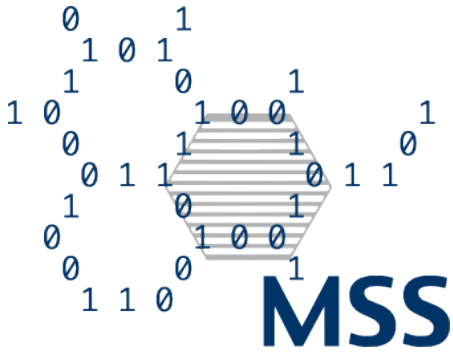

Institute for Multiscale Simulation of Particulate Systems
Friedrich-Alexander Universität Erlangen-Nürnberg
Prof. Thorsten Pöschel

BACHELOR THESIS

Validation of Dune Simulations using OpenFOAM

Britt Michelsen

Supervisors:
Dipl.-Inf Severin Strobl
Prof. Dr. Thorsten Pöschel

2012

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen habe und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 18.März 2012 Britt Michelsen

Abstract

In this thesis, the fluid flow around a two and three-dimensional dune is simulated with the help of the open source C++ library OpenFOAM[1]. The simpleFOAM solver, which is supplied by OpenFOAM, has proven to be reliable and has shown a strong correlation with field measurements conducted in the paper "*The role of streamline curvature in sand dune dynamics*" by Giles F. S. Wiggs in 1996. It can be shown, that the two-dimensional as well as the three-dimensional calculations are in good agreement with the experimental data.
However, in order to get reasonable steady-state results the initial conditions are calculated using the transient solver pimpleFOAM. The Reynolds-averaged Navier-Stokes equation is being used to simulate the time-averaged turbulent fluid flow around the dune and different meshing tools as well as cell geometries are compared.
The validation of a fluidized model as an alternative to modelling the individual sand particles has proven to be difficult, because sand doesn't show a conventional fluidic behaviour.

Im Rahmen dieser Arbeit wurde das Strömungsprofil sowohl um eine zweidimensionale als auch um ein dreidimensionale Düne mit Hilfe der Open-Source C++ Bibliothek OpenFOAM simuliert. Die simpleFOAM Simulationsroutine, welche von OpenFOAM zur Verfügung gestellt wird, stellte sich als verlässlich heraus und zeigte eine starke Relation zu den in "The role of streamline curvature in sand dune dynamics" veröffentlichten Werten von Giles F. S. Wiggs aus dem Jahre 1996. Sowohl der zwei- als auch der dreidimensionale Fall konnte in guter Übereinstimmung mit den Ergebnissen der Veröffentlichung simuliert werden.
Allerdings mussten die Anfangswerte mit Hilfe des transienten pimpleFOAM Gleichungslösers approximiert werden und dann anschließend in simpleFOAM übernommen werden. Die Reynolds-gemittelten Navier-Stokes-Geichungen wurden zur Approximation der turbulenten Strömungen verwendet und unterschiedliche Diskretisierungsgitter zur Validierung herangezogen.
Die Überprüfung der Gültigkeit von fluidisierten Modellen als Alternative zur Simulation einzelner Sandkörner stellte sich als kompliziert heraus, da Sand kein herkömmliches Fluidverhalten zeigt und somit nicht ausreichend vom Strömungslöser erfasst werden kann.

[1] http://www.openfoam.org/version2.1.0/

Table of Contents

1. Introduction

A dune is a hill of sand created by wind or water flow. While it can occur in many different shapes, this work is concentrating on crescent-shaped barchans as shown in Illustration 1, which are formed under wind that blows continuously from one direction.

The studying of sand dunes has a long history and is for example part of Ralph Bagnold's work “The Physics Of Blown Sand And Desert Dunes” from 1941. His work is mainly speculative and written at a time when the turbulent fluid flow became known thanks to the work of Prandl (1935). While Bagnold concentrated on the movement of individual sand grains [1] other studies for example by Wilson in 1973, concentrated on the formation of groups of dunes. Since then the theories have been improved and compared to wind tunnel measurements.

This thesis concentrates on the works by Giles F.S. Wiggs from 1996 published in the paper “The role of streamline curvature in sand dune dynamics” to compare the simulation results. It is a field study of a single, about 10 meter high, unvegetated barchan dune in the Oman, compared to wind tunnel measurements over a 1:200 scale fixed model. He found “similar patterns of wind and shear velocity over the dune, confirming significant flow deceleration upwind of and at the toe of the dune, acceleration of flow up the windward slope, and deceleration between the crest and brink ”. According to him this reflects previous studies and this thesis tries to verify his results with the help of computational fluid dynamics using OpenFOAM.

Illustration 1: Air view of a barchan dune

2. Theory

2.1. Basic equations

In the following chapter a short overview of the governing transport equations as well as a brief introduction to numerical solution schemes will be given.

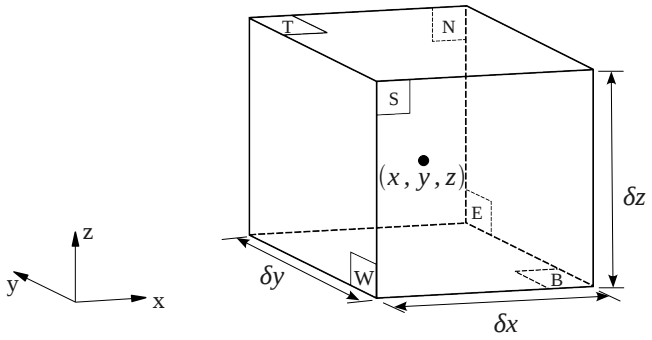

Illustration 2: Fluid element for conservation laws

Illustration 2 shows a small fluid element with a side length of δx, δy and δz. The fluid properties are given at a point, which represents the smallest possible element of fluid whose macroscopic properties are not influenced by individual molecules [2]. So the pressure, temperature, density and the velocity vector should be written as:

p(x, y, z, t), T(x, y, z, t), ρ(x, y, z, t), and **u**(x, y, z, t).

Further on to avoid this long-winded notation, the dependency on space and time will no longer be stated. The letters N, E, W and S are corresponding to the geographic directions and T and B are representing the top and the bottom. To define the flux over the cell faces of the fluid element, a Taylor series expansion around the cell centre is done, as shown exemplarily for the x direction in equations (1) for the density and velocity.

$$\begin{aligned}\rho_{E,t} &= \rho\left(x+\frac{1}{2}\delta x\right) = \rho + \frac{1}{1!}\frac{\partial \rho}{\partial x}\left(x+\frac{1}{2}\delta x - x\right) + \frac{1}{2!}\frac{\partial^2 \rho}{\partial x^2}\left(x+\frac{1}{2}\delta x - x\right) + \cdots \\ \boldsymbol{u}_{E,t} &= \boldsymbol{u}\left(x+\frac{1}{2}\delta x\right) = \boldsymbol{u} + \frac{1}{1!}\frac{\partial \boldsymbol{u}}{\partial x}\left(x+\frac{1}{2}\delta x - x\right) + \frac{1}{2!}\frac{\partial^2 \boldsymbol{u}}{\partial x^2}\left(x+\frac{1}{2}\delta x - x\right) + \cdots\end{aligned} \tag{1}$$

The solution can be expressed accurately enough, using the first two terms of the Taylor series, thus equations (1) can be simplified to equations (2).

$$\begin{aligned}\rho_{E,t} &= \rho + \frac{\partial \rho}{\partial x}\left(\frac{1}{2}\delta x\right) \\ \boldsymbol{u}_{E,t} &= \boldsymbol{u} + \frac{\partial \boldsymbol{u}}{\partial x}\left(\frac{1}{2}\delta x\right)\end{aligned} \tag{2}$$

The area A_E is given by its side lengths, δy and δz. To give a better understanding, the velocity vector **u** will be decomposed in its x, y and z components (u,v, w). Under consideration of these specifications, the mass flow rate for a face can be determined and is shown in equation (3).

$$\begin{aligned}\dot{m}_{E,t} &= \rho_{E,t} \cdot u_{E,t} \cdot A_E = \left[\rho + \frac{\partial \rho}{\partial x}\left(\frac{1}{2}\delta x\right)\right] \cdot \left[u + \frac{\partial u}{\partial x}\left(\frac{1}{2}\delta x\right)\right] \cdot \delta y \cdot \delta z \\ \dot{m}_{E,t} &= \left[\rho \cdot u + \rho \frac{\partial u}{\partial x}\left(\frac{1}{2}\delta x\right) + u \frac{\partial \rho}{\partial x}\left(\frac{1}{2}\delta x\right) + \frac{\partial \rho}{\partial x} \cdot \frac{\partial u}{\partial x}\left(\frac{1}{2}\delta x\right)^2\right] \cdot \delta y \cdot \delta z\end{aligned} \tag{3}$$

By neglecting $\frac{\partial \rho}{\partial x} \cdot \frac{\partial u}{\partial x}$ equation (3) can be simplified to equation (4).

$$\dot{m}_{E,t} = \left[\rho u + \frac{\partial (\rho u)}{\partial x}\left(\frac{1}{2}\delta x\right)\right] \cdot \delta y \cdot \delta z \tag{4}$$

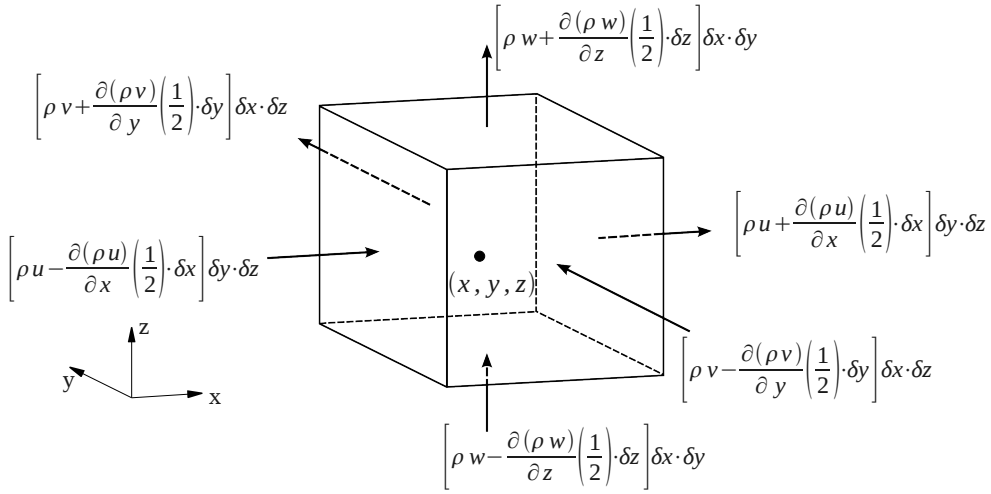

Illustration 3: Mass flows in and out of a fluid element

Illustration 3 shows the mass flow in and out of a fluid element, which is also displayed in equation (5).

$$\begin{aligned}
\frac{\partial m}{\partial t} &= m_W - m_E + m_S - m_N + m_B - m_T \\
\frac{\partial m}{\partial t} &= \left[\left(\rho u - \frac{\partial(\rho u)}{\partial x}\right) - \left(\rho u + \frac{\partial(\rho u)}{\partial x}\right)\right] \cdot [\delta y \cdot \delta z] \\
&+ \left[\left(\rho v - \frac{\partial(\rho v)}{\partial x}\right) - \left(\rho v + \frac{\partial(\rho v)}{\partial x}\right)\right] \cdot [\delta x \cdot \delta z] \\
&+ \left[\left(\rho w - \frac{\partial(\rho w)}{\partial x}\right) - \left(\rho w + \frac{\partial(\rho w)}{\partial x}\right)\right] \cdot [\delta x \cdot \delta y]
\end{aligned} \tag{5}$$

The result of expanding and solving Equation (5) is shown in Equation (6) and can be further simplified, as shown in Equations (7).

$$-\frac{\partial m}{\partial t} = \frac{\partial(\rho u)}{\partial x}[\delta x \cdot \delta y \cdot \delta z] + \frac{\partial(\rho v)}{\partial y}[\delta x \cdot \delta y \cdot \delta z] + \frac{\partial(\rho w)}{\partial z}[\delta x \cdot \delta y \cdot \delta z] \tag{6}$$

$$\frac{\partial m}{\partial t} = \frac{\partial(V\rho)}{\partial t} = V\frac{\partial \rho}{\partial t} + \rho\frac{\partial V}{\partial t} \quad \text{with} \quad V = [\delta x \cdot \delta y \cdot \delta z] = const. \quad \Rightarrow \frac{\partial m}{\partial t} = V\frac{\partial \rho}{\partial t}$$

$$-\frac{\partial m}{\partial t} = -[\delta x \cdot \delta y \cdot \delta z]\frac{\partial \rho}{\partial t} = \frac{\partial(\rho u)}{\partial x}[\delta x \cdot \delta y \cdot \delta z] + \frac{\partial(\rho v)}{\partial y}[\delta x \cdot \delta y \cdot \delta z] + \frac{\partial(\rho w)}{\partial z}[\delta x \cdot \delta y \cdot \delta z] \tag{7}$$

$$0 = \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} + \frac{\partial \rho}{\partial t}$$

In a more compact vector notation, Equation (7) can be written as shown in Equation (8). It is the unsteady, three-dimensional mass conservation or continuity equation at a point in a compressible fluid [2]. For an incompressible fluid the density is constant, so Equation (8) can be simplified to Equation (9) .

$$\frac{\partial \rho}{\partial t} + div(\rho \boldsymbol{u}) = 0 \tag{8}$$

$$\text{for} \quad \frac{\partial \rho}{\partial t} = 0: \quad div(\rho \boldsymbol{u}) = \rho\, div(\boldsymbol{u})$$

$$\Rightarrow div\ \boldsymbol{u} = 0 \tag{9}$$

The momentum equation can be derived in the same way. The derivation is due to the nine viscous stress components (τ_{ij} with $i, j = x, y, z$ – every time the viscous stress in j direction, on a plane, vertical to i, as shown in Illustration 4) and pressure, quite long and won't be part of this thesis. The pressure is a normal stress and denoted with p.

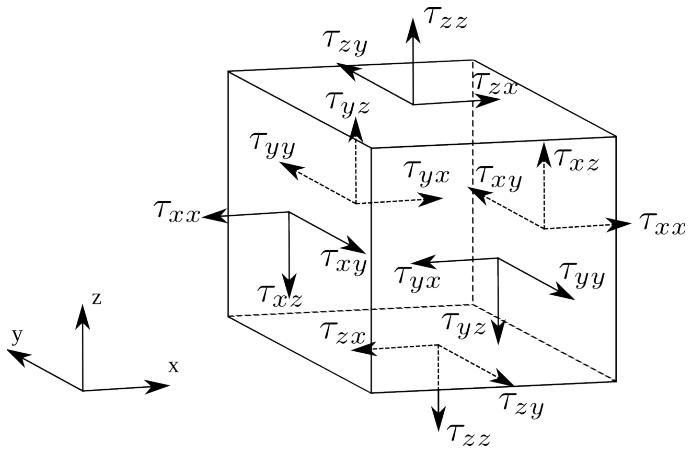

Illustration 4: stress components on three faces of a fluid element

Equation (10) shows the three dimensional momentum equation exemplarily for the x direction.

$$\rho \frac{Du}{Dt} = \frac{\partial(-p+\tau_{xx})}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + S_{Mx} \tag{10}$$

S_{Mx} represents the source terms, that are of volumetric nature, like e.g. the weight or electromagnetic forces. For Newtonian fluids, with a linear correlation between the viscous stress τ and linear or volumetric deformation, the Navier-Stokes equations can be derived. Due to the fact that it is the base for all further calculations, it will be shown here briefly. Under consideration of the linear correlation, the momentum equation for the x direction can be written as shown in Equation (11).

$$\rho \frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left(2\mu \frac{\partial u}{\partial x} + \lambda \, div\, \boldsymbol{u}\right) + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial z}\left[\mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)\right] + S_{Mx} \tag{11}$$

The dynamic viscosity μ relates stresses to linear deformation and the kinetic viscosity relates stresses to the volumetric transformation. The latter is approximated to $\lambda = -\frac{2}{3}\mu$ [3]. In (incompressible) fluids, volumetric deformation doesn't occur. By adding the less influential terms to the momentum source term, Equation (11) can be simplified as shown in Equation (12).

$$\frac{\partial}{\partial x}\left(2\mu\frac{\partial u}{\partial x}+\lambda\, div\,\boldsymbol{u}\right)+\frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]+\frac{\partial}{\partial z}\left[\mu\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)\right]+S_{Mx}=div(\mu\, grad\, u)+S'_{Mx}$$
$$S'_{Mx}=S_{Mx}+\frac{\partial}{\partial x}(\lambda\, div\,\boldsymbol{u})+\left[\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial z}\left(\mu\frac{\partial w}{\partial x}\right)\right] \tag{12}$$

Applying the conservation of mass allows to write the term $\rho\frac{Du}{Dt}$ as $\frac{\partial(\rho u)}{\partial t}+div(\rho u\boldsymbol{u})$ and therefore the Navier-Stokes equation as shown in equation (13).

$$\rho\frac{\partial u}{\partial t}+div(\rho u\boldsymbol{u})=div(\mu\, grad\, u)+\left(S'_{Mx}-\frac{\partial p}{\partial x}\right) \tag{13}$$

It is the so-called transport equation and and can be used to describe all fluid dynamic equations. Equation (14) shows it in a more general form. The pressure term is independent of the parameter Φ and can therefore be added to the source term.

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{Accumulation}+\underbrace{div(\rho\phi\boldsymbol{u})}_{Convection}=\underbrace{div(T\, grad\,\phi)}_{Diffusion}+\underbrace{S_\phi}_{Source} \tag{14}$$

So if further the diffusion coefficient T is mentioned, it doesn't necessarily mean, that it is a fluid-fluid diffusion coefficient. It can depending on the parameter Φ, either be the viscosity μ (in case of the momentum equation) or the thermal conductivity (in case of the energy equation). This has the huge advantage, that all parameters can be treated in the same way during the discretization process.

2.2. The finite volume method

After deriving the necessary equations, they still need to be discreticed for numerical solution. So far the equations have been stated in their differential form and can be solved by integrating over a finite control volume. This is the main feature of the finite volume method. Equation (15) shows Equation (14) in its integral form.

$$\int_{CV} \frac{\partial(\rho\phi)}{\partial t} dV + \int_{CV} div(\rho\phi\boldsymbol{u}) dV = \int_{CV} div(T\, grad\,\phi)\, dV + \int_{CV} S_\phi\, dV \tag{15}$$

By using Gauss' divergence theorem, the terms can be rewritten as integrals over the entire bounding surface of the control volume. This means, that the accumulation of a component in a volume will be written as the flow through the bounding surface of the control volume, as shown in Equation (16).

$$\int_{CV} div(\boldsymbol{v}) dV = \int_{A} \boldsymbol{n}\cdot\boldsymbol{v}\, dA \tag{16}$$

$\boldsymbol{n}\cdot\boldsymbol{v}$ describes the component of vector **v** in the direction of the vector **n** normal to the surface element dA. By meshing a fluid (dividing it into control volumes of known size and shape), the volume integrals can be easily solved by summation over the surface elements. Equation (17) shows the result of applying Gauss divergence theorem to equation (15).

$$\frac{\partial}{\partial t}\left(\int_{CV} \rho\phi\, dV\right) + \int_{A} \boldsymbol{n}\cdot(\rho\phi\boldsymbol{u})\, dA = \int_{A} \boldsymbol{n}\cdot(T\, grad\,\phi) dA + \int_{CV} S_\phi\, dV \tag{17}$$

In steady state conditions, Equation (17) can be reduced to the source term, the diffusion term and the convection term (Equation (18)). Time dependent problems can be solved by integrating the time over a small interval Δt as shown in Equation (19).

$$\int_{A} \boldsymbol{n}\cdot(\rho\phi\boldsymbol{u})\, dA = \int_{A} \boldsymbol{n}\cdot(T\, grad\,\phi) dA + \int_{CV} S_\phi\, dV \tag{18}$$

$$\int_{\Delta t}\frac{\partial}{\partial t}\left(\int_{CV}\rho\phi\,dV\right)dt+\int_{\Delta t}\int_{A}\boldsymbol{n}\cdot(\rho\phi\boldsymbol{u})\,dA\,dt=\int_{\Delta t}\int_{A}\boldsymbol{n}\cdot(T\,grad\,\phi)\,dA\,dt+\int_{\Delta t}\int_{A}S_{\phi}\,dV\,dt \tag{19}$$

A small example is going to show the numerical solution to a stationary, incompressible system, without a source term and with spatial discretization in a one dimensional case. The transport and continuity equations apply:

$$\rho\frac{\partial(\phi)}{\partial t}+div(\rho\phi\boldsymbol{u})=div(T\,grad\,\phi)+S_{\phi}\quad\text{and}\quad div(\rho\boldsymbol{u})=0 \tag{20}$$

And become:

$$\frac{\partial}{\partial x}(\rho\phi u)=\frac{\partial}{\partial x}\left(T\frac{\partial\phi}{\partial x}\right)\quad\text{and}\quad\frac{\partial}{\partial x}(\rho u)=0 \tag{21}$$

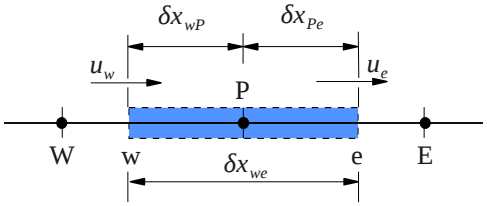

Illustration 5: 1D control volume around a node P

Illustration 5 shows the one dimensional control volume with the general node P, the neighbouring nodes W and E and the control volume faces w and e. Equation (22) shows the integrals, after using Gauss' divergence theorem around the node P.

$$\begin{aligned}\int_{CV}\frac{\partial}{\partial x}(\rho\phi u)\,dV&=\int_{A}n\cdot(\rho\phi u)\,dA=(\rho\phi u\,A)_e-(\rho\phi u\,A)_w\\ \int_{CV}\frac{\partial}{\partial x}\left(T\frac{\partial}{\partial x}\phi\right)dV&=\int_{A}n\cdot\left(T\frac{\partial\phi}{\partial x}\right)dA=\left(TA\frac{\partial\phi}{\partial x}\right)_e-\left(TA\frac{\partial\phi}{\partial x}\right)_w\\ \int_{CV}\frac{\partial}{\partial x}(\rho u)\,dV&=\int_{A}n\cdot(\rho u)\,dA=(\rho u\,A)_e-(\rho u\,A)_w=0\end{aligned} \tag{22}$$

As a simplification, the convective terms will be abbreviated with F and the diffusion terms with D:

$$F_w=(\rho u)_w \qquad F_e=(\rho u)_e \qquad D_w=\frac{T_w}{\delta x_{wP}} \qquad D_e=\frac{T_e}{\delta x_{Pe}} \tag{(23)}$$

Averaging (called central differencing) gives the results shown in Equation (24). As finding an exact solution is impossible for most real systems, numerical approximations have to be applied. Discretisation methods transform continuous problems into problems which have to be solved on discrete points only.

$$\phi_e=\frac{\phi_E+\phi_P}{2} \qquad \phi_w=\frac{\phi_W+\phi_P}{2} \qquad \left(\frac{\partial \phi}{\partial x}\right)_e=\frac{\phi_E-\phi_P}{\delta_{PE}} \qquad \left(\frac{\partial \phi}{\partial x}\right)_w=\frac{\phi_P-\phi_W}{\delta_{WP}}$$
$$F_e\phi_e-F_w\phi_w=D_e(\phi_E-\phi_P)-D_w(\phi_P-\phi_W) \tag{24}$$

Substituting the above expressions and identifying the coefficients as a_W and a_E leads to equation (25).

$$a_P\phi_P=a_W\phi_W+a_E\phi_E \tag{25}$$

With $a_W=D_W+\frac{F_W}{2}$, $a_E=D_E+\frac{F_E}{2}$ and $a_P=a_W+a_E+(F_e-F_w)$. The advantage of this factorization shown in Equation (25) is, that it offers the possibility to change the discretization, by simply changing the components a_W, a_E and a_P.

2.3. The SIMPLE algorithm

The equations mentioned above assume, that the velocity field is either well known or defined. This is usually not the case and because of the fact, that it is often hard to couple the pressure and the fluid velocity, they have to be solved iteratively.

To simulate the fluid flow around the dune the SIMPLE algorithm is used. It is included in the simpleFOAM solver, which can be used for steady-state cases with incompressible, turbulent flow. The acronym stems from Semi-Implicit Method for Pressure-Linked Equations and was introduced by Suhas Patankar and Brian Spalding in 1972. It is perhaps the oldest and most widely used iterative method for the stationary Navier-Stokes equation [4].

The SIMPLE algorithm is a predictor-corrector procedure to solve the discrete momentum Equation (26). Illustration 6 explains the subscripts in Equation (26), *nb* refers to the neighbouring nodes. In the numbering the system neighbours in E, W, N and S direction in the summation $\sum a_{nb} \cdot u_{nb}$ are (i - 1, J), (i + 1, J), (i, J + 1) and (i, J-1). For the sake of simplicity, the cardinal points are used as a reference in all further equations.

$$a_{i,J} \cdot u_{i,J} = \sum^{nb} a \cdot u + \underbrace{(p_{I-1,J} - p_{i,J})}_{\text{pressure forces}} \cdot A_{i,J} + \underbrace{b_{i,J}}_{\text{other sources}} \tag{26}$$

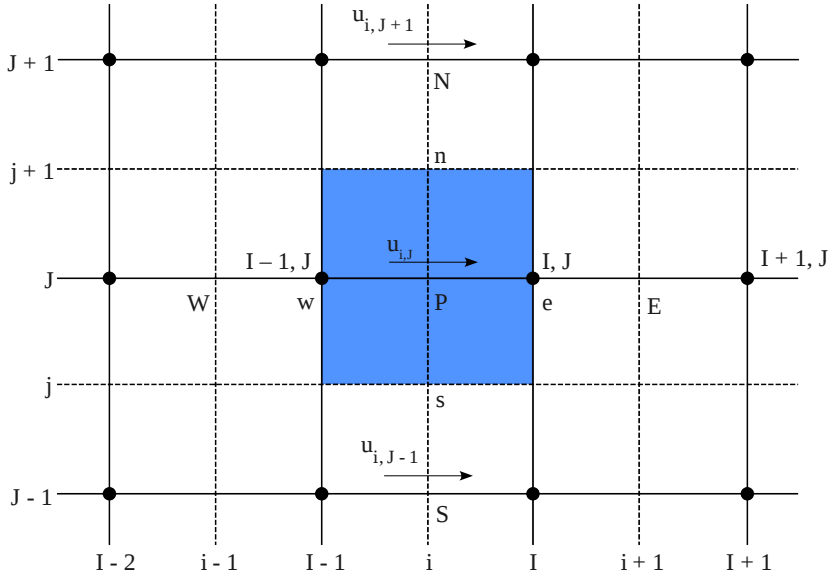

Illustration 6: A u-control volume and its neighbouring velocity components

A is the cross-sectional area of the control volume face and a is the diffusive transfer coefficient, which is defined as shown in Equation (27), with Γ being the interface diffusion coefficient and S_p the source.

$$a_W = \frac{\Gamma_W}{(\delta x_{WP})} \cdot A_W, \quad a_E = \frac{\Gamma_E}{(\delta x_{PE})} \cdot A_P, \quad a_P = a_W + a_E - S_P \tag{27}$$

At first Equation (26) is solved with the current pressure. With Equation (28) the changes in the velocity *u* are being related to the changes in the pressure *p*. The pressure and velocity correction are being referred to, as p^l and u^l. Due to the fact, that in the converged solution all correction will be zero, Equation (28) can be simplified to Equation (29) by neglecting the $\sum a_{nb} \cdot u'_{nb}$ term. To neglect this term is common practice in the SIMPLE algorithm and is is apart from the reason mentioned above hard to justify. It is the main reason why the resulting method does not converge very rapidly. A more gentle way would be to approximate the last term in the pressure equation. This is done in the SIMPLEC algorithm [5]. Another variation is the PISO (Pressure Implicit with Splitting of Operators) algorithm which can be used to solve the Navier-Stoke equation in unsteady problems. The main difference to the SIMPLE algorithm is, that the momentum correction step is performed more than once.

$$u_P^l = \frac{\sum a_{nb} \cdot u'_{nb}}{a_P} + d_p (p'_w - p'_e), \qquad d_p = \frac{A}{a_p} \tag{28}$$

$$u'_P \approx d_p (p'_w - p'_e) \tag{29}$$

Next the mass conservation is applied to a control volume centred on the pressure node. The net mass flux results from the current (u^*) plus the correction (u') velocity fields:

$$\sum_{faces} \rho u_n^* \cdot A + \sum_{faces} \rho u'_n \cdot A = 0 \tag{30}$$

i.e.

$$(\rho u' A)_e - (\rho u' A)_w + ... = -\dot{m}^* \tag{31}$$

Writing Equation (31) in terms of the pressure correction results in Equation (32). With

$a_E = (\rho u A)_e$ … and $a_P = \sum a_F$, it can be simplified to Equation (33).

$$(\rho u A)_e (p'_P - p'_E) - (\rho u A)_w (p'_W - p'_P) + ... = -\dot{m}^* \tag{32}$$

$$a_P p'_P - \sum_F a_F p'_F = -\dot{m}^* \tag{33}$$

Due to the fact that Equation (33) has precisely the same form, as the linearized, discretized scalar equation, the same solvers can be used. Afterwards the pressure and velocity have to be corrected, as shown in Equation (34).

$$\begin{aligned} p_P &\rightarrow p_P^* + p'_P \\ u_P &\rightarrow u_p^* + d_p (p'_w - p'_e) \end{aligned} \tag{34}$$

2.4. OpenFOAM

The name OpenFOAM stems from “Open Field Operation and Manipulation” it is a C++ library, which supplies *utilities* and *solvers* to perform data manipulation and solve specific problems in continuum mechanics. Examples of these processes include but are not limited to fluid flows with or without chemical reactions, turbulence modeling and heat transfer, as well as solid dynamics, electromagnetics and the pricing of financial options.

It offers users with a basic knowledge in C++ programming and object-orientation to write their own solvers or modify existing ones. Illustration 7 shows an overview of the OpenFoam structure. It supplies numerical solvers, as well as pre- and post-processing utilities.

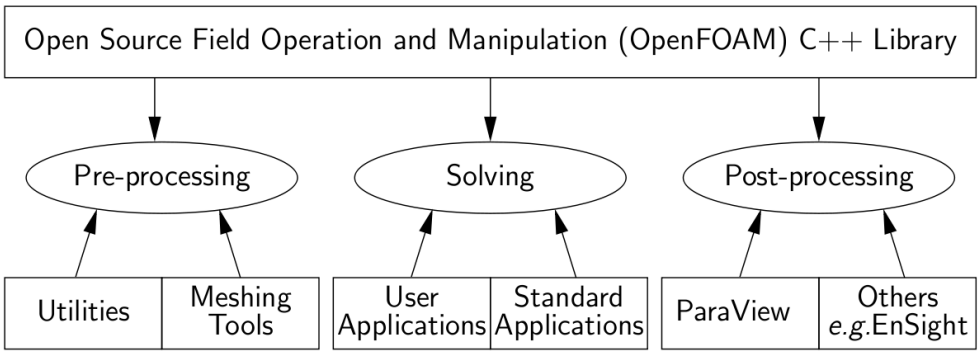

Illustration 7: Overview of the OpenFOAM structure [OpenFOAM User Guide (2011)]

The main difference to other cfd toolboxes is, that it is released under the GNU General Public License and completely open source and freely available.

Illustration 8 shows the basic directory structure to build an OpenFOAM case. The system directory contains at least three files. First of all the controlDict file in which for example the start and end time, as well as the time steps and data output are defined. It also contains the fvSchemes file, in which the discretisation schemes mentioned in the theory chapter can be found. Usually the standard Gaussian finite volume integration is the common choice. It is based on summing values on cell faces, which must be interpolated from cell centres. Other options are for example “leastSquares” or “fourth”. The third file is the fvSolution file, it contains for example the equation solvers and tolerances.

The constant directory contains information, concerning the mesh in the polyMesh sub-directory and specified physical properties. Before starting a case, a time folder has to be created, named

after the start time defined in the controlDict file (usually 0). It has to include initial values and boundary conditions. OpenFoam then writes the results to files into folders named after the time.

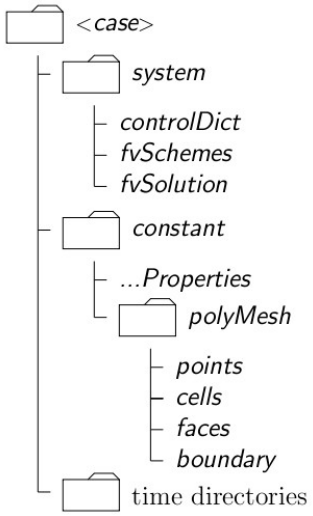

Illustration 8: Case directory structure [OpenFOAM User Guide (2011)]

2.4.1. The simpleFOAM solver

The SIMPLE algorithm is implemented in the simpleFOAM solver, which can be found in the $FOAM_SOLVERS/applications/solvers/incompressible/simpleFoam/simpleFOAM.C file. The assumption, that the dune simulation is incompressible is valid, because the occurring pressure gradient is very small and can therefore be neglected. It solves the Navier Stoke eqaution in an iterative procedure and basically undergoes the following steps:

1. Set the boundary conditions.
2. Solve the discretized momentum predictor.
3. Compute the cell face fluxes.
4. Solve the pressure equation and apply under-relaxation
5. Correct and adjust the values at the cell faces.
6. Correct the velocities on the basis of the new pressure field.
7. Update the boundary conditions
8. Repeat, until the convergence criteria are satisfied

As shown in Listing 1 the simple loop is implemented as a while-loop. It either ends, when the given run time or a specified convergence criteria is reached or the solver diverges.

In line 55 the pressure calculated at the previous iteration is being stored. In the beginning UEqn.H defines the equation for U and then under-relaxes the velocity. This helps to improve the stability, by limiting the amount of change a variable undergoes from one iteration to the next. The specifications can be given in the fvSolution file, which can be found in the “system” folder of the case. Afterwards the momentum predictor is being solved. In the pEqn.H file, the boundary conditions for p are updated, then the diffusive transfer coefficient a_p and the velocity are calculated. Afterwards the flux is calculated. After defining and solving the pressure equation repeatedly for the number of times defined in the “fvSolution” file, it is being corrected. Then the continuity errors are calculated and the pressure is under-relaxed for the momentum corrector and, if need corrected. Back in listing 1 the convergence is being checked and repeated until the specified convergence criteria are satisfied.

Listing 1: SIMPLE loop in simpleFOAM

```
51    Info<< "\nStarting time loop\n" << endl;
52    while (simple.loop())
53    {
54        Info<< "Time = " << runTime.timeName() << nl << endl;
55        p.storePrevIter();
56        // --- Pressure-velocity SIMPLE corrector
57        {
58            #include "UEqn.H"
59            #include "pEqn.H"
60        }
61        turbulence->correct();
62        runTime.write();
63        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
64            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
65            << nl << endl;
66    }
67    Info<< "End\n" << endl;
```

2.4.2. The pimpleFOAM solver

As the simpleFoam solver, doesn't converge with the given geometry, the starting parameters will be taken from a transient solver. It uses the PIMPLE algorithm, which is merged from the PISO and SIMPLE algorithm. Like the SIMPLE algorithm, the PISO algorithm neglects the velocity correction in the first step, but will calculate it later using the first velocity correction, which leads to additional corrections for the pressure [6].

3. Modelling the wind flow around a two dimensional dune

3.1. The geometry

To validate the results of the simulation, they are compared to a study by Giles F.S. Wiggs et al. (1996). They suggest, that surface shear stresses are induced by streamwise acceleration as well as streamline curvature. To prove this theory, they took field measurements on an unvegetated about 10 meter high barchan dune and compared them with measurements of a 1:2000 scale fixed model in a wind tunnel.

Illustration 9 displays the centre line of the dune used in this studies. Illustration 10 shows the corresponding geometry created with *blockMesh*, which is a mesh generating utility supplied by OpenFOAM. It generates the mesh from the input specified in the *blockMeshDict* dictionary, which is located in the <case>/*constant/polyMesh* folder. Furthermore, cells, faces and patches are created. A patch includes one or more areas of the boundary surface, that do not necessarily have to be physically connected. The geometry has to be generated in three dimensions, because OpenFOAM operates in a three dimensional Cartesian coordinate system. By specifying the boundary condition for the front and back plane, referred to by the patch "*defaultFaces*" as "*empty*" means, that no solution for the third dimension is required.

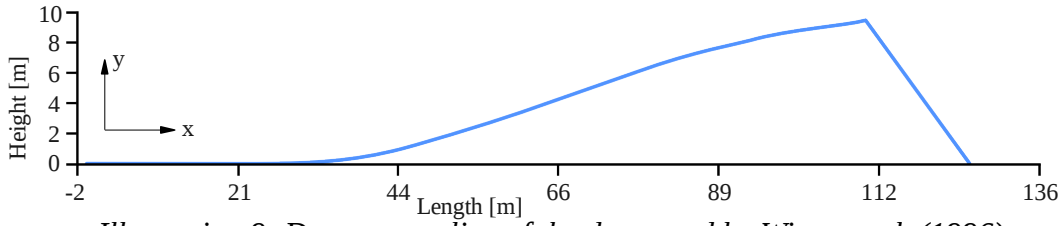

Illustration 9: Dune center line of the dune used by Wiggs et al. (1996)

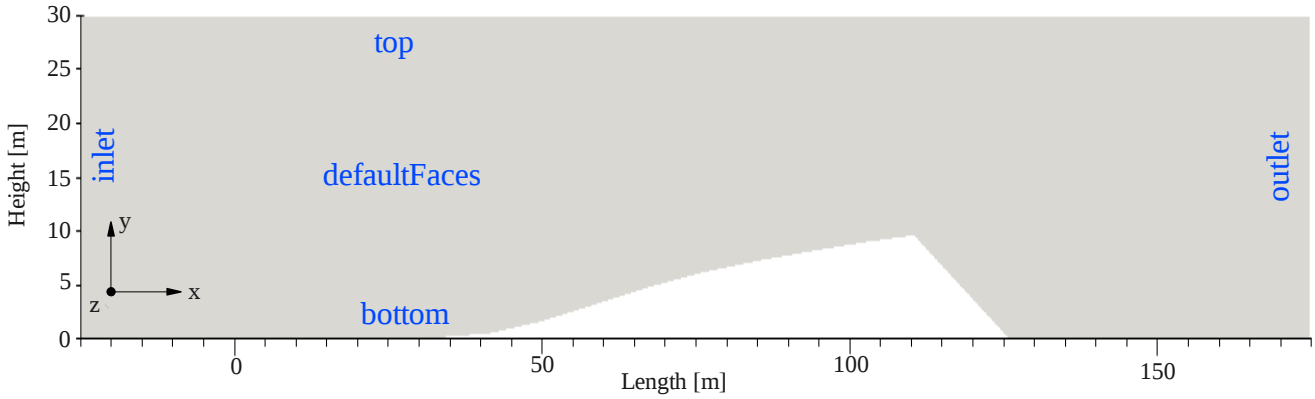

Illustration 10: Geometry and patches (blue) displayed in ParaView, created with blockMesh

3.1.1. Mesh generation in OpenFOAM

To create a mesh with *blockMesh* the geometry is manually decomposed into a set of one or more hexahedral blocks. Edges of the blocks can be lines, arcs or splines. To refine the mesh a number of cells in each direction of the block can be adapted. Illustrations 11-13 show different graded meshes created with *blockMesh*.

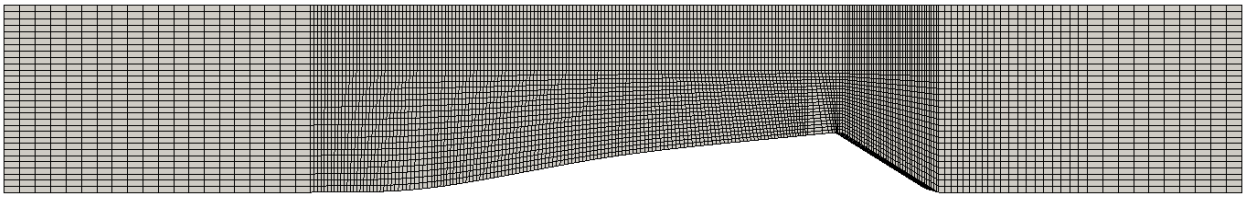

Illustration 11: Coarse mesh created with blockMesh

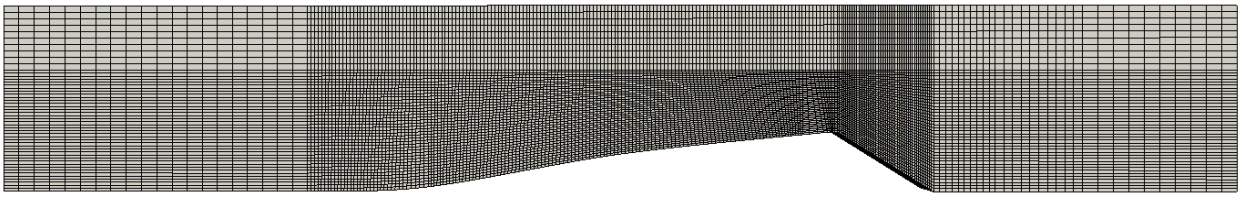

Illustration 12: Medium graded mesh created with blockMesh

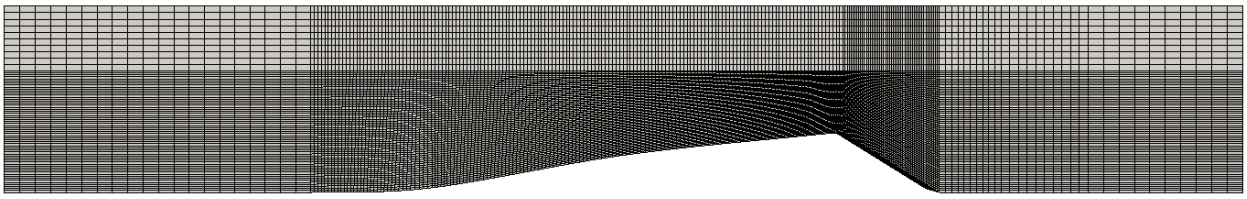

Illustration 13: Fine graded mesh created with blockMesh

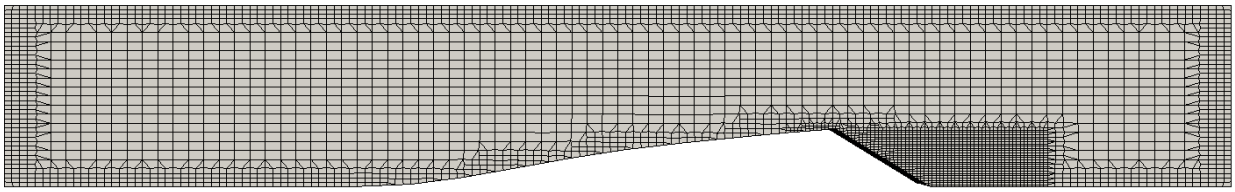

Illustration 14: Fine graded Mesh created with snappyHexMesh

The fine graded mesh created with blockMesh contains 17.500 hexahedra cells, compared to 12500 cells for the medium graded mesh and 7200 for the coarse mesh.

To compare not only the solver, but also different types of meshing tools, another OpenFOAM utility called *snappyHexMesh* is used. It creates meshes containing hexahedra and split-hexahedra geometries. In order to get the Stereolithography (*.stl) file that *snappyHexMesh* needs, yet an other OpenFOAM utility, called *surfaceMeshTriangulate* is being used. It converts

the outside of the mesh created with *blockMesh* to a surface file. To create a mesh in *snappyHexMesh* a dictionary is needed. The *snappyHexMeshDict* can be found in the system sub-directory of the case. Illustration 14 shows, that the turbulence region behind the dune is refined with the code shown in Listing 2. The type "*searchableBox*" defines a region by a bounding box.

Listing 2: Excerpt from snappyHexMeshDict

```
38 //Refine at the downwind slope
39 refineDune
40  {
41      type searchableBox;
42      min (105 0 -5);
43      max (145 10 5);
44  }
```

The cell splitting is also done according to the *snappyHexMeshDict* in the *castellatedMeshControls* sub-dictionary. A detailed explanation can be found in the OpenFOAM User Guide, supplied by the OpenFOAM Foundation, in Chapter 5.4. The mesh shown in Illustration 14 consits of 35608 hexahedra, 739 prism and 2661 polyhedra cells.

3.2. The boundary conditions

In OpenFOAM the boundary conditions can be found in the "*startTime*" folder, which is defined in the *controlDic*t file (usually *<caseDIR>/0*). The boundary conditions for the velocity are shown in table 1. The most common boundary conditions are either fixed values or fixed gradient. If a variable is specified with a value on a boundary patch, it is a Dirichlet boundary condition and referred to as "fixed value". If a gradient is defined at the patch, then it is called Neumann boundary condition and referred to as "fixed gradient".

patch	boundary	value [m/s]	Meaning of the boundary condition
inlet outlet	*freestream*	(8 0 0)	If the flow is going outside the boundary it will be locally "*zeroGradient*" (normal gradient of Φ is zero), if it is going to the inside the the boundary it will be locally "*fixedValue*" (Normal value of Φ is specified with (8 0 0)).
bottom	*fixedValue*	(0 0 0)	The velocity at the wall is zero.
top	*inletOutlet*	(8 0 0)	Switches the velocity and pressure between *fixedValue* and zeroGradient depending on direction of the velocity. All fields are initialized as an inlet flow with a fixed value of (8 0 0) m/s. If the pressure forces the fluid flow outward at any part of the boundary, this specific facet ist treaded as an outlet, hence has zero gradient. This prevents an unnatural pressure build-up and acts like a driven free flow direction.
defaultFaces	*empty*	---	The geometry will be treated as two dimensional.

Table 1: Boundary conditions for the velocity

Sadly no information concerning the approximate wind speed is given in the paper. So it is set to 8 m/s which is about 28.8 km/h or 15.55 knots. Table 2 shows the boundary conditions for the pressure. The unit is [m^2/s^2], because OpenFOAM uses the specific pressure, which is p/ρ.

patch	boundary	value [m^2/s^2]	Meaning of the boundary condition
inlet top	*fixedValue*	uniform 1.13e5	The value of Φ is $1.15 \cdot 10^5 m^2/s^2$
bottom outlet	*zeroGradient*	---	The fluid gradient at the wall is zero.
defaultFaces	*symmetryPlane*	---	The geometry will be treated as two dimensional.

Table 2: Boundary conditions for the pressure

3.3. The turbulence model

Turbulence is a flow regime, which is highly irregular. It is the most commonly occurring flow state and is maintained by shear in the mean flow. A huge amount of research is dedicated to the development of numerical methods to better understand turbulence.

There are three main groups of methods, Reynolds-averaged Navier-Stokes equations (RANS), large eddy simulations (LES) and direct numerical simulations (DNS). In this introduction, we are mainly focusing on the turbulence models for Reynolds-averaged Navier-Stokes equations. The Navier-Stokes equation is time-averaged prior to the application of numerical methods. This adds extra terms to the flow equations, due to the interaction between various turbulence fluctations. Those turbulent fluctuations can be described implicitly using different modelling approaches - the so called turbulence models. The best known one is the k-ε model. It is being used in this case. The model assumes, that the turbulent viscosity μ is isotropic. Even though this results in it not performing very well in cases of large adverse pressure gradients, it can be used in this case.

Instead of the one equation Spalart-Allmaras model, that was used before in the “airFoil2D” OpenFoam tutorial (which is used as a basis for this case), two transport equations have to be solved. One for the turbulent kinetic energy k and another one for the rate of dissipation of turbulent kinetic energy ε [7]. While ε determines the scale of the turbulence, k determines the energy of the turbulence. To estimate k and ε, Equations (35) and (36) can be applied. With U being the mean velocity, which is estimated to be 8 m/s and I being the turbulence intensity, which is estimated to 1% of the mesh inlet height [7].

$$k = \frac{3}{2}(U \cdot I)^2 \tag{35}$$

C_μ is a turbulence model constant and is usually taken as $C_\mu = 0{,}09$ and l is the turbulence length scale and will be approximated with 2%.

$$\epsilon = C_\mu \frac{k^{\frac{3}{2}}}{l} \tag{36}$$

Applying these changes is done by adding the boundary conditions for k and ε to the “0” folder of the case, changing the “RASModel” in the “RASProperties” file to “kepsilon” and adjusting the “fvSchemes” and “fvSolution” files.

patch	boundary	value [m^2/s^2]	Meaning of the boundary condition
inlet top outlet	inletOutlet	uniform 8.64	Switches between fixedValue and zeroGradient depending on direction of the velocity.
bottom	kqRWallFunction	uniform 0	Special turbulent wall treatment
defaultFaces	symmetryPlane	---	The geometry will be treated as two dimensional.

Table 3: Boundary conditions for k

The boundary conditions for epsilon are identical, with the exception that the uniform value for the inlet, outlet and top is 13.17 and the bottom patch is epsilonWallFunctions.

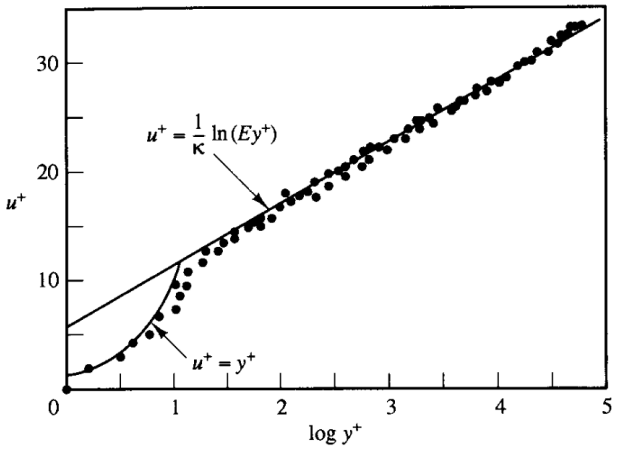

Illustration 15: Velocity distribution near a solid wall [Schlichting, H. (1979)]

In turbulent modelling theory flows far away from solid walls are almost only inertia dominated, whereas near walls viscous effects are important.

Those effects are described using correlation equations - the so called "wall functions" - which incorporate different model and wall parameters. In the k-epsilon model the treatment of wall functions depends on the local Reynolds number.

3.4. Results

In the paper G. F. S. Wiggs normalized the velocity data by measurements at a reference station, positioned 50 m upwind of the toe on the centre line. Equation (37) shows the calculation for the fractional speed-up ratio (δs) as used by Jackson and Hunt (1975).

$$\delta_s = \frac{(u_y - U_y)}{U_y} \tag{37}$$

u_y is the velocity at a height y on the dune and U_y is measured at the same height at a reference station. Using Equation (37), flow acceleration occurs at fractions, so a δs of 0.28 indicates an acceleration compared to upwind values at the same height of 28% [8].

Wiggs noticed a reduction in near-surface wind velocity, which is succeeded by a uniform increase in wind speed to a maximum near the crest. According to the paper [8] the wind speed is greatest at y = 1m with δs = 0.39 as shown in Illustration 16. This is in contrast to the acceleration at 0.25 m height, where δs is only 0.27-0.28 [8].

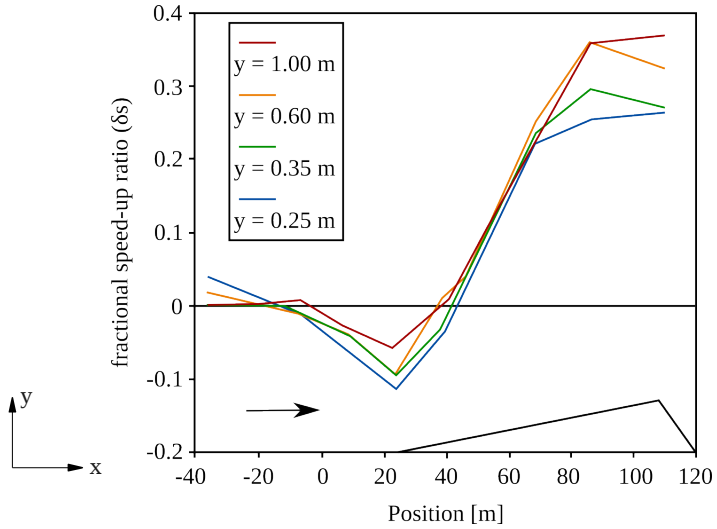

Illustration 16: Field measured fractional speed-up ratio [Wiggs (1996)]

In order to quickly compare the simulation data, it is sampled using a list of points, at the same height as used in the paper. This is done using the sample utility supplied by OpenFOAM, by specifying the case through a sampleDict in the case system directory. Then, using Equation (37), the fractional speed-up ratio is calculated. The results can be found in Illustrations 17-19.

A variance in the results of the different graded meshes is clearly visible, the number of jumps in the graph increases depending on how close the values are taken to the surface and on how coarse the mesh is. They can be noticed especially at inflexion points in the dune geometry. This happens because in coarser graded meshes, the gradient or rather the fluctuations in speed do also occur higher above the geometry. Locally the mass balance has to apply in every cell, so that in a coarser mesh, irregularities will propagate more throughout the flow region.

The trend between the paper and the simulation shows a strong correlation. Though the fractional increase in speed is lower, the maxima are about the same and the same tendencies can clearly be seen. The slower increase could be due to inaccurate turbulence parameters, which make the air seem more viscous than it actually is. Furthermore, the wind speed is not known and could have an significant effect on the speed-up. Another reason could be, that the geometry is treated as a two dimensional case, even though the measurement were obviously taken in three dimensions.

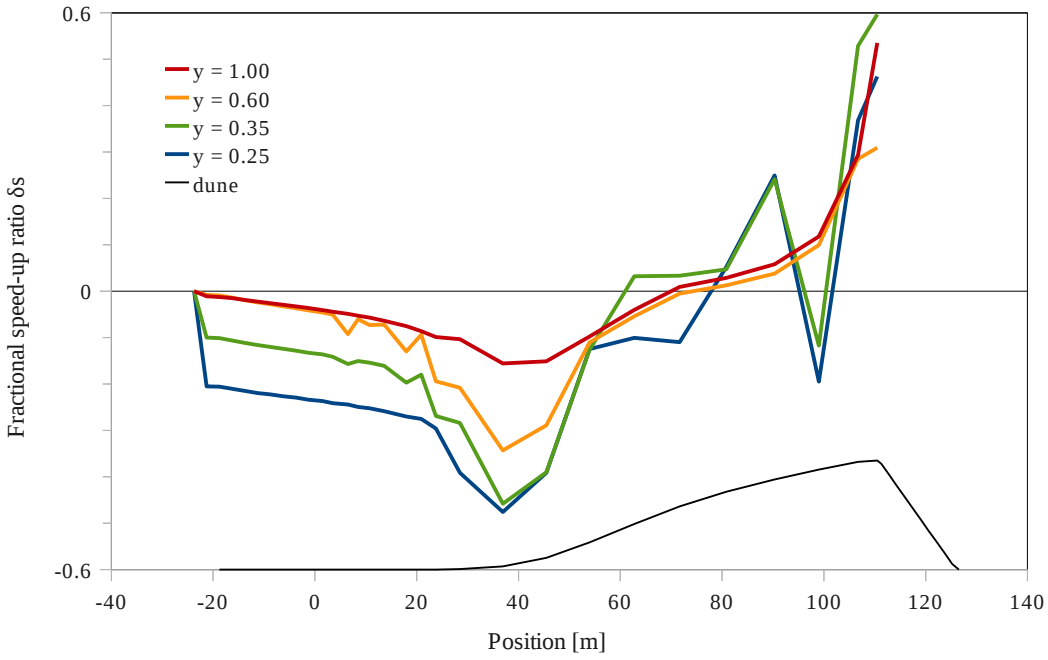

Illustration 17: Fractional speed-up ration for a coarse graded mesh blockMesh

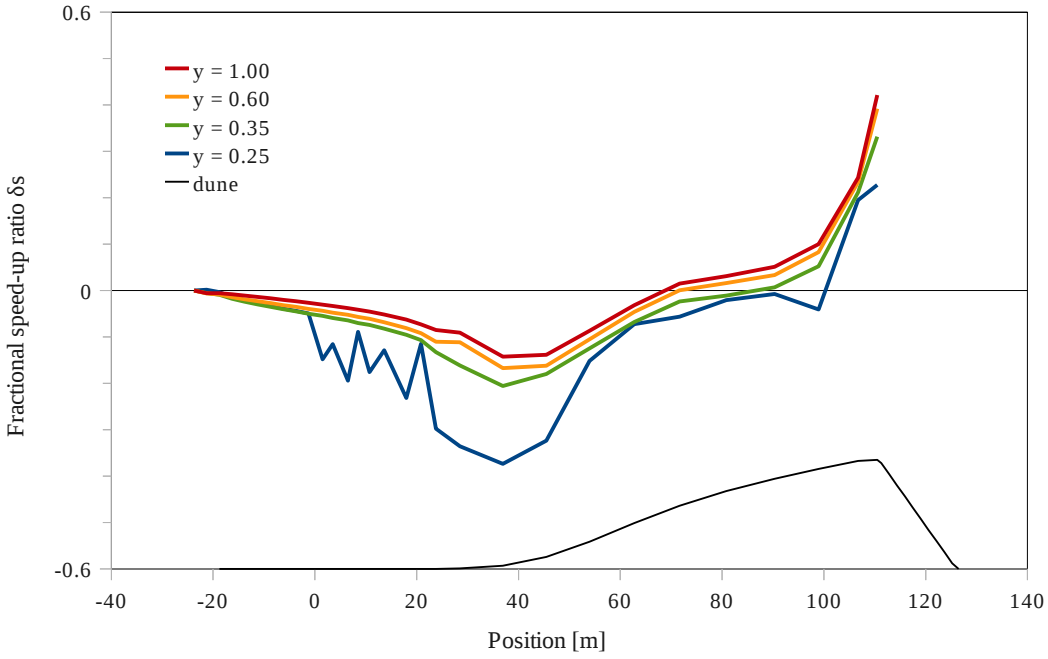

Illustration 18: Fractional speed-up ratio for a medium graded mesh in blockMesh

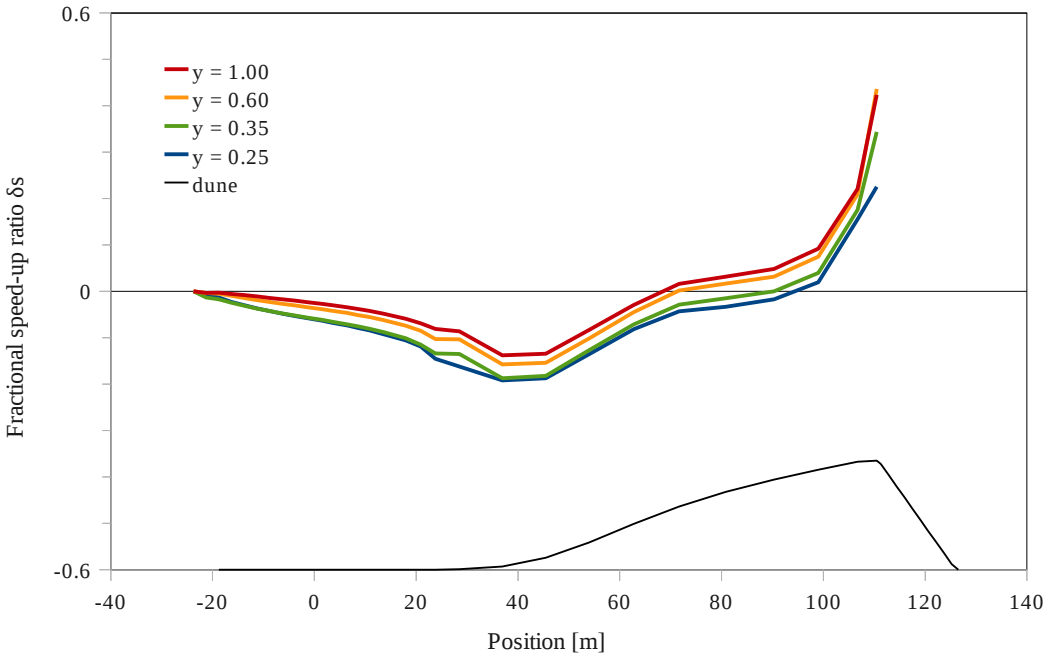

Illustration 19: Fractional speed-up ratio for a fine graded mesh in blockMesh

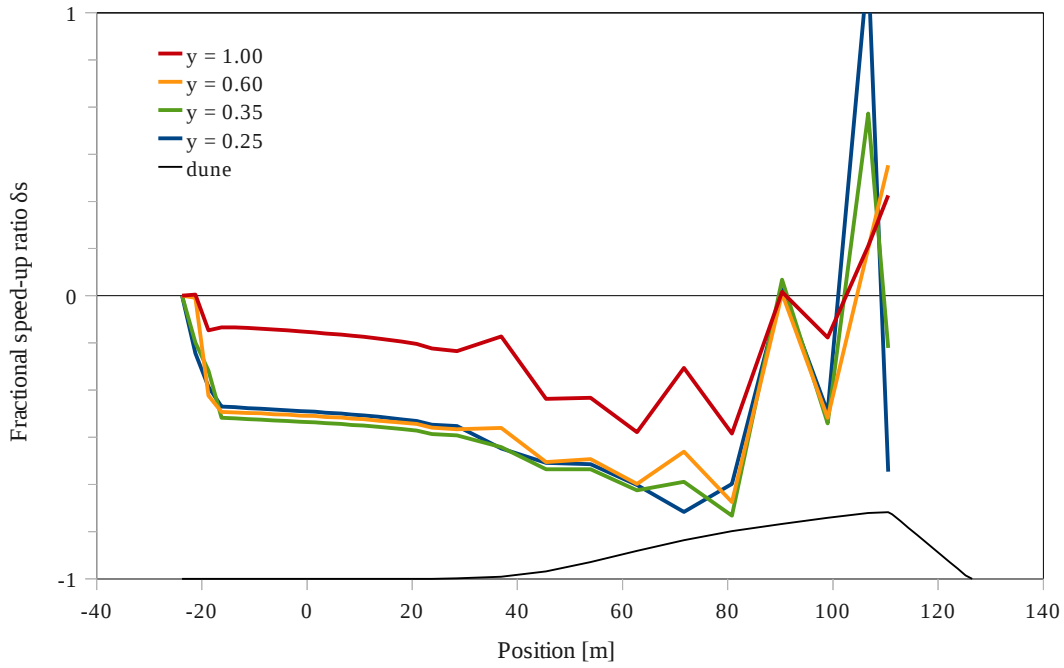

Illustration 20: Fractional speed-up ratio for a fine graded mesh created in snappyHexMesh

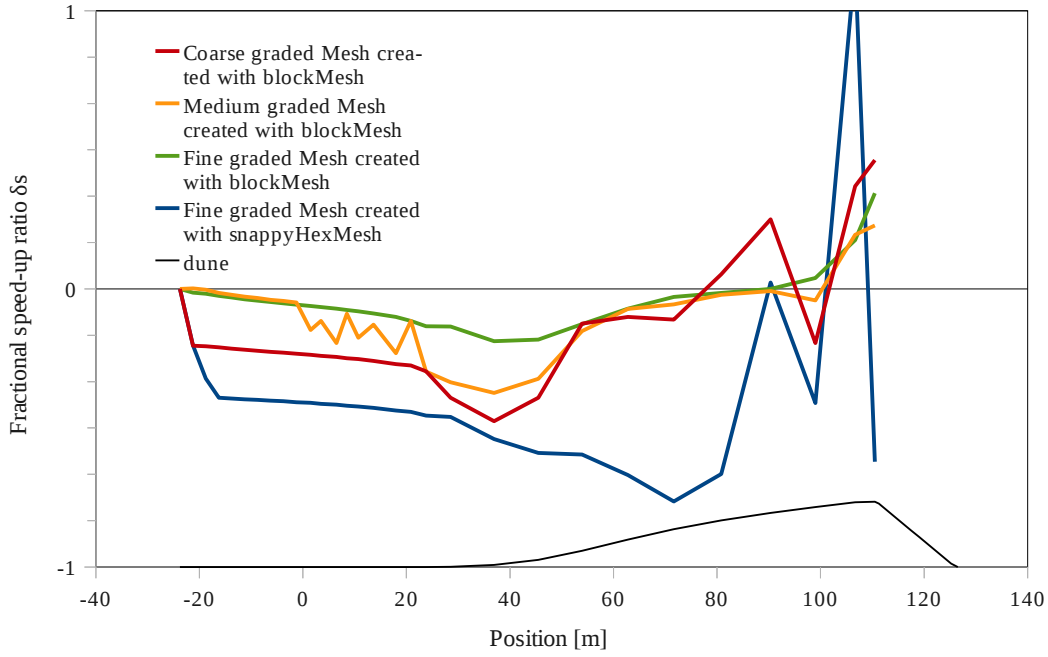

Illustration 21: Comparison of the different mesh types at 0.25 meters

Illustration 20 shows the fractional speed-up ratio for a fine graded mesh created in snappyHexMesh (Illustration 14). It strongly resembles the coarse mesh created in blockMesh. This suggests, that the geometry has to be refined further along the dune. Illustration 21 shows the comparison of the different meshes sampled at a height of 0.25 meters. Further refinement won't be part of this thesis, because the simulation time of the mesh created with snappHexMesh is approximately 120 times higher than using the optimized 2D blockMesh. Creating a mesh in snappyHexMesh only makes sense for more complex three dimensional cases.

Illustration 22 shows the overall velocity profile of the fine graded mesh created in blockMesh. The speed maximum is at the peak of the dune with about U=11.76 m/s. An eddy is formed behind the dune.

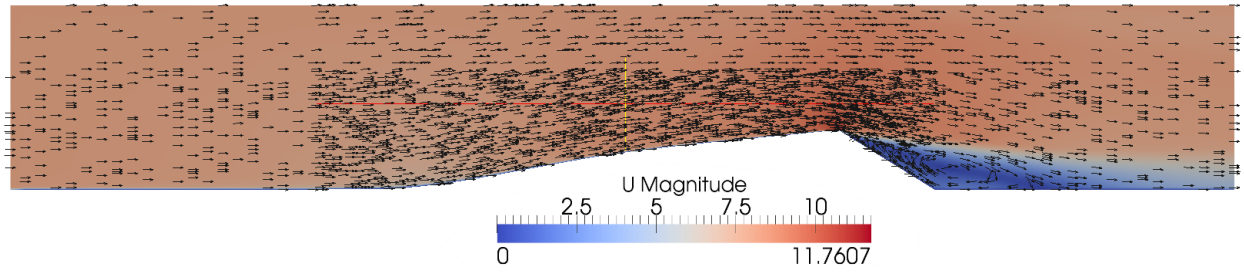

Illustration 22: Velocity profile of the fine graded dune created in blockMesh

Overall the simulation gave good results and can be used, with further adjustments, for example by changing the turbulence parameters.

4. Modelling the wind flow around a three dimensional dune

4.1. The geometry

In the paper “The role of streamline curvature in sand dune dynamics“ from 1996, G. Wiggs does not give any information about the total length of the dune. Instead the position of the study site in Oman is given. Illustration 23 shows a satellite picture of the location. With this information, the three dimensional dune shown in Illustration 24 is created. To do so an other free, open-source software named SALOME[2] is used. It offers pre- and post-processing tools for numerical simulations. The length of the dune is assumed to be 165 meters, with the given distance from crest to toe of 86 meters and a width of 130 meters between the two horns.

Illustration 23: Satellite picture of the study side in Oman showing sand dunes

The same centre line, that is already used in the two dimensional case is applied to make a direct comparison possible. In order to extract it from the mesh created with blockMesh fifteen points from the dune windward side are sampled and then transferred into SALOME.

[2] http://www.salome-platform.org/

Illustration 24: Three dimensional dune used in the simulations

4.1.1. The mesh

The Mesh is created in SALOME using the Netgen 1D-2D-3D algorithm. NETGEN is an automatic 3D tetrahedral mesh generator. A tetrahedron is a polyhedron formed by four triangular faces, with three faces meeting at each vertex. Illustration 25 shows a comparison of tetrahedral and hexahedral cells in two and three dimensions.

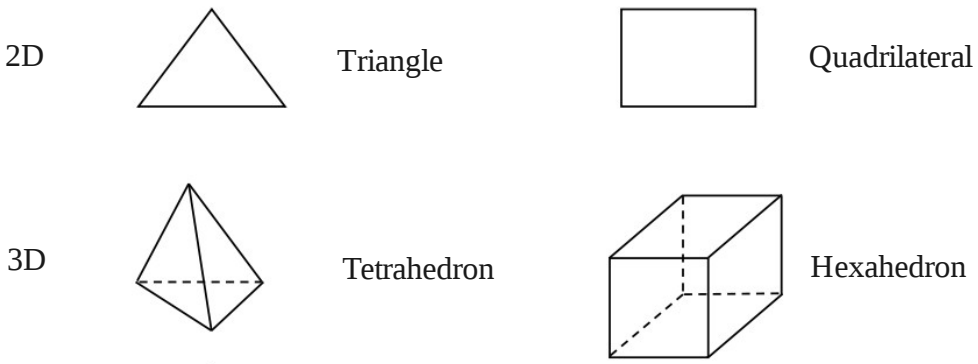

Illustration 25: Comparison of tetrahedral and hexahedral cells

The mesh created in SALOME is unstructured, as shown in Illustration 26. The possibility to

account for pertinent flow features is not offered, meaning that it is not possible to automatically create a mesh with a higher resolution close to the bottom. The mesh consists of 401,002 tetrahedral cells, which is close to the limit set in Salome of 500000 elements. So it is not possible to capture all relevant flow features above the dune (there are not enough cells in y-direction). The maximum skewness of 0.6 is acceptable, it should be smaller then 0.8 to get good results. It is important to check the skewness, because it can have a huge impact on the accuracy and robustness of the CFD solution. The skewness for tetrahedral and triangular cells is calculated as shown in Equation (38).

$$skew = \frac{optimal\ cell\ size - cell\ size}{optimal\ cell\ size} \tag{38}$$

To convert the Salome mesh into a FOAM mesh, the ideasUnvToFoam command is used. To prove the theory, that comparing the cell count, hexahedral meshes will give more accurate solutions, the mesh is converted to a hexahedral mesh with snappyHexMesh. This theory should especially apply if the grid lines are aligned with the flow.

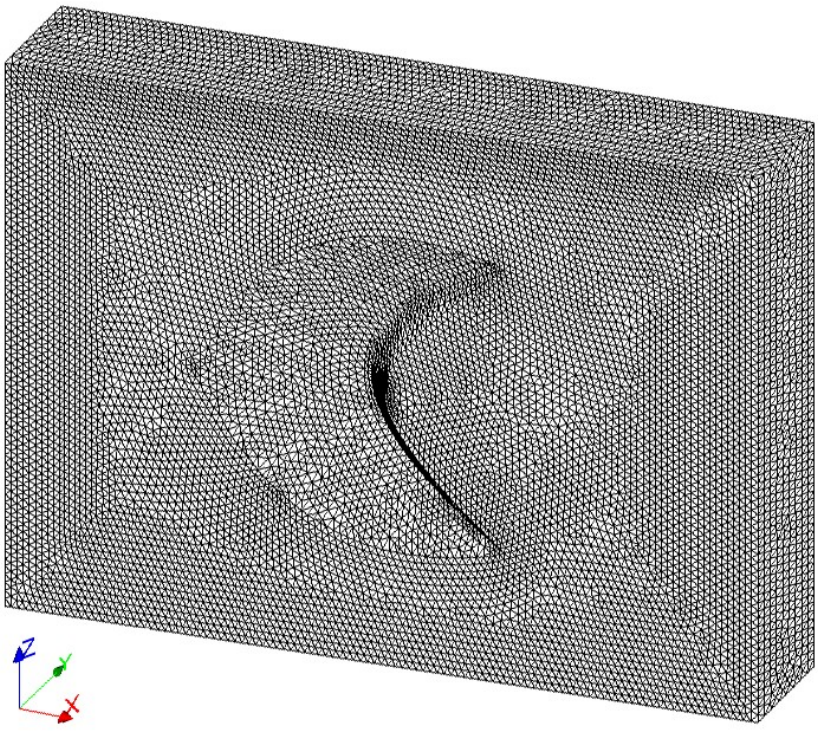

Illustration 26: Mesh created in SALOME

Illustration 27 shows the mesh created in snappyHexMesh. The two dimensional case has shown the necessity to use a small cell size above the dune to get good results. The mesh is created accordingly. It consist of 340818 hexahedral and 9361 polyhedral cells.

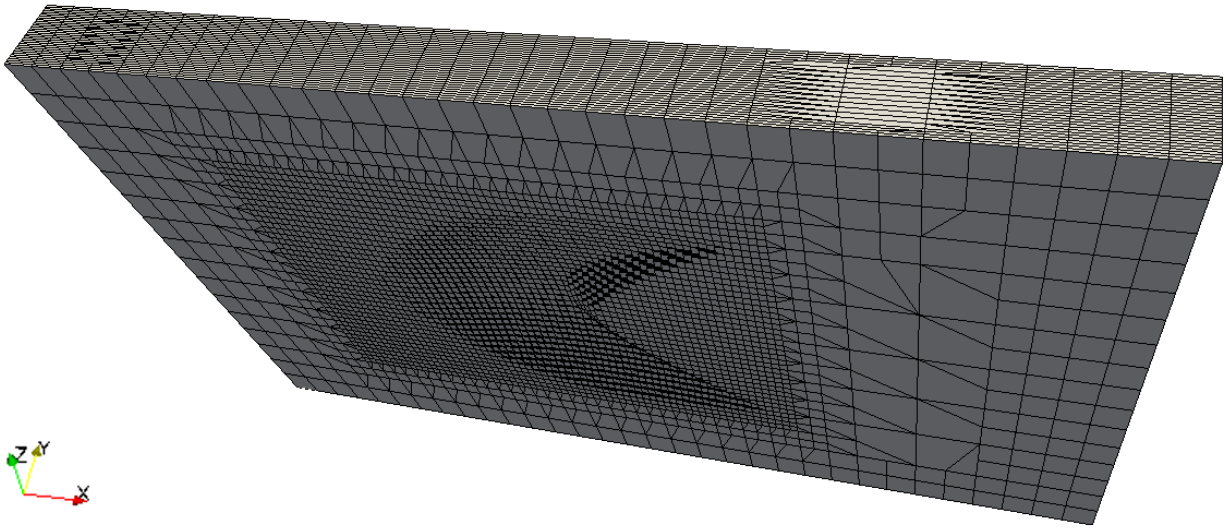

Illustration 27: 3D mesh created with snappyHexMesh

The skewness of about 0.3 is pretty good. For a hexahedral or quadrilateral cell it is calculated by considering the deviation from the minimum or maximum angle compared to the right angle as shown in Equation (39).

$$skew = max\left[\frac{\theta_{max} - 90}{90}, \frac{\theta_{min} - 90}{90}\right] \tag{39}$$

Illustration 28 shows a comparison between an optimal and a skewed cell for the two different cell types.

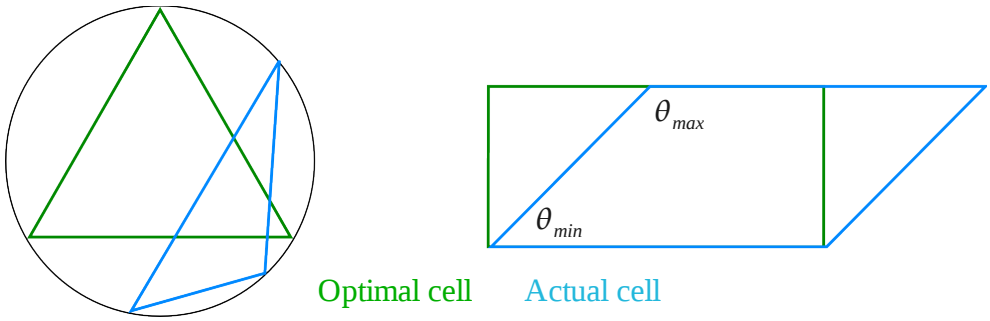

Illustration 28: Comparison between an optimal and a skewed cell

4.2. The boundary conditions and turbulence model

In the three dimensional case, the same boundary conditions and the same turbulence model are chosen as in in the two dimensional case.

The only difference is, that the *defaultFaces* patch is defined as *symmetryPlane* and not as *empty*. This suggests, that the domain is infinite in z direction and that the boundary has no influence on the case.

4.3. Results

Again, in order to quickly compare the simulated data, it is sampled using a list of point, at the same height, as used in the paper. Then using Equation (37) the fractional speed-up ratio is calculated. Illustration 29 clearly shows the difference in quality between the tetrahedral mesh created with the NETGEN algorithm and the hexahedral mesh created with snappyHexMesh.

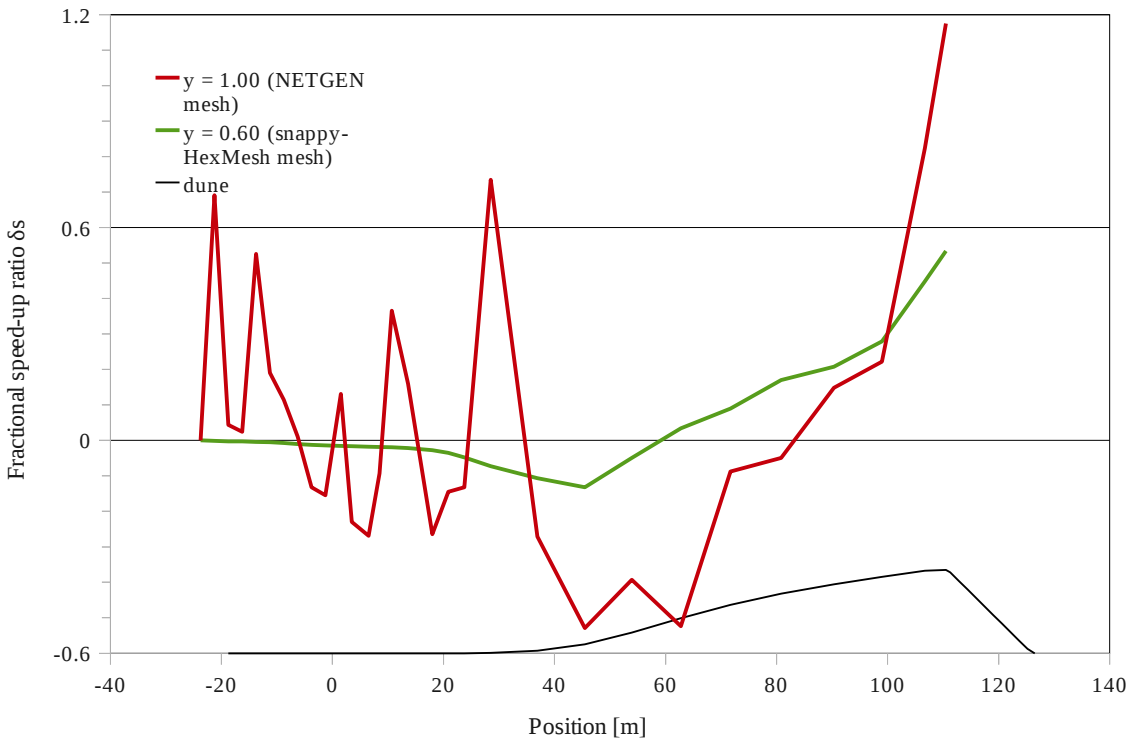

Illustration 29: Comparison of a fractional speed-up ratio of a tetrahedral and hexahedral mesh

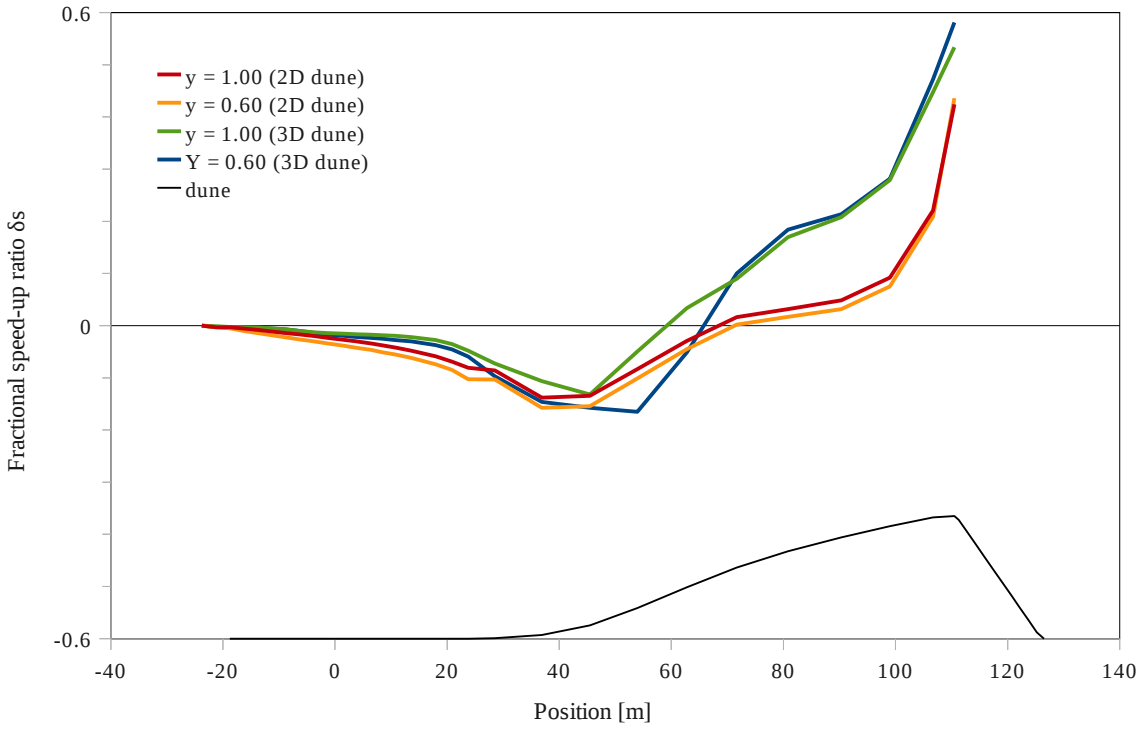

Illustration 30: Comparison of a fractional speed-up ratio of a 2D and 3D hexahedral mesh

The speed-up ratio sampled from the snappyHexMesh mesh follows a clear trend, while the trend sampled from the tetrahedral mesh is interrupted by various jumps. The difference is mainly, because there are not enough cells in y direction of the mesh created with the NETGEN algorithm.

Illustration 30 shows a comparison between the speed up ratio of the two and three dimensional cases created with hexahedral meshes. All four graphs show the same tendency, by slightly decreasing in the beginning of the dune. Afterwards they follow an upward trend. Starting with a significant increase, then reaching a plateau between about 75 and 100 meters. Afterwards increasing again and reaching a peak at the peak of the dune.

It has to be noted, that the second increase after hitting the plateau can't be found in the original paper. This could be because the centre line given in the paper, isn't an accurate representation of the actual dune.

5. Simulating particles

The first pioneers in the field of fluid-particle interaction were Sir Isaac Newton (1642-1727) and Jean le Rond d'Alembert (1717-1783). Though it has a fairly long history, the appliance to numerical simulation is fairly new. With the OpenFOAM version 2.0.0 (16th June 2011) the discrete element method (DEM) has been introduced. It is very interesting for simulating the stresses and displacements in a volume containing a large number of particles. This offers the possibility to solve the averaged Navier-Stokes equation using the Finite Volume Method (FVM), as shown in this thesis and the solid sand phase using the DEM. A coupling and improvement is attempted by various researchers around the world.

A coupling is already archived in an other open source package called CFDEM[3] developed by Christoph Goniva (JKU Linz). It can be integrated into OpenFOAM, but seems to have problems with the latest version (v2.1.0) due to incompatibility with the engine Search model.

Though it is possible to combine FVM and DEM, the approach is not very practical for simulating sand dunes, because they consist of an enormous amount of particles. An other more practical approach would to tread them as fluidized models.

[3] http://web678.public1.linz.at/Drupal/

6. Fluidized models

Generally speaking a fluidized model is a fluid-solid mixture that shows fluid-like properties. The process of fluidization occurs when a fluid is passed through a quantity of a solid particle substance, for example in a bed reactor.

6.1. Surface tracking

The name surface tracking refers to a case with a separate mesh for each phase, which obtains its motion from the boundary conditions. OpenFOAM includes multiple solvers that offer surface tracking methods, for example the interFoam solver. It can be used for two incompressible fluids and offers for example the possibility to model the turbulence with the Reynolds averaged model, that has already been introduced in the turbulence model of the two dimensional case. In the interDymFoam solver a tool is added, that allows to refine the mesh dynamically. Illustration 31 shows different types of rheological models. While the fluid can be treated as Newtonian, this is not possible for the sand dune.

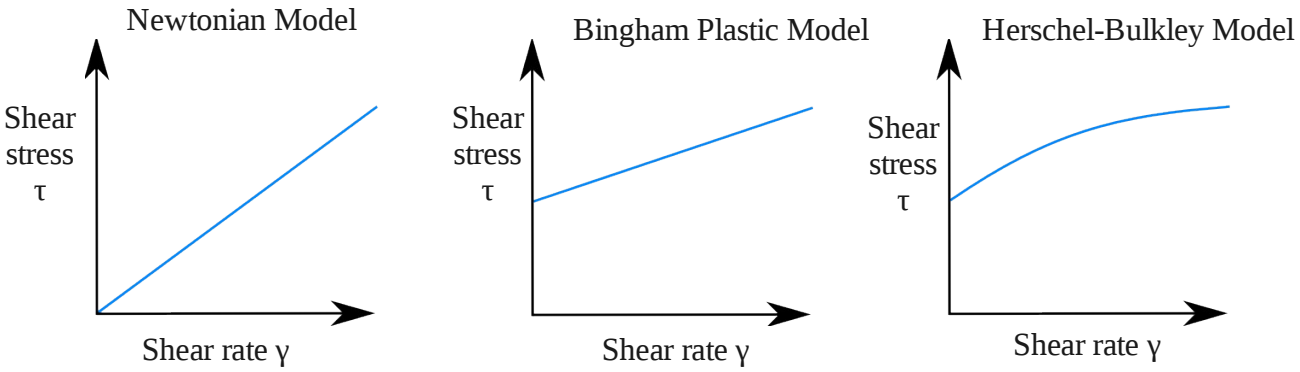

Illustration 31: Comparision of different rheological models

Herschel-Bulkley fluids as well as Bingham fluids require a certain minimum amount of stress before they start to flow. They are defined by the consistency, which is a constant of proportionality, the flow index, which measures the degree to which the fluid is shear-thinning or shear-thickening and the shear stress. Sadly those constants have proven to be difficult to find.

6.2. Dispersed models

OpenFOAM includes an Eularian two-phase solver for transient simulations of a fluidized bed. Instead of tracking the motion and computing the rates of change of conserved properties for collections of fluid particles as in the Lagrangian approach it is making up a region fixed in space for collections of fluid elements [1]. The twoPhaseEulerFoam solver can be used for incompressible two-phase turbulent flows. It offers models for two alternatives to simulate particle-particle interaction.

7. Conclusion

The approach to simulate the sand particles is not yet practical and trying to simulate the dune as a fluidized model has shown, that quite a few constants are unknown.

OpenFOAM has proven to be a very potential tool, when it comes to the simulation of fluids. Whereby the design and construction of a quality grid is crucial to the success of the CFD analysis. The results of the wind flow simulation for the two and three dimensional cases have shown, that the simpleFOAM solver gives good and reliable results, after taking the boundaries from a transient solver.

In future works the flow over more complex dune structure could be validated, for example of two dunes lying close to each other. The work could also be extended for aqueous dunes or different aeolian dune shapes.

As soon as the fluid-particle coupling is more advanced, it could be applied to simulate the sand-wind interaction at least at the surface of the dune.

8. Sources

8.1. Images

[1] N.J. Balmforth (2001), Geomorphological Fluid Mechanics, Berlin: Springer-Verlag , page 414

[2] H.K. Versteeg & W.Malalasekera (2007): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, second edition, page 10

[3] H.K. Versteeg & W.Malalasekera (2007): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, second edition, page 11

[5] H.K. Versteeg & W.Malalasekera (2007): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, second edition, page 135

[6] H.K. Versteeg & W.Malalasekera (1995): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, first edition, page 140

[9] Wiggs et al. (1996), The role of streamline curvature in sand dune dynamics, page 34

8.2. Bibliography

[1] : Ian Livingstone et al. (2006): Geomorphology of desert sand dunes: A review of recent progress
[2] : H.K. Versteeg & W.Malalasekera (2007): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, second edition
[3] : H. Schlichting (1979), Boundary Layer Theory, Berlin: Springer-Verlag, page 60
[4] : Pieter Wessling (2000), Principles of Computational Fluid Dynamics, Berlin: Springer-Verlag, page 298
[5] : J. H. Ferziger, M. Peric (2001), Computational Methods for Fluid Dynamics, Berlin: Springer-Verlag, page 176
[6] : J. H. Ferziger, M. Peric (2001), Computational Methods for Fluid Dynamics, Berlin: Springer-Verlag, page 195
[7] : H.K. Versteeg & W.Malalasekera (1995): An introduction to computational fluid dynamics, Upper Saddle River: Pearson - Prentice Hall, first edition, page 64
[8] : Absi, R. (2007), Standard k-epsilon model, http://www.cfd-online.com/Wiki/Standard_k-epsilon_model
[9] : Wiggs et al. (1996), The role of streamline curvature in sand dune dynamics