. *AIAA Journal*, 11(2):189–196, 1973. doi: 10.2514/3.50449.

D. C. Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1993.

M. V. Zagarola and A. J. Smits. Mean-flow scaling of turbulent pipe flow. *Journal of Fluid Mechanics*, 373:33–79, 1998. doi: 10.1017/S0022112098002419.

Appendix A

Second order RANS turbulence modelling

A.1 Limitations of first order closure

The Boussinesq hypothesis is presented in section 2.6.1. The pros of models based on it include easy implementation, few differential equations to solve and a wide base of knowledge about their performance in various situations. It is however based on two assumptions, which are not always correct.

An example where the first one, named the intrinsic assumption, fails, is given by Pope [2000]. Two experiments by Tucker and Reynolds [1968] and Warhaft [1980] are compiled into one figure, shown below. The first picture in figure A.1 shows the experimental setup annotated with the strain rates:

$$\overline{S}_{ij} = \frac{1}{2}(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}) \tag{A.1}$$

In the first straight section, the mean axial velocity ($\overline{U}_1$) is ideally uniform, leading to a zero mean strain rate. The following contraction is designed to enforce a uniform axial strain rate, $\overline{S}_{11} = \frac{\partial \overline{U}_1}{\partial x} = \mathcal{S}_\lambda$, with the lateral strain rates $\overline{S}_{33}$ and $\overline{S}_{33}$ being $-\frac{1}{2}\mathcal{S}_\lambda$. Back in the second straight section, with the same simplifications as for the first, the mean strain rate is back to zero.

Beneath it are measured anisotropies, here defined as $b_{ij} \equiv \frac{1}{2k}a_{ij}$, on two different timescales. Recall that $a_{ij} \equiv <u_i'u_j'> - \frac{2}{3}k\delta_{ij}$. $\mathcal{S}_\lambda t$ is the flight time after the contrac-

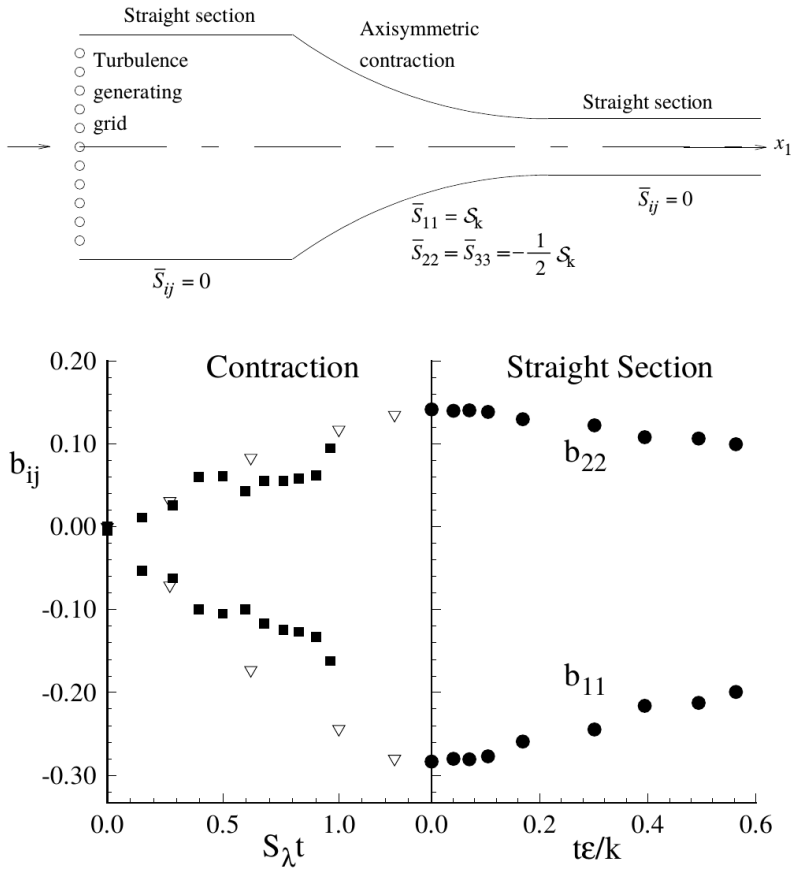

Figure A.1: Example of failure of the Boussinesq hypothesis, taken from Pope [2000].

tion starts normalized by the mean strain rate, giving a dimensionless quantity. $t\varepsilon/k$ is flight time after the straight section starts, this time made dimensionless by the turbulent timescale k/ϵ.

The intrinsic assumption states that the anisotropies scale with the local mean flow gradients, represented by the strain rate, see equation 2.50. This is visibly wrong here, as the anisotropies go to zero much slower than strain rate after the contraction ends. They exist in the straight section, not because of local gradients, but as a result of history effects on the flow. This history effect is neglected when using the Boussinesq hypothesis, as only local strain is accounted for. To remedy this, transport equations for the Reynolds stresses need to be added.

A rule of thumb given by Durbin and Reif [2011] states that the Boussinesq hypothesis performs well when the ratio $\mathcal{P}/\epsilon$ is close to unity. For the experiment in figure A.1 the ratio is high ($\mathcal{O}(10)$) in the contraction and zero in the straight section, explaining some of the unfavorable approximations made by the hypothesis.

A.2 Second order closure

Also known as Reynolds stress transport models, this category of models is available if one wants to avoid the pitfalls of the Boussinesq hypothesis. One extra level of closure is added by solving the Reynolds stress transport equations, 2.39. Going to second order closure makes the production tensor closed, as it contains only the dependent variable $<u_i'u_j'>$. This leaves ε_{ij}, $\mathcal{T}_{ijk}$ and $\mathcal{R}_{ij}$ to be modelled.

The dissipation tensor is modelled as $\varepsilon_{ij} = \frac{2}{3}\epsilon\delta_{ij}$, based on observations of it turning isotropic in high Reynolds number flow (Spalart [1988]). Special treatment is needed close to walls, as the local Reynolds number here is lower and the dissipation tensor anisotropy increases.

For the turbulent transport, it is convenient to split it into three parts:

$$\mathcal{T}_{kij} = \underbrace{\overline{u_i'u_j'u_k'}}_{\mathcal{T}_{kij}^{(u)}} + \underbrace{\overline{p'u_j'}\delta_{ik} + \overline{p'u_i'}\delta_{jk}}_{\mathcal{T}_{kij}^{(p)}} - \nu\frac{\partial\overline{u_i'u_j'}}{\partial x_k}. \tag{A.2}$$

$\mathcal{T}_{kij}^{(u)}$ is a triple correlation term, and is shown in experiments to dominate the turbulent transport (Bell and Mehta [1990]). $\mathcal{T}_{kij}^{(p)}$ is the transport done by the pressure. The last term is closed. Details of the modelling of the two first terms are different between different models, but a common method is to lump them together and use a gradient diffusion model (Durbin and Reif [2011]).

Lastly and most importantly, $\mathcal{R}_{ij}$ must be modelled. This term has a zero trace, and does therefore not contribute to the turbulent kinetic energy. It is only of redistributive character, taking energy from one velocity component to another through pressure fluctuations. To model it, the pressure is split into a harmonic, a rapid and a slow part (Pope [2000]):

$$\nabla^2 p^{(harmonic)} = 0 \tag{A.3}$$

$$\nabla^2 p^{(rapid)} = -2\rho\frac{\partial <u_i>}{\partial x_j}\frac{\partial u_j'}{\partial x_i} \tag{A.4}$$

$$\nabla^2 p^{(slow)} = -\rho\frac{\partial^2}{\partial x_i \partial x_j}(u_i u_j - <u_i'u_j'>) \tag{A.5}$$

These are then put in for p' in the equation for $\mathcal{R}_{ij}$, 2.42, leading to:

$$\mathcal{R}_{ij} = R_{ij}^{(h)} + R_{ij}^{(r)} + R_{ij}^{(s)} \tag{A.6}$$

How these three terms are modelled makes up the main differences between different Reynolds stress transport models.

A.2.1 The Launder, Reece and Rodi (LRR) model

As mentioned above, the modeling of the pressure strain correlation is the central issue in Reynolds stress transport models. The basis of most models for this comes from Rotta [1951], and one of the earliest models to come out of it was made by Launder et al. [1975]. Only the central equations are presented here, with more thorough explanations given in chapter 7.1 of Durbin and Reif [2011] and chapter 11 of Pope [2000]. The pressure-strain tensor 2.42 is rewritten to a redistribution tensor:

$$\Pi_{ij} = \mathcal{R}_{ij} - \frac{1}{3}\mathcal{R}_{kk}\delta_{ij} \tag{A.7}$$

The advantage of this formulation is that it disappears at no-slip walls. LRR then models it as:

$$\begin{aligned}\Pi_{ij} = &- C_1 \varepsilon b_{ij} \\ &+ 0.8 k S_{ij} \\ &+ \frac{18C_2' + 12}{11} k (b_{ik}S_{jk} + b_{jk}S_{ik} - \frac{2}{3} b_{mn}S_{mn}\delta_{ij}) \\ &+ \frac{20 - 14C_2'}{11} k (b_{ik}W_{jk} + b_{jk}W_{ik})\end{aligned} \tag{A.8}$$

The only new term here is:

$$W_{ij} \equiv \frac{u_{i,j} - u_{j,i}}{2} \tag{A.9}$$

Dissipation is modelled almost identically as in the k-epsilon model 2.55. Production is evaluated directly, with all the variables being given. The diffusion term is allowed to be anisotropic, as it is when approaching walls (Pope [2000]. Its differential equation reads:

$$\frac{\overline{D}\varepsilon}{\overline{D}t} = \frac{\partial}{\partial x_i}(C_\epsilon \frac{k}{\epsilon} <u_i' u_j'> \frac{\partial \epsilon}{\partial x_j}) + C_{\epsilon 1}\frac{\mathcal{P}\epsilon}{k} - C_{\epsilon 2}\frac{\epsilon^2}{k}. \tag{A.10}$$

In addition to this, some extra care is taken close to the walls. More about that in section 2.6.2. The constants have gone through some change over time. Launder et al. [1975] sets

$C_2' = 0.4$, Shabbir and Shih [1992] uses 0.55 while OpenCFD [2018a] uses 0.6. Similar stories go for C_1. The full list of constants used by OpenCFD [2018a] is:

$$\begin{aligned} C_\mu &= 0.09 & C_{\varepsilon 1} &= 1.44 & C_{\epsilon 2} &= 1.92 & C_1 &= 3.6 \\ C_2' &= 0.6 & C_s &= 0.25 & C_\epsilon &= 0.15 & C_{ref1} &= 0.5 \\ C_{ref2} &= 0.3 \end{aligned} \tag{A.11}$$

The LRR model is the most general model based on the constraints of Rotta [1951] (Shabbir and Shih [1992]). Other models, like SSG and LRR-QI, have more involved differential equations tuned for special phenomenons. This does, however, not always lead to better results. Shabbir and Shih [1992] found better performance by LRR than the SSG and LRR-IP model in thirteen flow configurations. Some hybrid models are also available, drawing from the positives of both the LRR and the SSG models Dudek and Carlson [2017].

A.2.2 The LRR model in OpenFOAM

OpenFOAM has two second order RANS models available; Speziale, Sarkar and Gaski (SSG) and the aforementioned LRR model. The latter is simpler to implement and make converge in OpenFOAM (Karvinen and Ahlstedt [2008]).

Using the LRR model in OpenFOAM requires setting the keyword RASModel to LRR. In addition to the files used by the k-epsilon model, an initial guess of the Reynolds stress field is necessary. This can be accomplished by running the command line argument **postProcess -func R** on an earlier case, which makes a file named R in all the time folders. Both the flux of the Reynolds stress and the stress itself needs an entry in the *fvSchemes* file.

Attempts at using the LRR model

It was planned to do additional tests of how accelerated and decelerated flow entering a lab tunnel affects head loss, due to the connecting pipe being either larger of smaller than the tunnel itself. To get as accurate results as possible it was desirable to use a Reynolds stress transport model, for the reasons depicted in section A.1. Getting the model to converge proved to be problematic. To prove its accuracy, it was attempted to remake the head loss and velocity profile data from Garcia [2017], as was done in Mølmann [2017] for the k-epsilon model. Using the refineHexMesh utility of OpenFOAM on the mesh used for the

k-epsilon models, a base size of 1.25 mm for the whole mesh was reached, including additional refinements at the walls. After mapping the converged k-epsilon results using mapFields and calculating the Reynolds stress fields, it was tried to run simpleFoam with the LRR model. It promptly exploded. Changing the schemes used for Reynolds stress and its flux to the linear and upwind scheme respectively made the residuals stabilize at 10^{-4}, after which the solution started to oscillate. Further changing the scheme for the velocity flux to a first order upwind scheme removed the oscillations and made the solution converge. In the process, the underrelaxation factor was set to 0.2 for both the pressure and Reynolds stress fields, making the convergence gruelingly slow.

Being both time consuming and hard to stabilize, it was decided to not go further in the pursuit of using the LRR model. When the original goal of increased precision could only be met by decreasing the accuracy of the schemes used, the endevour seemed to defeat its own purpose.

Appendix B

Additional figures and tables

B.1 OpenFOAM files

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

castellatedMesh true;
snap            true;
addLayers       false;

geometry
{
    tunnel.stl
    {
        type triSurfaceMesh;
        regions
        {
                outlet  { name outlet;}
                inlet   { name inlet;}
                walls   { name walls;}
                strips  { name strips;}
        }
    }

    measure1
    {
        type searchableBox;
        min (0.1425 -0.0625 -0.005);
        max ( 0.1475  0.0625 0.125);
    }

        ...

    measure35
    {
        type searchableBox;
        min (3.5425 -0.0625 -0.005);
        max ( 3.5475  0.0625 0.125);
    }

};
```

```
castellatedMeshControls
{
        maxLocalCells 100000000;
        maxGlobalCells 800000000;
        minRefinementCells 0;
        maxLoadUnbalance 0.20;
        nCellsBetweenLevels 5;
        resolveFeatureAngle 25;
        locationInMesh (0.0002 0.0002 0.0002);
        allowFreeStandingZoneFaces false;

    features
    (
        {
                file "tunnel.eMesh";
                level 1;
        }
    );

        refinementSurfaces
        {
        tunnel.stl
        { level (0 0);
                regions
                {
                walls   {level (1 1); patchInfo {type wall; }}
                strips  {level (2 2); patchInfo {type wall; }}
                inlet   {level (0 0); patchInfo {type patch; }}
                outlet  {level (0 0); patchInfo {type patch; }}
                }
        }

        measure1
        {
            level (0 0);

            faceZone    faceZone1;
            cellZone    c1;
            cellZoneInside inside;

            boundary    internal;
        }
```

```
        ...

        measure35
        {
            level (0 0);

            faceZone    faceZone35;
            cellZone    c35;
            cellZoneInside inside;

            boundary    internal;
        }
    }

         refinementRegions
        {
        }
}

snapControls
{
    nSmoothPatch 3;
    tolerance 1.8;
    nSolveIter 300;
    nRelaxIter 30;
    nFeatureSnapIter 30;
    implicitFeatureSnap true;
    explicitFeatureSnap true;
    multiRegionFeatureSnap true;
}


addLayersControls
{
}

meshQualityControls
{
    #include "meshQualityDict"
    relaxed
    {
        maxNonOrtho 65;
    }

    nSmoothScale 4;
    errorReduction 0.75;
}

mergeTolerance 1e-6;
```

Figure B.1: ***snappyHexMeshDict*** **file used for the meshing of all shape test geometries.**

```
Exec   : checkMesh -parallel -latestTime
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Time = 2

Mesh stats
        ...
    cells:            698786

Overall number of cells of each type:
    hexahedra:     680986
        ....

Checking topology...
    Boundary definition OK.
        ...

Checking geometry...
        ...
    Max aspect ratio = 5.8014 OK.
    Min volume = 3.26177e-10. Max volume = 3.59647e-08.
    Mesh non-orthogonality Max: 35 average: 3.73098
    Max skewness = 2.30055 OK.
        ...
Mesh OK.

End
```

Figure B.2: Example output from checkMesh.

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    p{  solver           GAMG;
        tolerance        1e-6;
        relTol           0.01;
        smoother         GaussSeidel;
        nPreSweeps       0;
        ...     }

    U{  solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-08;
        relTol          0.002;
        }

    k           {...}
    epsilon     {...}
}
SIMPLE
{   nNonOrthogonalCorrectors 2;
    pRefCell            0;
    pRefValue           0;
   residualControl{
        p               1e-3;
        U               1e-3;
        nuTilda         1e-3;
        k               1e-3;
        epsilon         1e-3;
    }
}

relaxationFactors
{
    fields{     p       0.3;}

    equations{  U       0.7;
                k       0.7;
                nut     0.7;
                epsilon 0.7;}
}
```

Figure B.3: ***fvSolution*** **file used for my k-epsilon simulations.**

```
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "include/initialConditions"
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform $flowVelocity;

boundaryField
{
    walls       {type           noSlip;}

    inlet       {type           fixedValue;
                value           $internalField;}

    outlet      {type           inletOutlet;
                inletValue      $internalField;
                value           $internalField;}
}
```

Figure B.4: U file used for my simulations.

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;

boundaryField
{
    walls       {type           zeroGradient;}

    inlet       {type           zeroGradient;}

    outlet      {type           fixedValue;
                value           $internalField;}
}
```

Figure B.5: p file used for my simulations.

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "include/initialConditions"
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform $turbulentKE;

boundaryField
{
    walls       {type          kqRWallFunction;
                 value         $internalField;}

    inlet       {type           fixedValue;
                 value          $internalField;}

    outlet      {type         inletOutlet;
                inletValue   $internalField;
                value        $internalField;}
}
```

Figure B.6: *k* file used for my simulations.

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "include/initialConditions"
dimensions      [0 2 -3 0 0 0 0];
internalField   uniform $epsilon;

boundaryField
{
    walls       {type           epsilonWallFunction;
                value           $internalField;}

    inlet       {type           fixedValue;
                value           $internalField;}

    outlet      {type           zeroGradient;}
}
```

Figure B.7: *epsilon* file used for my simulations.

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "include/initialConditions"
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform $nut;

boundaryField
{
    walls       {type          nutkWallFunction;
                 value         $internalField;}

    inlet       {type           calculated;
                 value          $internalField;}

    outlet      {type         zeroGradient;}
}
```

Figure B.8: ***nut*** **file used for my simulations.**

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType RAS;

RAS
{
    RASModel        kEpsilon;
    turbulence      on;
    printCoeffs     on;
}
```

Figure B.9: ***turbulenceProperties*** **file used for my simulations.**

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
transportModel  Newtonian;

nu              [0 2 -1 0 0 0 0] 1e-06;
```

Figure B.10: ***transportProperties*** **file used for my simulations.**

```
sample
{
type                            sets;
libs                            ("libsampling.so");
writeControl                    writeTime;
interpolationScheme             cellPointFace;
setFormat                       gnuplot;

sets
(

profile1{       type            midPoint;
                axis            z;
                start           (0.14075 0.00075 -0.001);
                end             (0.14075 0.00075 0.165);}

   ...

profile36{      type            midPoint;
                axis            z;
                start           (3.64075 0.00075 -0.001);
                end             (3.64075 0.00075 0.165);}
);

fields                          (U);

}
```

Figure B.11: The file controlling sampling of velocity profiles.

```
HL1
{
    type                fieldValueDelta;
    libs                ("libfieldFunctionObjects.so");
    writeControl        writeTime;
    writeFormat         ascii;
    operation           subtract;
    region1
    {
        type            volFieldValue;
        libs            ("libutilityFunctionObjects.so");
        log             false;
        writeControl    writeTime;
        writeFields     false;
        regionType      cellZone;
        name            c1;
        operation       volAverage;
        fields(         total(p));

    }

    region2
    {
        ...
    }
}
```

Figure B.12: The *fieldValueDelta* file, measuring head loss between two cell zones.

```
streamLines
{
    libs                ("libfieldFunctionObjects.so");
    type                streamLine;
    writeControl        writeTime;
    setFormat           vtk;
    trackForward        true;
    fields              (p U k);
    lifeTime            20000;
    nSubCycle           3;
    cloudName           particleTracks;

    seedSampleSet
    {
    type                uniform;
    axis                y;
    start               (0.0005 -0.049 0.0805);
    end                 (0.0005 0.049  0.0805);
    nPoints             40;
    }
}
```

Figure B.13: The file seeding particles for streamlines.

```
cuttingPlane
{
    type                        surfaces;
    libs                        ("libsampling.so");
    writeControl                writeTime;
    surfaceFormat               vtk;
    fields                      ( U );
    interpolationScheme         cellPoint;
    start                       (0.0005 -0.048 -0.0105);

    surfaces
    (
        xNormal
        {
            type                cuttingPlane;
            planeType           pointAndNormal;
            pointAndNormalDict
            {
                point           (0.15075 0 0);
                normal          (1 0 0);
            }

            interpolate         true;
        }
        zNormal
        {
            type                cuttingPlane;
            planeType           pointAndNormal;
            pointAndNormalDict
            {
                point           (0 0 -0.015);
                normal          (0 0 1);
            }

            interpolate         true;
        }
    );
}
```

Figure B.14: The file guiding the creation of cutting planes.

```
set terminal postscript eps enhanced size 4.5in,3in color
set output "percentageZoom.eps"
set xlabel 'Strip #'
set ylabel 'Change in head loss'
set xrange [6.5:17.5]
set yrange [0:5]
set mxtics 2
set mytics 1
set ytics add ("0\%%" 0, "1\%%" 1, "2\%%" 2, "3\%%" 3, "4\%%" 4, "5\%%" 5)
set key right top Left box 1

## Grid and border style

set style line 11 lc rgb '#808080' lt 1
set border 3 back ls 11
set tics nomirror
set style line 12 lc rgb '#808080' lt 3 lw 0.7
set grid back ls 12

## Line styles

set style line 1 lc rgb '#e41a1c' pt 4 ps 1 lt 1 lw 2 # red
set style line 2 lc rgb '#377eb8' pt 4 ps 1 lt 1 lw 2 # blue
set style line 3 lc rgb '#4daf4a' pt 4 ps 1 lt 1 lw 2 # green
set style line 4 lc rgb '#984ea3' pt 4 ps 1 lt 1 lw 2 # purple



plot    "circle/HeadLoss.csv" using 1:3 title "Circle" with lp ls 1,\
        "tunnel/HeadLoss.csv" using 1:3 title "Normal tunnel" with lp ls 2,\
        "strange/HeadLoss.csv" using 1:3 title "Birkeland" with lp ls 3,\
        "square/HeadLoss.csv" using 1:3 title "Square" with lp ls 4
```

Figure B.15: The gnuplot script used to make figure B.16.

B.2 Shape test

Geometries

Radius	0.05 m
Circumference	0.3142 m
Area	0.0079 m^2
Hydraulic radius	0.025 m

Table B.1: Circle geometry.

Radius (R)	0.06 m
Theta (Θ)	2.23°
Height (h)	0.0329 m
Width (c)	0.107 m
Area	0.0088 m^2
Circumference	0.352 m
Hydraulic radius	0.025 m

Table B.2: Birkeland geometry. Made by cutting of a circle; Θ is the angle of the cut to the horizontal.

Height	0.05 m
Width	0.1 m
Area	0.0089 m^2
Circumference	0.357 m
Hydraulic radius	0.025 m

Table B.3: Normal tunnel geometry. Made by adding a half circle with radius equalling half the width to a rectangle.

Height and width	0.1 m
Area	0.01 m^2
Circumference	0.4 m
Hydraulic radius	0.025 m

Table B.4: Square geometry.

Head loss

The grey area in the development graphs indicates where the flow is considered developed.

Strip CC	Velocity	Geometry	y^+					
			Strip			Walls		
			Min	Max	Mean	Min	Max	Mean
10 cm	1 m/s	Circle	1	68	21	1	73	26
		Square	1	54	23	1	63	29
		Birkeland	1	68	22	1	82	27
		Normal tunnel	1	69	22	1	72	28
	2 m/s	Circle	6	136	41	2	145	51
		Square	2	107	46	3	126	58
		Birkeland	5	136	43	2	164	53
		Normal tunnel	4	137	44	2	142	55
	4 m/s	Circle	9	275	83	2	292	101
		Square	9	215	91	9	264	116
		Birkeland	10	271	87	3	324	106
		Normal tunnel	9	274	88	2	282	109
15 cm	1 m/s	Circle	-	-	-	3	118	31
		Square	-	-	-	7	93	35
		Birkeland	-	-	-	2	120	33
		Normal tunnel	-	-	-	2	117	34
	2 m/s	Circle	-	-	-	6	237	63
		Square	-	-	-	10	183	69
		Birkeland	-	-	-	6	233	66
		Normal tunnel	-	-	-	6	237	67
	4 m/s	Circle	-	-	-	11	476	125
		Square	-	-	-	18	368	137
		Birkeland	-	-	-	10	465	130
		Normal tunnel	-	-	-	9	450	128
20 cm	1 m/s	Circle	1	62	20	1	66	26
		Square	1	48	23	1	61	29
		Birkeland	1	62	22	1	79	27
		Normal tunnel	1	62	22	1	65	27
	2 m/s	Circle	6	123	40	2	132	51
		Square	7	97	45	7	113	58
		Birkeland	6	124	43	1	159	53
		Normal tunnel	6	124	43	1	130	55
	4 m/s	Circle	10	251	81	2	266	101
		Square	10	194	90	10	226	114
		Birkeland	10	249	85	2	313	106
		Normal tunnel	9	200	87	3	296	105

Table B.5: y^+-values for all meshes and velocities. Note that the strip and wall patches were fused for the 15 cm geometries, thus coming out as one.

Table B.6: Head loss over 60 cm with computed friction factors for all geometries.

Strip CC	U	Geometry	Head loss [mH2O]	Friction factor
10 cm	**1 m/s**	Circle	0.056	0.183
		Birkeland	0.054	0.176
		Tunnel	0.053	0.173
		Square	0.050	0.163
	2 m/s	Circle	0.224	0.183
		Birkeland	0.215	0.176
		Tunnel	0.211	0.173
		Square	0.200	0.163
	4 m/s	Circle	0.912	0.186
		Birkeland	0.873	0.178
		Tunnel	0.845	0.173
		Square	0.799	0.163
15 cm	**1 m/s**	Circle	0.044	0.143
		Birkeland	0.043	0.140
		Tunnel	0.042	0.137
		Square	0.040	0.131
	2 m/s	Circle	0.171	0.140
		Birkeland	0.171	0.140
		Tunnel	0.166	0.136
		Square	0.153	0.125
	4 m/s	Circle	0.685	0.140
		Birkeland	0.686	0.140
		Tunnel	0.666	0.136
		Square	0.607	0.124
20 cm	**1 m/s**	Circle	0.036	0.119
		Birkeland	0.035	0.115
		Tunnel	0.034	0.112
		Square	0.033	0.107
	2 m/s	Circle	0.145	0.119
		Birkeland	0.141	0.115
		Tunnel	0.137	0.112
		Square	0.131	0.107
	4 m/s	Circle	0.586	0.120
		Birkeland	0.566	0.116
		Tunnel	0.549	0.112
		Square	0.523	0.107

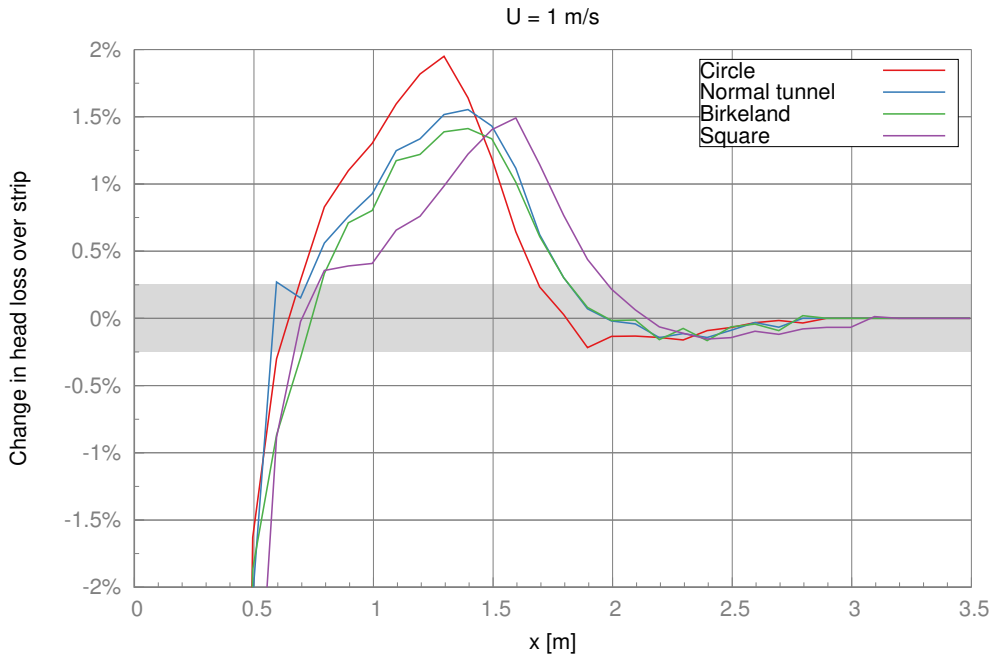

Figure B.16: Head loss change between strips for CC 10 cm, 1 m/s.

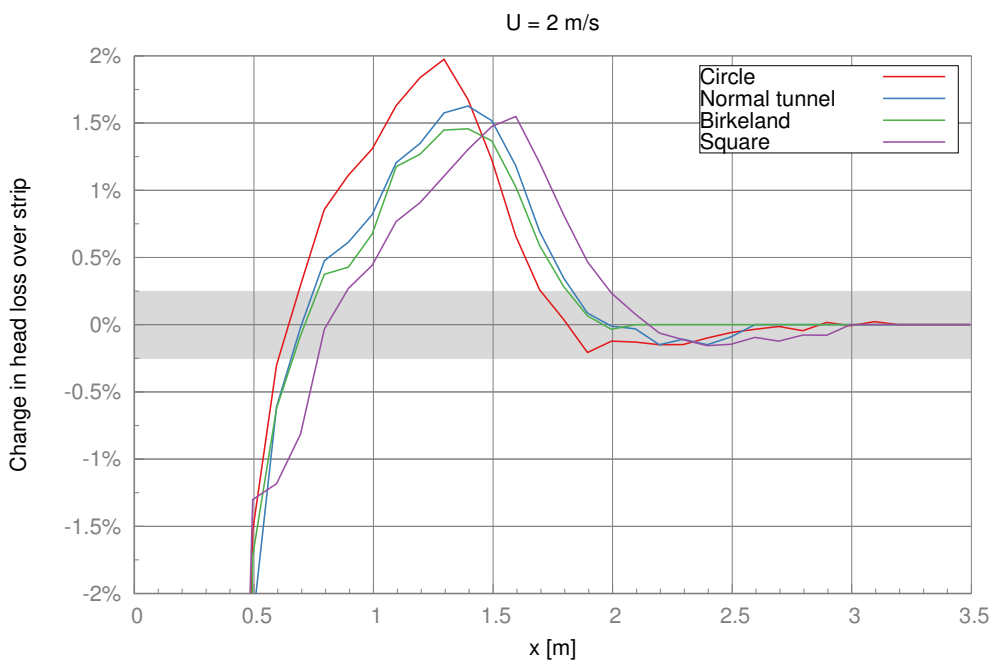

Figure B.17: Head loss change between strips for CC 10 cm, 2 m/s.

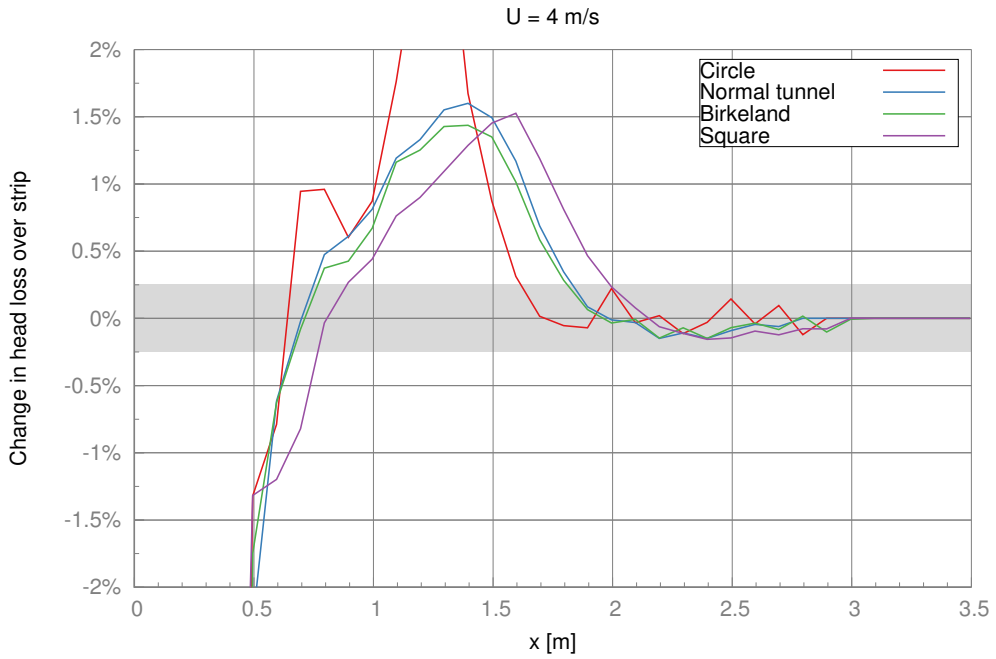

Figure B.18: Head loss change between strips for CC 10 cm, 4 m/s.

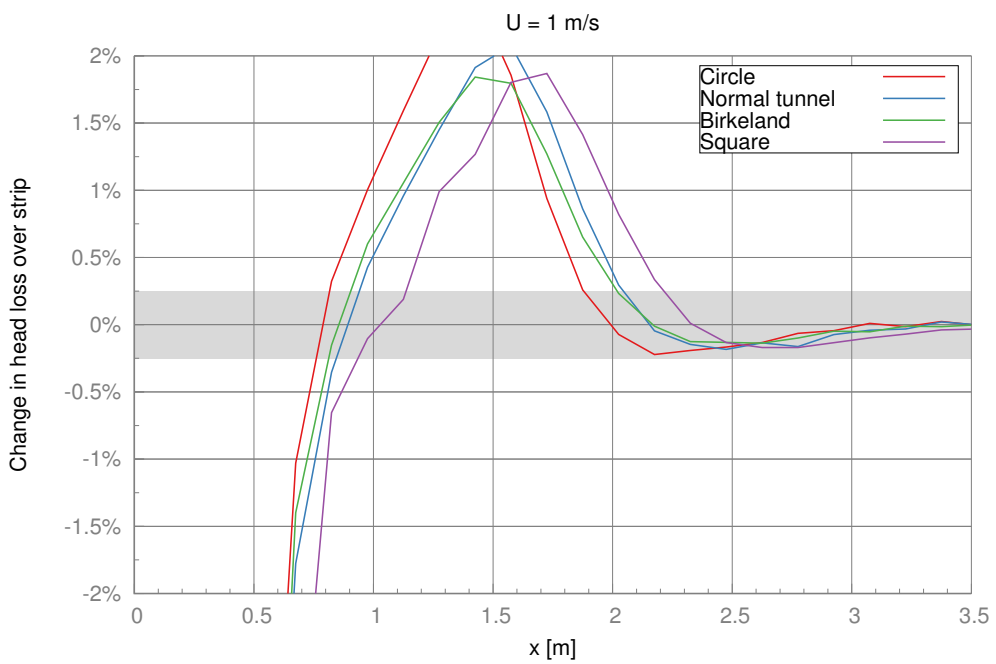

Figure B.19: Head loss change between strips for CC 15 cm, 1 m/s.

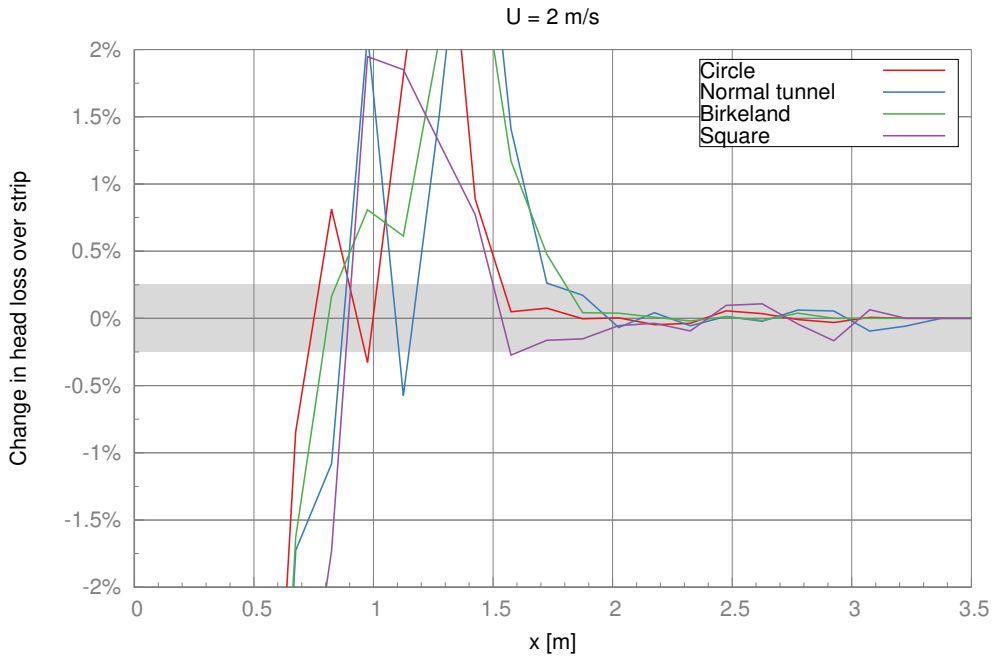

Figure B.20: Head loss change between strips for CC 15 cm, 2 m/s.

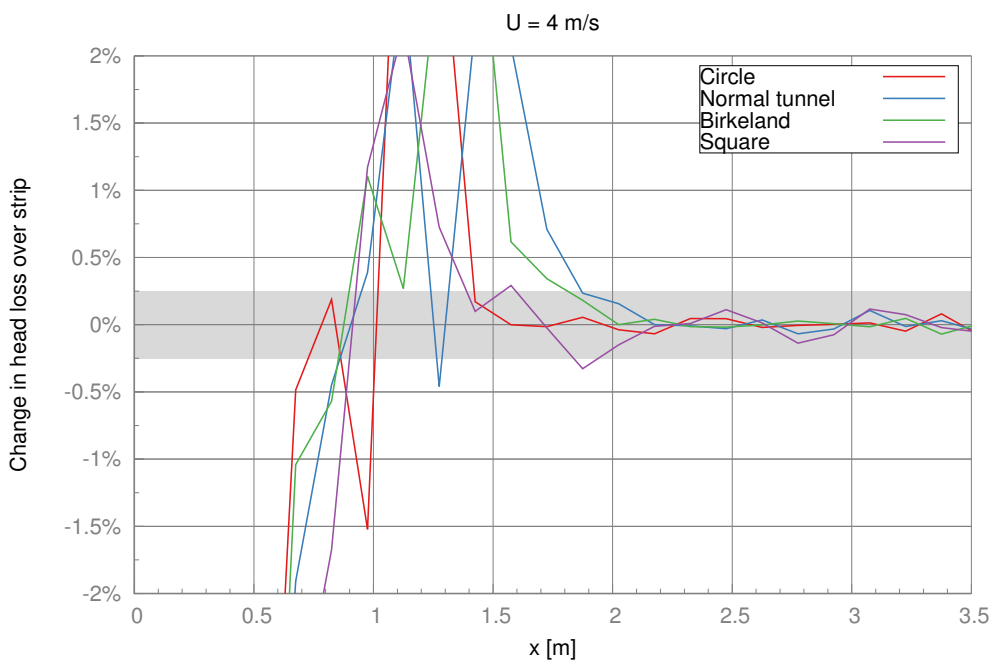

Figure B.21: Head loss change between strips for CC 15 cm, 4 m/s.

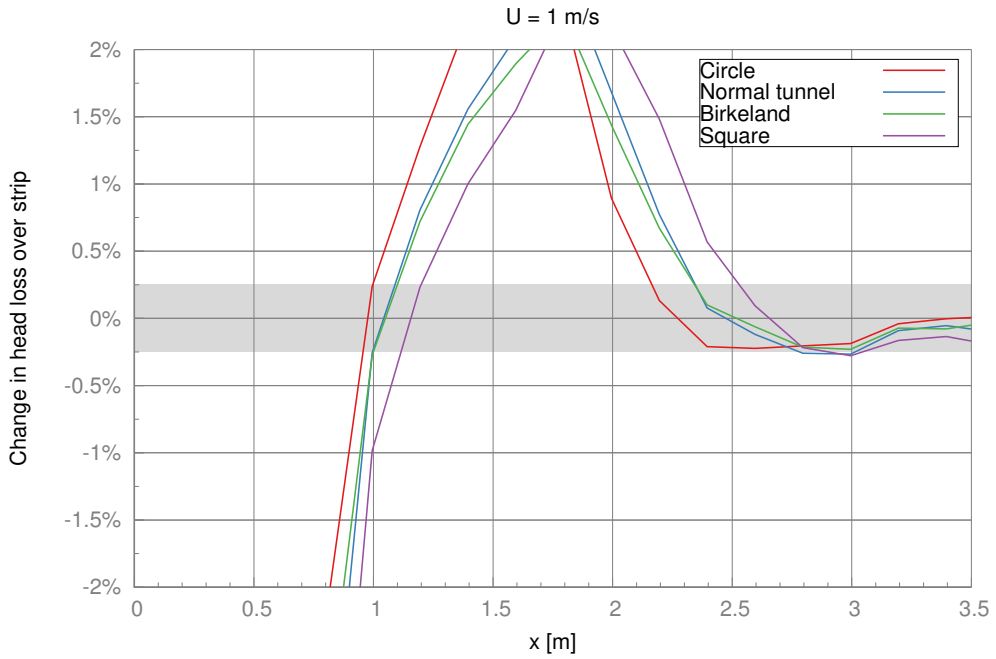

Figure B.22: Head loss change between strips for CC 20 cm, 1 m/s.

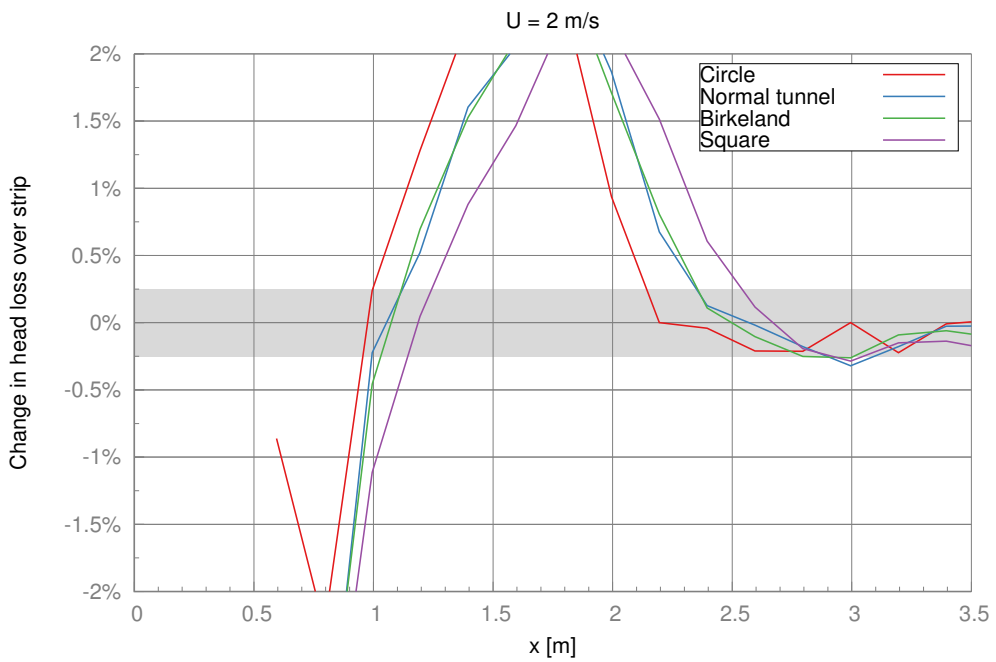

Figure B.23: Head loss change between strips for CC 20 cm, 2 m/s.

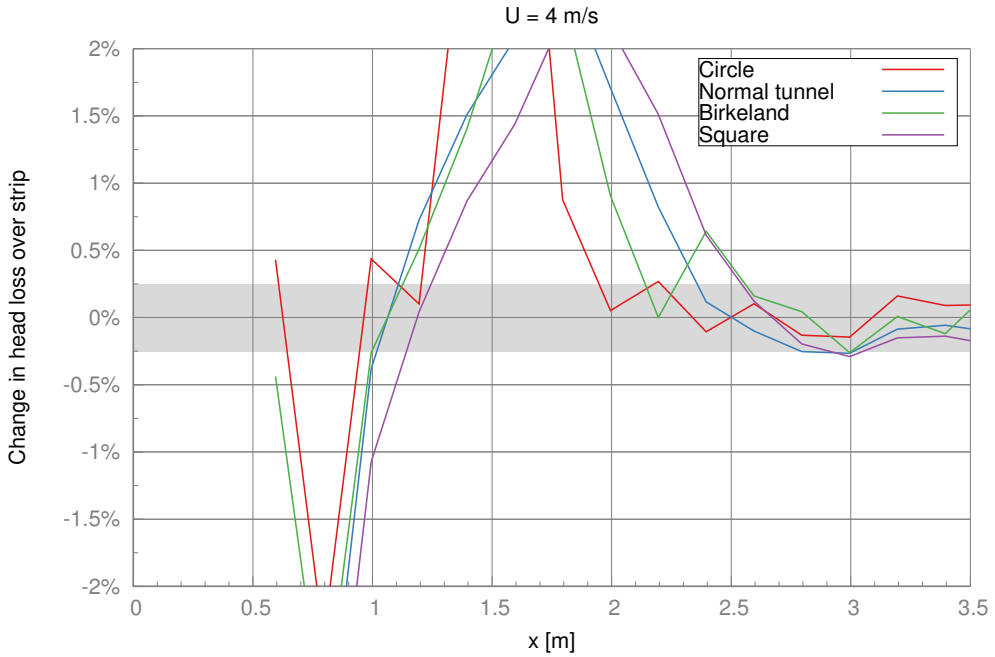

Figure B.24: Head loss change between strips for CC 20 cm, 4 m/s.