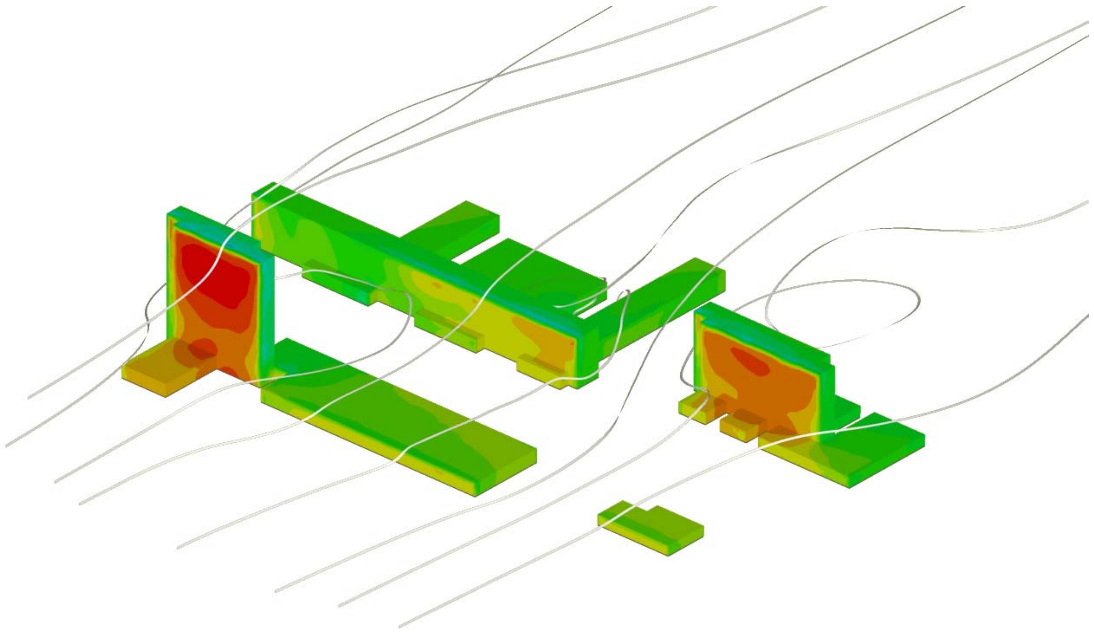
Design Tools for the Virtual Wind Tunnel

Setting up the geometry for CFD calculations



Msc thesis by R.A.G. Kerklaan
December 2006

Delft University of Technology
Faculty of Civil Engineering and Geosciences
Structural Design Lab (SDL)

Design Tools for the Virtual Wind Tunnel

Setting up the geometry for CFD calculations

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science (MsC.)

in

Civil Engineering

by

R.A.G. Kerklaan
born in Delft, The Netherlands

Delft University of Technology
Faculty of Civil Engineering and Geosciences
Building Engineering
Structural Design Lab (SDL)

Preface

This report is the result of my graduation project and marks the end of my Master of Science study at the faculty of Civil Engineering of Delft University of Technology. The report describes the research project that has been performed at the Structural Design Lab from February 2006 to December 2006. The subject of the thesis is wind engineering, with the focus on the development of a set of design tools that can be used to determine the wind load on a building or structure using Computational Fluid Dynamics (CFD). A lot of people have contributed, directly or indirectly, to this project. First of all I would like to thank the members of my graduation committee for their ideas and comments during our meetings and for their contribution to this report. I also want to thank Rogier van Nalta and Dennis Snijders for their previous work on this subject. Next to these people, I would like to thank the other graduate students of the SDL that work in room 0.72 as well. Although most work of this thesis is performed at home, I have had a pleasant time there. Above all, thanks go to my parents, brother and girlfriend for their continuous support and encouragement.

Ruud Kerklaan
Naaldwijk, December 2006

Graduation committee

Prof. dipl- ing. J.N.J.A. Vamberský (Chairman)

Organization:	Delft University of Technology
Faculty:	Civil Engineering and Geosciences
Section:	Building Technology and Process
Address:	Stevinweg 1, room 1.36 – Stevin II
	2628 CN Delft
Telephone:	+31 15 2785488
E-mail:	J.N.J.A.Vambersky@CiTG.TUDelft.nl

Prof. Ir. L.A.G. Wagemans

Organization:	Delft University of Technology
Faculty:	Civil Engineering and Geosciences
Section:	Structural Design Lab
Address:	Stevinweg 1, room 1.59 – Stevin II
	2628 CN Delft
Telephone:	+31 15 2784752
E-mail:	L.A.G.Wagemans@CiTG.TUDelft.nl

Ir. J.L. Coenders

Organization:	Delft University of Technology
Faculty:	Civil Engineering and Geosciences
Section:	Structural Design Lab
Address:	Stevinweg 1, room 1.58 – Stevin II
	2628 CN Delft
Telephone:	+31 15 2785711
E-mail:	J.L.Coenders@CiTG.TUDelft.nl

Dr. dipl-ing. S. Zlatanova

Organization:	Delft University of Technology
Faculty:	OTB Research Institute for Housing, Urban and Mobility Studies
Section:	GIS-Technology
Address:	Jaffalaan 9, room 2.240
	2628 BX Delft
Telephone:	+31 15 2782714
E-mail:	S.Zlatanova@OTB.TUDelft.nl

Dr. Ir. L.L.M. Veldhuis

Organization:	Delft University of Technology
Faculty:	Aerospace Engineering
Section:	Aerodynamics
Address:	Kluyverweg 2, room 041
	2629 HT Delft
Telephone:	+31 15 2782009
E-mail:	L.L.M.Veldhuis@LR.TUDelft.nl

Contact data student

Name:	R.A.G. (Ruud) Kerklaan
Workplace:	Delft University of Technology
	Faculty of Civil Engineering and Geosciences
	Structural Design Lab
	Stevinweg 1, room 0.72
	2628 CN Delft
Student no.:	9428437
Telephone:	06-14770763
E-mail:	R.A.G.Kerklaan@student.TUDelft.nl

Abstract

Since the introduction of new and stronger materials in building engineering in the second half of the 20th century, wind load has become an important factor for the design of buildings. Since then methods are developed to take the effects of wind into account in the design process. The building code is an appropriate method to determine the wind load on standard, simple designs. For more complex designs one can rely on wind tunnel experiments to determine the wind load. However, as they are very expensive and time-consuming, wind tunnel studies are not suitable for the early stage of the design process. Computational methods have been introduced recently to determine the effects of wind using a technique called Computational Fluid Dynamics. At the Structural Design Lab of Delft University of Technology a Virtual Wind Tunnel has been proposed that enables structural engineers to predict wind loads in the early stage of the design process using CFD. The computational Van Nalta domain is developed in which CFD calculations on building models can be performed.

CFD is a complex field that requires a thoroughly knowledge of fluid dynamics, numerical analysis and wind engineering. Performing a calculation is a difficult procedure that can be very time-consuming. The definition of geometry for CFD calculations is also complicated. To enable structural engineers to predict wind loads and compare alternative geometries in a relatively short time, the complex process of CFD has to be simplified.

The purpose of this Master's thesis is to develop a set of design tools that enables the structural engineer to setup the geometry for a CFD calculation. With this toolbox the wind load on a building or structure can be predicted and several geometries can be compared in a relatively short period without much interference of the user. This should make shape optimization with respect to wind loads possible. To predict the wind loads on a building that is placed in a built environment, design tools are developed to generate a 3D model of the environment. Other tools are developed to simplify the building of interest by reducing the input of the building design. This reduces the calculation time of the CFD software considerably. As the dimensions of the computational domain in which the calculations are performed depend on the dimensions of the research area, a final design tool is developed that generates the domain dependent on the dimensions of the research area.

It is concluded that the developed design tools work for CFD applications. Many tests with the various tools have shown that in most cases promising results are obtained and the geometry that is generated by the tools is suitable for CFD analysis. For several test cases some CFD calculations are performed in the Van Nalta domain. It was not the intention to obtain very accurate results from the calculations, as there is still quite some uncertainty about the use of CFD in wind engineering. To a large extent the accuracy of CFD calculations depends on the available computer resources and further research is required to increase the accuracy. However, following the current trends in the development of computer resources, it will probably be possible in the nearby future to calculate the wind effects on complex building models that are placed in a built environment accurately. For wind engineering problems CFD can become a valuable tool for structural engineers then.

Glossary

CFD: Computational Fluid Dynamics; the numerical calculation of flow problems using computational methods

Virtual Wind Tunnel: proposed application to indicate the wind loads on a building or structure in the early stage of the design process using CFD

Van Nalta domain: computational domain developed at the Structural Design Lab of Delft University of Technology in which CFD calculations can be performed

Rhinoceros: CAD environment in which the several design tools of this thesis are developed using the Rhinoceros Visual Basic language

GIS: Geographic Information Systems in which geographical information about the earth can be processed

Top10Vector: GIS dataset containing a digital 2D map of the Netherlands

Object-oriented models: building models that originate from object-oriented CAD systems. The models are constructed from objects that represent the elements from which a building is built of, like beams, columns, floor and walls. The models have sense for the computer

AHN: Algemeen Hoogtebestand Nederland; GIS dataset containing an elevation model of the Netherlands

Fluent: CFD software that is used in this thesis to solve numerical flow problems

Gambit: Fluent's pre-processor to setup the geometry and generate meshes for CFD analysis

Grid/mesh: subdivision of the continuous domain into discrete control volumes where the variables of the partial differential equations describing the fluid flow are calculated

Size function: function to smoothly control the growth in mesh size over a particular region of the geometry

Map scheme: mesh method that generates a regular, structured grid on a surface

Pave scheme: mesh method that generates an unstructured grid on a surface

Interval size: mesh method that divides an edge in intervals of the specified size

Cooper scheme: mesh method that extrudes the 2D grid on a surface to form a 3D grid

Tecplot: Post-processing software to analyze the results of a CFD calculation

Table of Contents

1. Introduction

The goal of this thesis is to make a significant contribution to the creation of a set of design tools that enables the structural engineer to quickly setup a CFD calculation to predict the wind loads on a building or structure in the early design stage. The design tools must be able to compare several alternative geometries in a relatively short time to optimize the design with respect to wind loads. This chapter will give the problem definition and the research goal of the Master's thesis project.

1.1 Problem definition

For a long time wind loads were not taken into account in the design of buildings. This was mainly because of the very large dead weight, low slenderness and still limited height of these structures. From the 1950's new and stronger materials were introduced in building engineering. Structural elements became lighter and more slender, which decreases the permanent vertical load. As a consequence, wind loads became more important for the design of high-rise buildings. Research into wind loads on buildings resulted in various design codes to take the effects of wind into account. These codes were mainly based on standard, rectangular building shapes and supplied coefficients for estimating pressures on a structure. However, for more complex building shapes it is not possible to determine the wind load with the existing codes. Examples of such complex buildings are the Tenerife Opera House by Santiago Calatrava, given in the left picture of Figure 1.1, and the Guggenheim Museum in Bilbao by Frank O. Gehry, given in the right picture. Experimental methods like wind tunnel tests have been developed to gain insight in the flow pattern around complex building shapes. Nevertheless, these tests are very expensive and time-consuming. This discourages designers to perform such a test in the early stage of the design process. But it is just this stage of the process where very important design decisions are made. So, more insight in wind loads and the flow pattern around a building is desired in the early design process.

Figure 1.1: It is difficult to predict wind loads on complex building shapes (Internet, [14])

Design codes and experimental methods are not fully adequate to provide complete insight in the wind effects on buildings. Another option to determine the wind loads on buildings is the use of Computational Fluid Dynamics. Within the framework of wind engineering, CFD can be described as the numerical calculation of the flow pattern using computational methods. CFD has become a very suitable tool for analyzing flows for which no analytical solutions are available. In theory CFD is able to solve the mathematical models up to a desired accuracy without the limitation of geometry. Changes in the physical model or shape of the object are relatively easy to perform, what can make CFD a very powerful tool to determine the wind effects on a building in the early design process. Analysis results can be used to optimize the design, leading to a more efficient shape or a better structure.

However, at this moment, CFD is not generally accepted yet for use in wind engineering. This is mainly because there is still contradiction on the use of CFD to predict wind effects on buildings and structures. Because there is no general approach for setting up a flow problem, results of CFD calculations are not widely accepted.

To develop a general approach for CFD in wind engineering, a computational wind tunnel has been proposed at the Structural Design Lab of Delft University of Technology. The tunnel is named *The Virtual Wind Tunnel* (Nalta, [16]) and the goal is an application that can be used in the early stage of the design process to indicate the wind loads on a building or structure using CFD. The Virtual Wind Tunnel should enable the structural engineer without much knowledge of CFD and wind engineering to predict the wind loads and compare alternative geometries in a relatively short time. The complex process of setting up a CFD simulation has to be automated as much as possible to enable the structural engineer to perform some quick calculations. In recent graduation research at the Structural Design Lab a domain has been developed by Van Nalta [16] and Snijders [23]. The so-called Van Nalta domain shows promising results for the calculation of the pressure distribution on relatively simple shapes, like cubes and cylinders. Also the grid generation process is studied extensively, what has resulted in guidelines to create proper grids for various shapes.

However, an important problem is the current capacity of desktop computers. As the Virtual Wind Tunnel is intended to be a design tool for structural engineers, most users will have a normal desktop computer to perform the calculations. Nevertheless, the computational demand for complex CFD calculations is very high and the limited power influences the ability to make accurate computations. Even for the relatively simple shapes that were investigated in the Van Nalta domain, the maximum number of cells that can be used to generate a grid was quickly consumed. However, following the current trend of the development in computer resources, increasing the computation processing speed and memory capacity continuously, these problems will certainly overcome in the nearby future.

To simulate the flow pattern around complex objects and especially built environments, the computer power is still too small. Nevertheless, anticipating on future developments, methods to automate the process of CFD to enable engineers to quickly set up some calculations still have to be developed. In the other graduation studies, only the wind effects on single objects were investigated. But the flow pattern around a building highly depends on its surroundings. One of the goals of the Virtual Wind Tunnel could be to simulate the flow pattern around a building that is placed in a built environment. To create a realistic model of the surroundings of the building of interest, a method has to be developed that can generate a 3D model of the existing built environment. The actual situation of the surroundings can be obtained from GIS technology. However, modeling complex building geometry for CFD calculations is still a major bottleneck. CAD models of a building usually contain lots of little details, like door handles, casings and railings. They are not relevant to determine the global wind loads on a structure, but will increase the calculation time of the CFD software tremendously due to the grid complexity. A method that simplifies the geometry of a building and neglect small details would be very valuable. Nevertheless, CAD data is usually not directly suited for use in CFD and problems arise in the translation of information from CAD data to CFD software. Only the surfaces that come in direct contact with the flow are required for CFD analysis. Calculations of the flow pattern around a building require airtight building models. If edges, surfaces and solids do not meet and gaps are introduced, wind can flow through the model and problems will arise as the meshing algorithms require a perfect model description as input.

1.2 Research goal

The complex process of CFD requires a thoroughly knowledge of wind engineering, fluid dynamics and numerical analysis of the user. Setting up the geometry for CFD calculations of complex buildings in a built environment is also a very difficult process. To enable structural engineers without much knowledge of fluid dynamics to predict wind loads and compare alternative geometries in a relatively short time, the process of CFD and the input of geometry has to be simplified as much as possible. With a manageable toolbox containing a set of design tools, the process of setting up the geometry and performing the simulations can be automated.

Anticipating on future developments of computer resources, the goal of this Master's thesis is to develop a set of design tools that enables the structural engineer to quickly setup a CFD calculation. With this toolbox the wind loads on a building or structure can be predicted and several geometries can be compared in a relatively short time without much interference of the user. The focus of the design tools lies on the creation and preparation of the geometry and the setup of the CFD calculations. To predict the wind loads on a building that is placed in a built environment, the first design tools will generate a 3D model of the built environment for a certain location using GIS technology. To limit the calculation time of the CFD software, another design tool will simplify the various CAD models of the building of interest by reducing the input of the building design. Because the inner geometry that is also modeled frequently is not relevant to determine the wind loads, the design tool will remove the eventual inner geometry. In case of eventual gaps in the model, the tool will also make the models airtight. After the simplified building model and the model of the built environment are joined, it can be translated to the CFD software. Possible problems that occur in this translation process will be investigated. The model of the research area is then placed in a computational domain that bounds the flow problem. As the dimensions of this domain depend on the dimensions of the research area, a final design tool will automatically create the right computational domain for a certain research area. After the geometry is setup in the CFD software, the complex process of meshing the geometry will be investigated. When a general method is developed to mesh the computational domain and the research area, finally some calculations will be performed for several test cases.

Chapters 2, 3, 4 and 5 discuss the required theory about computational fluid dynamics, wind engineering, GIS technology and CAD. Additional information about these topics is included in Appendix A. Chapter 6 gives the strategy for using the design tools to setup the geometry for a CFD calculation. In Chapter 7 the development of the various tools is discussed extensively. All developed methods, procedures and scripts originate from own ideas and work, except where otherwise mentioned. In Chapter 8 several CFD calculations with various building models are discussed in detail and a comparison is made with the building codes. In Chapter 9 an evaluation is given of this Master's thesis and in Chapter 10 finally the conclusions and recommendations are discussed.

2. Computational Fluid Dynamics

2.1 Introduction

In many branches of engineering there has to be an understanding of the motion of fluids. Examples are the aircraft and automobile industries, where the aerodynamics of airplanes and cars must be determined. One way to obtain this aerodynamic information is the use of wind tunnel tests. Such tests might consume many hours of preparation and wind tunnel time and are very expensive. Another method to solve fluid dynamic problems is the use of CFD.

Computational Fluid Dynamics (CFD) can be described as the use of computers to produce information about the flow pattern in given situations (Shaw, [22]). It is in general a technique in which equations describing the fluid flow are solved numerically on a computer. The flow volume is divided in a finite amount of cells in which physical quantities like velocity and pressure are supposed to be constant. Subsequently the so-called Navier-Stokes equations are solved iteratively for each cell. To be useful, the results must be a realistic simulation of the fluid in motion. If the results are realistic depends on the problem being simulated, the software being used and the skill of the user. Users of CFD must be very familiar with the flow problem they wish to simulate.

CFD is used in a variety of industrial sectors nowadays, including aerospace, defense, automotive, physics, engineering and computer science. For example, predictions can be made of the following:

- Lift and drag of an aircraft
- Flames in burners
- Air flow inside internal combustion engines
- Dispersion of pollutants into rivers and oceans
- Flow of cooling air inside electrical equipment

In this chapter the process of CFD is discussed extensively. In Appendix A.1 the theory behind fluids in motion is included as background information. Appendix A.2 includes some aspects of turbulence modeling.

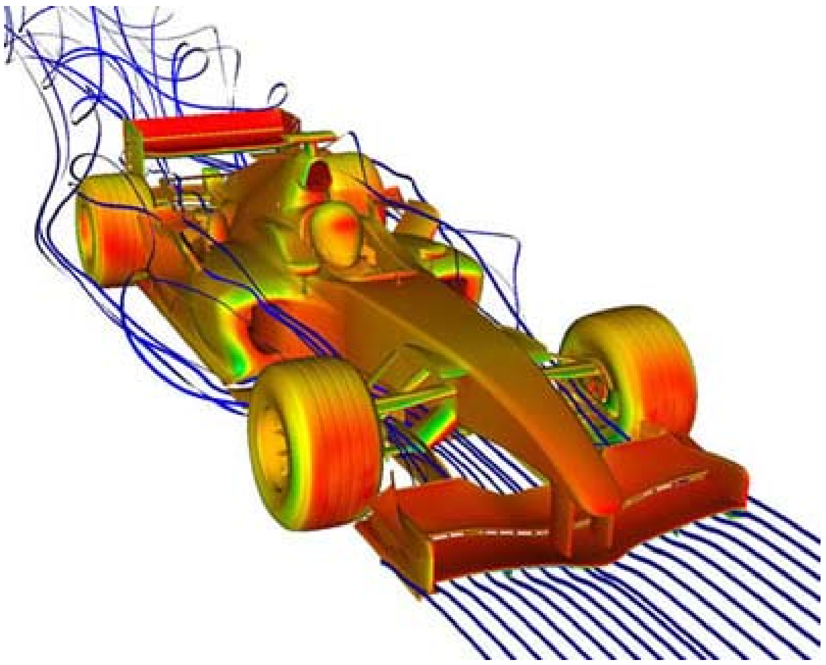


Figure 2.1: CFD simulation of a Formula 1 car in Fluent (Internet, [9])

2.2 The process of CFD

A fluid flow problem can be described by a series of partial differential equations that govern the flow. These partial differential equations can be discretized by replacing the derivates in the equations by discretized algebraic forms. In this way a numerical analogue of the partial differential equations is built that can be solved to obtain the flow variables at discrete points. The boundary conditions and the initial conditions that are applied determine the flow problem. To produce a solution from the numerical analogue many equations have to be calculated, which requires a broad number of calculations to be carried out. Computers are the ideal tool for this.

The essential steps in a CFD analysis, which are formulated by Arif [3], are presented in Figure 2.2. The analysis starts with the description of the geometry. The geometry can be generated by a CAD program or with a pre-processor. This is a special program within the CFD software to generate geometry. The second step is the generation of a grid or mesh on the geometry on which the Navier-Stokes equations can be discretized. The third step is the discretization of the partial differential equations. These continuous equations can be approximated with a system of algebraic equations for the variables at discrete locations. In the following step a turbulence model has to be selected to takes turbulence into account. In the next steps the flow properties are specified. The boundary conditions, initial conditions and some fluid properties specify the flow problem that has to be solved. Once the calculation is setup, the numerical solution can be computed using a solver. The set of approximating equations is solved numerically for the flow field variables at each point or cell. Finding a solution is an iterative process. The calculation is repeated until a satisfactory solution is found. In the last step the results have to be analyzed to see if the solution is physically reasonable. Using graphs or contour plots, information about the flow parameters can be presented in a visual form.

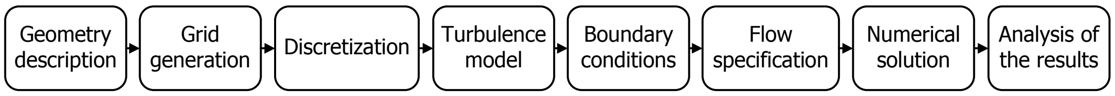


Figure 2.2: Essential steps in a CFD analysis

In most CFD packages three main pieces of software are provided to assist the user in carrying out the analysis process. These pieces are:

- A pre-processor
- A solver
- A post-processor

All tasks that take place before the calculation of the numerical solution is started are called pre-processing. The most difficult task in the pre-processing phase is the generation of the grid or mesh. Before solving the numerical equations, all relevant data that has been defined by the pre-processor must be given to the solver. When the equations are finally solved, a large number of variables are calculated in the flow domain. Computer graphics is often the only way to analyze the data written by the solver program. The post-processing program is used to display the results.

In the following paragraphs the process of CFD is discussed more extensively.

2.2.1 Geometry description

To simulate the flow around an object a model has to be built of the object. This model can be drawn in a CAD program or with a pre-processor. For simple models the pre-processor can be used, but for complex geometries it is recommended to use a CAD program or object-oriented and parametric modelers. The geometry is defined by the coordinates of the object as well as any outer boundaries. Only the exterior surfaces of the objects to be modeled are needed, because only these surfaces come in contact with the flow. Interior surfaces do not have to be drawn and can be deleted. The amount of desired detail will have consequences for the complexity and amount of computer power required to obtain reliable results.

2.2.2 Grid generation

One of the most important steps in the CFD process is the generation of a grid or mesh throughout the flow domain. When a partial differential equation is set to be solved, a subdivision of the continuous domain into discrete volumes where the variables are calculated is required. This discretization defines a computational grid and the process to perform this discretization is called *grid generation*. The results of the CFD analysis highly depend on the grid type and amount of cells used. The accuracy increases when more cells are used. However, more cells lead to longer computation time and computer resources limit the amount of available cells. The grid generation phase is the phase of the analysis process that determines the total time required to obtain results from a calculation too a large extent. (Thompson et all, [24]). The grid itself however will be influenced by the used discretization method.

Grids can be divided into two types: structured grids and unstructured grids. In a structured grid all elements are similar in shape and placed in a regular way. The grid is relatively easy to generate and has a clear, transparent structure. For complex geometries it is however very difficult to generate a mesh using a structured grid. In an unstructured grid, the elements vary in topology and size. There is no global structure of the elements. It is more difficult to generate these grids, but as the complexity of the geometry increases, the grid quality remains high. However, the structure of the grid is less transparent than a structured grid and the computational demand is higher. Unstructured grids are more suitable for local mesh adaptation, where the grid is refined only where necessary. Figure 2.3 and Figure 2.4 give examples of structured and unstructured grids.

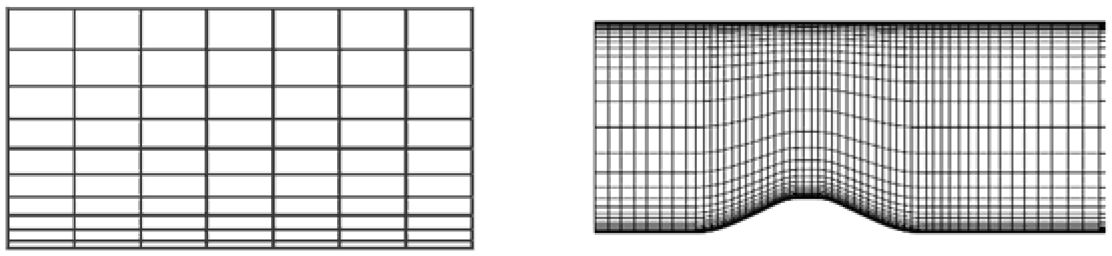


Figure 2.3: Examples of structured grids (Apsley, [2])

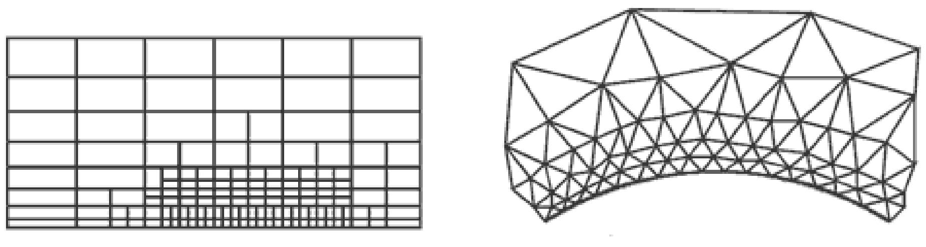


Figure 2.4: Examples of unstructured grids (Apsley, [2])

Grids are made of cells that can have any shape, but in practice the shape of the cell is similar to one of the basic shapes given in Figure 2.5.

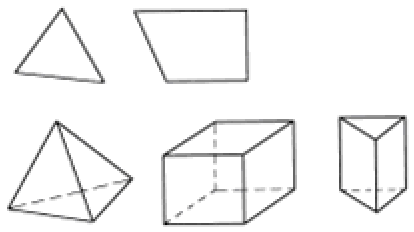


Figure 2.5: Common used cell shapes for meshing 3D geometry (Snijders, [23])

The grid should be more refined in those regions where high gradients of the flow occur. The difficulty now is to know where these regions are and how fine the grid should be. The positions of the critical areas can be assumed manually to build a grid taking this into account. Doing a simulation with such a grid may help to determine the actual positions of the regions of high flow gradients. This technique is known as *adaptive meshing* (Shaw, [22]). Depending on the software used, the grid refinement can also be done automatically, where the program decides where to refine the grid.

2.2.3 Discretization

To obtain a numerical solution of partial differential equations, discretization of the equations is required. The continuous partial differential equations describing the fluid flow (the Navier-Stokes equations) are approximated by a system of discrete algebraic equations (Thompson et all, [24]). These equations are then solved numerically. In the continuous domain each flow variable is defined at every point in the domain. In the discrete domain each flow variable is defined only at discrete locations, which are determined by the applied grid.

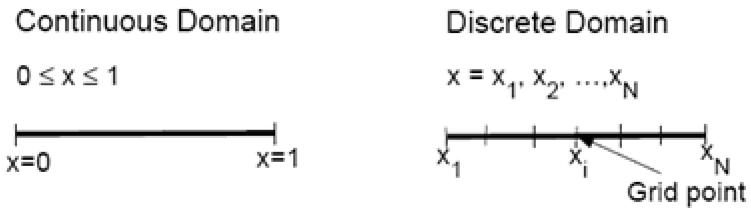


Figure 2.6: Continuous vs discrete domain (Internet, [9])

A variety of techniques can be used to perform the discretization. Three of the major methods are the finite difference method, the finite element method and the finite volume method. In this thesis the CFD software Fluent will be used. Fluent uses the finite volume method as discretization method.

Finite Volume Method (FVM)
The finite volume method is the most popular numerical discretization method used in CFD. The solution domain is subdivided into a finite number of small control volumes by a grid, which defines the boundaries of the control volumes. The governing partial differential equations are converted into a numerical form by integration over the control volume. The variability in the flow is balanced through the volume and considered to be constant in each volume. The solution variables of a volume are stored at the center of the volumes in so-called computational nodes. The finite volume method is not limited to a grid type; both structured and unstructured grids can be used. The finite volume method is explained in Figure 2.7.

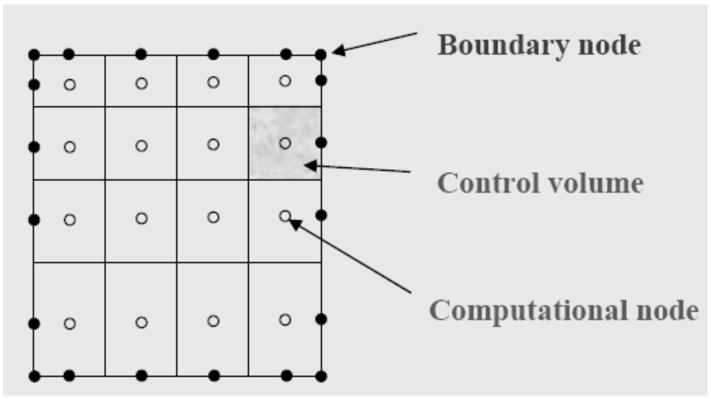


Figure 2.7: Discretization according to the Finite Volume Method (Apsley, [2])

2.2.4 Selection of a turbulence model

Depending on the investigated flow problem and available computer capacity, an appropriate choice needs to be made for the turbulence model. Distinction is made between the Large Eddy Simulation (LES) method and the Reynolds Averaged Navier-Stokes (RANS) method with its Reynolds stress transport models and a large number of variants of the κ-ε models. RANS modeling is much faster than LES but it is assumed to be unfit when high accuracy is needed. Additional information about turbulence modeling is given in Appendix A.2.

2.2.5 Boundary conditions and flow specification

The next step in the CFD process is to setup the boundary conditions of the variables to be calculated. Examples of boundary conditions are inlet, outlet, wall and far-field conditions. Figure 2.8 shows an example of the boundary conditions for the two-dimensional flow over a block. At the inlet, fluid enters the domain and usually a prescribed velocity is used as boundary condition. At the outlet, fluid leaves the domain and a prescribed pressure is usually used as boundary condition. For viscous fluids, usually the no-slip boundary condition is used for walls, which means that there is no velocity right at the surface. Far away from the block the fluid flows with a free-stream velocity V_∞.

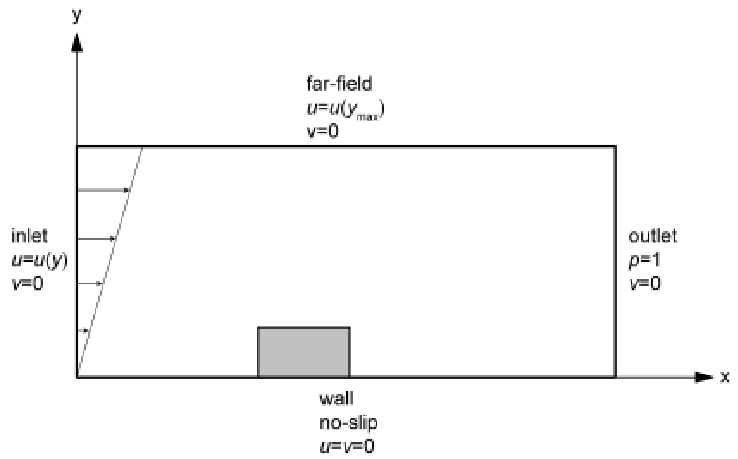


Figure 2.8: Boundary conditions for a 2D flow over a block (Nalta, [16])

The boundary conditions, together with the initial conditions and some fluid properties such as velocity, density and viscosity, specify the actual flow problem that is to be solved. The number and type of boundary conditions must accord with the governing equations of the flow. The initial conditions specify the number of iterations required to get a solution.

2.2.6 Calculation of the numerical solution

In this step the set of approximating equations are solved numerically in an iterative process for the flow field variables at each point or cell. The calculation is repeated until a solution with a sufficient and user-defined accuracy is found. There are many methods available to calculate the numerical solution, ranging from fast and relatively inaccurate calculations to slow and very accurate calculations.

2.2.7 Analysis of the results

Huge amount of data are produced after the simulation. The results have to be checked to see if the solution is physically reasonable. To interpret the results, information about the flow parameters can be presented in a visual form like graphs or contour plots.

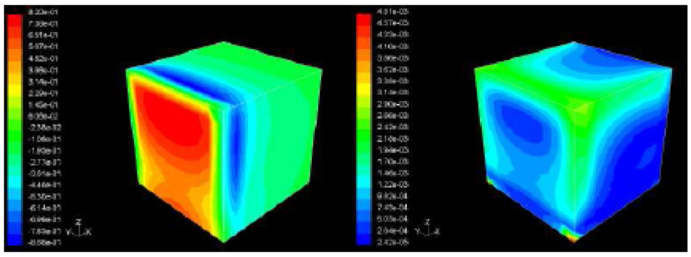


Figure 2.9: Visual representation of CFD results (Snijders, [23])

2.3 Fluent

There are several programs available for numerical flow calculations. They range from very specialized programs, which solve problems within strict boundaries, to extensive commercial packages that solve the complete Navier-Stokes equations in three dimensions. The CFD software that will be used during this Master's thesis is Fluent. Fluent is a commercial CFD software package developed by Fluent, Inc. (Internet, [9]) that enables engineers and designers to simulate fluid flow, heat and mass transfer and lots of related phenomena like turbulence.

As in most commercial CFD packages the finite volume method is used in Fluent to discretize the partial differential equations. The domain can be discretized with the structured as well as the unstructured grid technology. The grid can consist of elements in a variety of shapes, from 2D rectangles and triangles to 3D tetrahedrals, prisms and pyramids. These elements are created using a mesh generation module called Gambit, which is also included in Fluent. The pre-processor Gambit is based on ACIS geometrical libraries, which uses Non-Uniform Rational B-Splines (NURBS) to describe geometry. It can also be used to read CAD files and adapt the imported geometry for CFD analysis. Fluent's post-processing tools can be used to generate meaningful graphics and animations.

In Fluent the Reynolds stress transport models and several versions of the popular κ-ε model are available to deal with turbulence. Also the Large Eddy Simulation (LES) method is supported, what makes it a very powerful package for wind engineering in the nearby future considering the increasing computer power and decrease in computer cost.

3. Wind Engineering

3.1 Introduction

Wind is a natural motion of the air characterized by a direction and velocity. It originates from the irregular heating of the earth by the sun, which causes differences in air pressure. As a consequence, large areas with dimensions of some hundreds of kilometers can arise where the air pressure is relatively high or relatively low. These differences in pressure, together with the rotation of the earth and friction of the air with the earth's surface, cause wind. The air flows from areas with a high air pressure to places with a low air pressure.

Until the second half of the 20th century, wind loads were hardly taken into account in the structural design of buildings. Because of the very large dead weight of these structures at that time and the limited height and slenderness, wind loads did not have a significant influence on the structure. From the 1950's new and stronger materials were introduced in building engineering. Structures became lighter and more slender, reducing the member sizes and their stiffness. Buildings and structures became higher and more sensitive for wind from now on, which led to more research in wind loads and finally resulted in several design codes for wind loads on buildings and structures. With these codes it is possible to determine the wind loads for common building shapes on the basis of various coefficients.

For the simple shapes of the first high-rise buildings, the design codes to determine the wind load were sufficient. But for more complex building shapes it is not possible to determine the wind loads with the existing codes. Wind tunnel tests have been introduced to predict the flow pattern around buildings and structures by experiment. With these tests it is also possible to determine the wind loads on buildings with complex shapes. The latest technology to investigate the wind loads is the use of Computational Fluid Dynamics. CFD can form an alternative for the expensive and time consuming wind tunnel tests and could also be used to calculate flow problems for the built environment, like pollution dispersion and ventilation.

Figure 3.1: Predicting wind loads on complex buildings can be very difficult (Internet, [8])
Left picture: Turning Torso in Malmo, Sweden, by Santiago Calatrava
Right picture: Walt Disney Music Hall, Los Angeles, by Frank O. Gehry

When wind hits a building, a part of the kinetic energy will be transformed into forces and pressures on that building. Besides wind direction and velocity the pressure is determined by the shape of the building and type of landscape. For the calculation of the wind pressure on a structure the following equation can be used (Cauberg, [6]):

$$p_w = \frac{1}{2} \cdot \rho \cdot C_p \cdot U^2 \quad [N/m^2] \qquad (3.1)$$

Where: ρ = density of the air (1,2 kg/m^3)
C_p = coefficient, depending on the shape of the object
U = wind velocity

An obstacle like a building or structure in the flow domain influences the flow pattern in the domain. At the windward side of the building the air is slowed down and a part flows around and over the building. Because more air has to pass here in the same time, the velocity will increase. The other part of the incoming air flows down generating a vortex in front of the building. A lot of wind hindrance will occur in this area. At the top of the windward side of the building the air separates because of the sharp edge of the roof. This develops a wake at the leeward side of the building. The wind profile of the wake with high turbulence and velocity gradients is very divergent from the undisturbed profile. As a consequence of the flow pattern around a building an area with relatively high pressure will develop at the windward side of the building. At the leeward side of the building an area with relatively low pressure will develop where the air velocity is usually low.

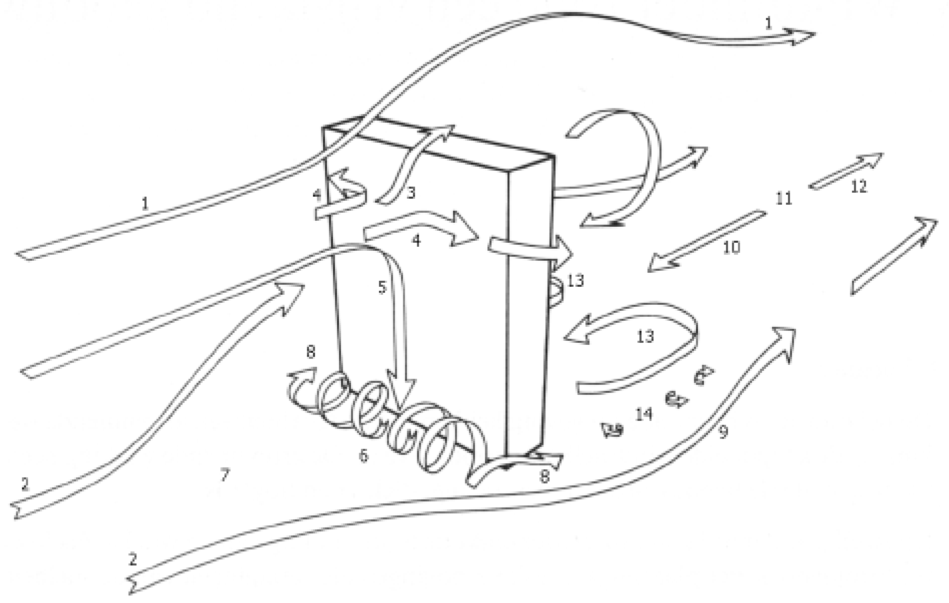


Figure 3.2: Flow pattern around a high-rise building (Cauberg, [6])

The wind velocity varies in time. The Beaufort scale divides the wind velocity in 12 classes, varying from windless until hurricane velocities, and gives a relation between wind velocity and wind strength (Cauberg, [6]). In a wind rose the expected wind strength over a year is given for the various wind directions using charts. The length of a chart reflects the frequency in which the wind strength will occur. The frequency zones are given by circles with an increasing chance of occurrence. The thickness of a chart gives the wind force on Beaufort scale. Figure 3.3 gives the wind rose of Vlissingen for a year. It is to be expected that most of the wind comes from the south-west direction.

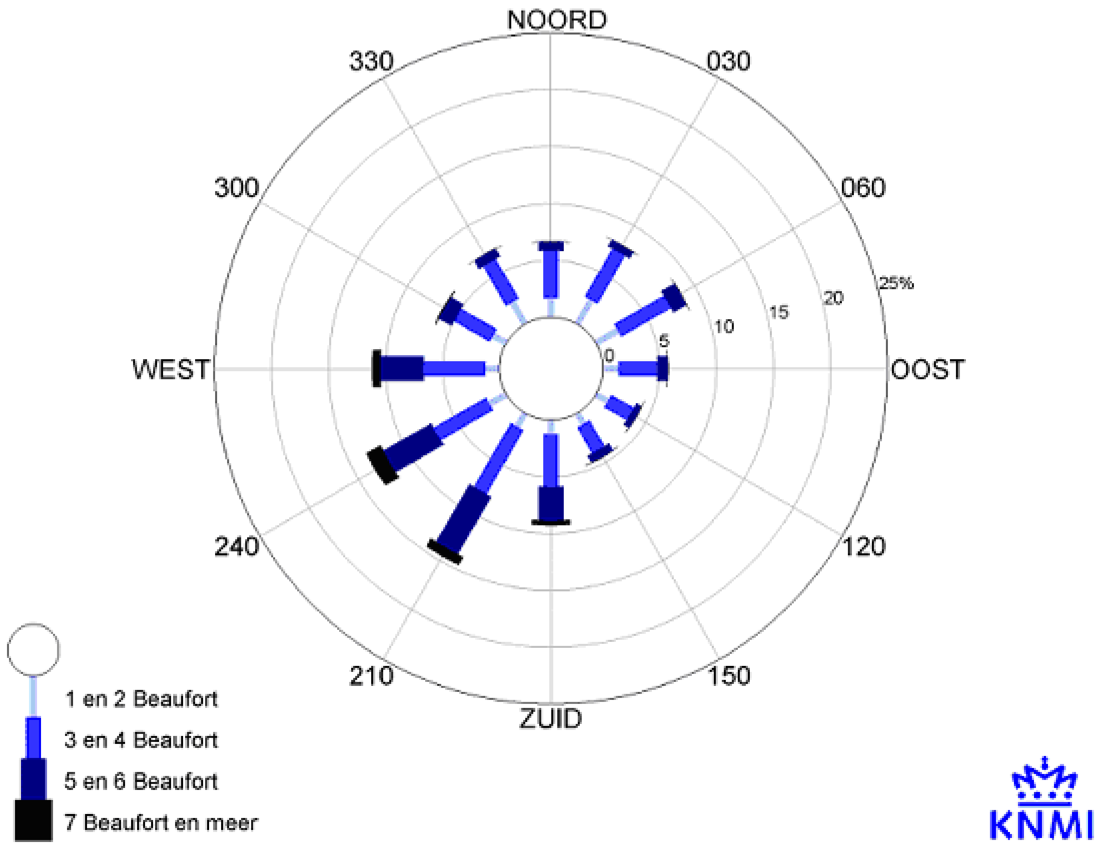


Figure 3.3: Wind rose of Vlissingen for a year (Internet, [7])

3.2 Wind standards

Wind forces are very important for the structural design of high-rise buildings. The size of these forces grows exponentially with the height. Fluctuations in wind velocity result in a dynamic load on a building. Stiffness and mass determine the influence of the wind loads on the construction. Because of the development of light and flexible structural materials the influence of wind is increasing. Building codes have been the most commonly used design tool for structural engineers. The codes are based on wind studies, are easy to use and can be applied to a wide range of cases. Most countries have their own building code. However, the European Eurocode is being developed at the moment to replace several national codes. In this paragraph the procedure to calculate the wind load according to both the Dutch code and the Eurocode is summarized.

3.2.1 Dutch code

Wind is considered first as a load case in the '*Technische Grondslagen voor bouwconstructies*' (TGB) from 1955. In the TGB 1955 the wind loads are determined by multiplying a pressure with coefficients for under- or overpressure. TGB 1972 uses statistic variables for the first time. The pressure is derived from an hour-average wind velocity, exceeded once in five years with a chance of 50%. The current Dutch codes for the calculation of wind forces on buildings and structures are NEN 6700 and NEN 6702. The code NEN 6700 gives the basis demands for the structural safety of buildings and structures, during the construction phase as well as the utilization phase. The code NEN 6702 gives a calculation model and form factors to determine the wind loads. These form factors are derived from wind tunnel tests and relate the shape of a building with the amount of wind pressure. However, for complex building shapes the tables of NEN 6702 are often not valuable. For those cases wind tunnel tests will be necessary to determine the wind loads.

Some important aspects of NEN 6702 towards wind loads are (Geurts, [10]):

- The Netherlands are divided in three geographic areas with a representative, constant wind velocity, based on the variation of wind velocity and geographical differences; Distinction is made between built and unbuilt surroundings, leading to six standardized wind climates
- The basis for the wind velocity is an extreme hour-average velocity that will be exceeded once in 12,5 years. On average only one of the four wind directions is important for a construction. Therefore the reference period for wind loads is shortened from 50 years according to the TGB 1972 to 12,5 years.
- Figures are developed for the form factors, where distinction is made in form factors for an entire building and local form factors

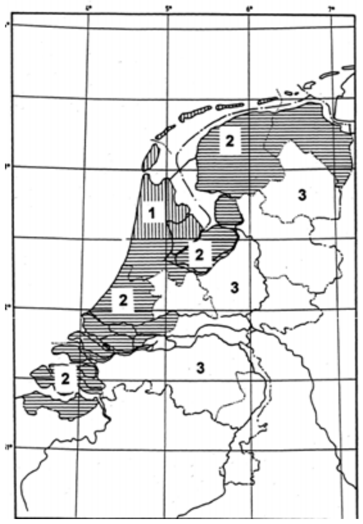


Figure 3.4: The Netherlands can be divided in three wind areas (NNI, [18])

NEN 6702 prescribes the wind load on a building or structure with the following equations:

$$p_{rep} = p_w \cdot C_{dim} \cdot C_{index} \cdot \phi_1 \tag{3.2}$$

$$F_{index} = A \cdot p_{rep} \tag{3.3}$$

Where:	p_{rep}:	representative wind pressure
	p_w:	extreme wind pressure, depending on the global and local wind climate
	C_{dim}:	relation between wind gusts and building dimensions
	C_{index}:	form factor for pressure, suction and friction
	Φ_1:	dynamic factor
	F_{index}:	representative wind force on a surface
	A:	surface exposed to the representative wind pressure

Global wind climate

The extreme wind pressure p_w in the above mentioned equation is influenced by the global wind climate. This climate can be described on the basis of observations of meteorological institutes. According to NEN 6702 [18] the Netherlands can be divided in three areas, where for each area an extreme hour-average wind velocity is determined with a reference period of 12,5 years.

Local wind climate

The height of the layer in which the flow pattern is influenced by the roughness of the surface is called the *atmospheric boundary layer*. It is the layer in which the interaction between the wind and structures takes place. Above a smooth surface the atmospheric boundary layer will be smaller than above a city with many high-rise buildings. Within the boundary layer the wind velocity varies over the height. As can be seen in Figure 3.5, a different roughness of the terrain leads to a different profile of the boundary layer. The air velocity at pedestrian level for example will be smaller above a rough surface than above a smooth surface. As a consequence, higher wind loads will be measured on structures in an open country than on structures that are placed in a built environment.

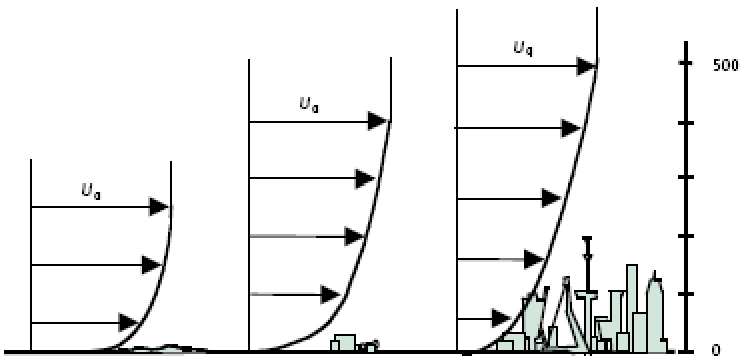


Figure 3.5: Wind velocity profile for different terrain roughness (Woudenberg, [29])

The wind pressure p_w of Equation (3.2) is on local level influenced by the surroundings. High above the ground air can move freely due to differences in air pressure. Near the ground the air is slowed down by the local environment and turbulence is increasing. The influence of the surroundings on the average wind velocity can be described by the wind velocity profile. The profile describes the development of the wind velocity over the height. Obstacles on the surface influence the profile: above a rough surface the air is slowed down more and the atmospheric boundary layer will be higher.

Within the atmospheric boundary layer, the variation of the mean wind velocity with the height can be described by a logarithmic velocity profile. This profile is used in NEN 6702 as well and can be used up to 150 m. The profile is given by:

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z-d}{z_0}\right) \tag{3.4}$$

Where:	u^*:	friction velocity
	κ:	Von Karman constant, which is 0.42
	z:	height above the ground
	z_0:	roughness height of the terrain
	d:	displacement length

The friction velocity u^* is a measure for the exchange of the horizontal wind velocity from a high altitude to a low altitude. The roughness height z_0 is a factor that represents the roughness of the ground surface and varies from 0.0002 above sea to 2.0 above cities. The displacement length d is only used for dense areas, like city centers. In such areas the flow pattern above buildings and other obstacles in the area has no relation with the flow pattern between the obstacles. The displacement length defines a fictive ground level from which the logarithmic profile is valid. In the layer below d the velocity field is undefined and very turbulent. Figure 3.6 gives the characteristics of the atmospheric boundary layer.

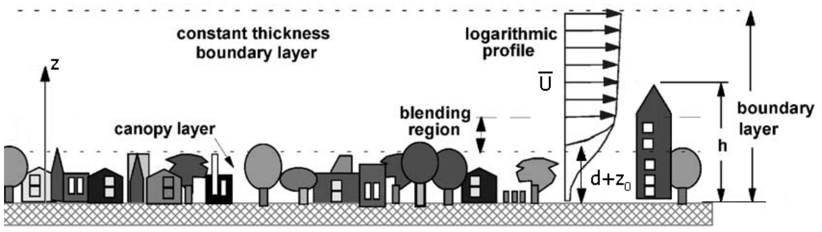


Figure 3.6: Characteristics of the atmospheric boundary layer (Nalta, [16])

Roughness

Obstacles on the earth determine the roughness of the surface. In wind engineering the various types of landscapes are usually classified in roughness classes. A high roughness class of 3 or 4 refers to landscapes with many trees and buildings, while a sea surface is classified as 0. In general it can be said that the rougher the surface, the larger the turbulence will be. Turbulence can be described as the mean deviance in wind velocity compared to the hour-average wind velocity (Cauberg, [6]). The turbulence is largest near the surface and is decreasing with increasing height. In a built environment the wind loads on a specific height are lower than in an open area. But because of a larger turbulence the dynamic part of the wind load will be larger in a built environment. In norm tables the extreme wind pressures for several heights above the ground are prescribed.

Form factors

To what extent the wind results in forces on a building depends on the shape and dimensions of the building. They determine the flow pattern around a building and as a consequence the pressure, suction and friction. NEN 6702 gives form factors for the response of the building. These factors give the relation between the shape of a building and the total amount of wind pressure and wind suction on a building. However, most buildings do not satisfy the given shapes in the norms. For buildings with a shape divergent from the code wind tunnel research remains the appropriate method to determine the form factors accurately.

In the European norm Eurocode 1 tables are given to take the influence of rounded corners and the slenderness of a building into account. The slenderness is an important factor that is not considered for the form factors in the NEN 6702 code. However, the slenderness is very important for high buildings because it determines to what extent the wind will flow around a building (Woudenberg, [29]). For high buildings wind tunnel research is a very appropriate method to obtain wind form factors and wind loads.

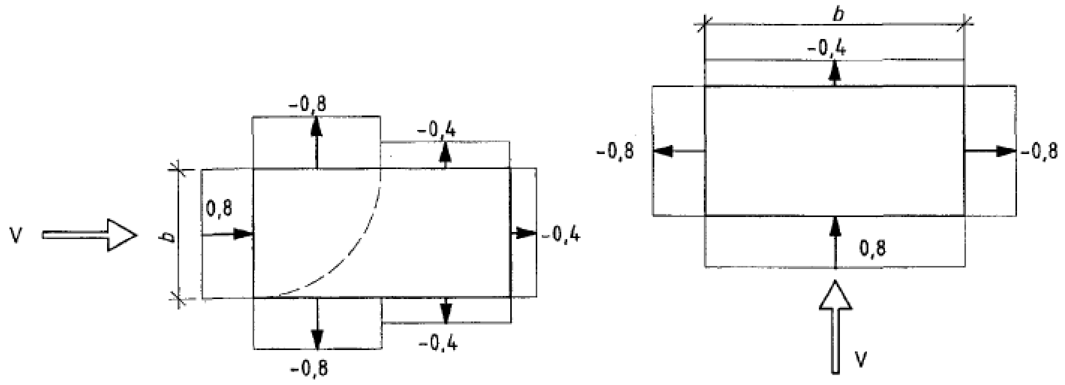


Figure 3.7: Form factors for pressure, suction and friction for buildings with a rectangular floor plan (NNI, [18])

3.2.2 Eurocode

In 1990 the European committee initiated to develop a uniform, technical code for the design of buildings and infrastructures. This Eurocode will replace the several national codes of the EU member states. It provides common structural design rules for the design of structures and their components. The Eurocode is a collection of the following codes (NNI, [17]):

- Eurocode 0: general rules for calculations according to the limit states method
- Eurocode 1: gives design values for load, such as dead load, fire, snow and wind

There are special codes for the various building materials:

- Eurocode 2: concrete structures
- Eurocode 3: steel structures
- Eurocode 4: composite structures
- Eurocode 5: timber structures
- Eurocode 6: masonry structures
- Eurocode 9: aluminium structures

And finally there are two separate codes for:

- Eurocode 7: soil mechanics
- Eurocode 8: seismic loads

In 2006 these Eurocode parts will be published and in 2010 they are expected to be fully implemented to replace all national codes. For the calculation of wind loads the Eurocode 1 part 1-4 is used. The code includes design rules for whole structures, parts of a structure and elements attached to a structure. It is applicable to buildings and structures with heights up to 200 m.

In the Eurocode, the wind action is represented by a simplified set of pressures and forces whose effects are equivalent to the extreme effects of the turbulent wind. The Eurocode prescribes the wind load on a building or structure with the following equation:

$$F_w = q_{ref} \cdot c_e(z_e) \cdot c_d \cdot c_f \cdot A_{ref} \tag{3.5}$$

Where:	q_{ref}:	The reference mean velocity pressure
	c_e:	The exposure coefficient. This coefficient takes the wind profile into account
	z_e:	The reference height. Usually this is the height of the building or structure
	c_d:	This coefficient takes the dynamic effects of the response of the building or structure into account
	c_f:	The force coefficient. This coefficient takes the geometry of the building or structure into account
	A_{ref}:	The reference area of the building or structure that is subjected to the wind

The several factors of this equation are discussed on the following pages.

The reference mean velocity pressure

The wind velocity and the velocity pressure are in the Eurocode composed of a mean and a fluctuating component. The mean wind velocity is the characteristic 10 minutes mean wind velocity at 10m above the ground in the open field, irrespective of wind direction and time of the year. The wind forces are based on this mean wind profile and depend on the reference mean velocity pressure:

$$q_{ref} = \frac{\rho}{2} v_{ref}^2 \tag{3.6}$$

Where:	ρ:	the air density
	v_{ref}:	the reference wind velocity

The reference wind velocity can be calculated from:

$$v_{ref} = c_{dir} \cdot c_{season} \cdot v_{ref,0} \tag{3.7}$$

Where:	c_{dir}:	the directional factor
	c_{season}:	the season factor
	$v_{ref,0}$:	the fundamental value of the reference wind velocity

The reference wind velocity is calculated from a basic value multiplied by factors for the direction of the wind and the season. The season factor can be used for temporary structures.

The exposure coefficient

The wind velocity profile is taken into account with the exposure coefficient c_e. The exposure coefficient is defined by a roughness coefficient and a topography coefficient, which all depend on the reference height z_e:

$$c_e(z_e) = c_r^2(z_e) c_t^2(z_e)\left[1 + 2gI_v(z_e)\right] \tag{3.8}$$

Where:	g:	the peak factor with value 3,5
	$I_v(z_e)$:	the turbulence intensity
	$c_r(z_e)$:	a roughness coefficient
	$c_t(z_e)$:	a topography coefficient

The turbulence intensity is defined as the ratio of the maximum or minimum wind speed and the mean wind speed at a certain height. The influence of hills and cliffs for a certain mean wind speed is taken into account by the topography coefficient. The roughness coefficient takes the roughness of the terrain into account.

Dynamic effects

The effects of the vibrations of the structure due to turbulence are taken into account with the dynamic factor c_d. It is obtained from the following expression:

$$c_d = \frac{1 + 2 \cdot k_p \cdot I_v(z_e) \cdot \sqrt{B^2 + R^2}}{1 + 7 \cdot I_v(z_e) \cdot \sqrt{B^2}} \tag{3.9}$$

Where:	k_p:	the peak factor defined as the ratio of the maximum value of the fluctuating part of the response to its standard deviation
	I_v:	the turbulence intensity, which depends on the reference height
	B^2:	the background factor; allows for the lack of full correlation of the pressure on the structure surface
	R^2:	the response factor; allows for turbulence in resonance with vibration

The force coefficient
The force coefficient takes the geometry of a building into account. For a rectangular floor plan it is defined by:

$$c_f = c_{f,0} \cdot \psi_r \cdot \psi_\lambda \tag{3.10}$$

Where:	$c_{f,0}$:	the force coefficient of rectangular sections with sharp corners
	ψ_r:	the reduction factor for square sections with rounded corners
	ψ_λ:	the end-effect factor for elements with free-end flow

The various parameters are based on results of wind tunnel tests.

The reference area
The reference area depends on the length and width of the structural element being considered and can be determined with the following expression:

$$A_{ref} = l \cdot b \tag{3.11}$$

3.3 Wind tunnel tests

For complex buildings that are placed in a diverse built environment it is very hard to judge the flow pattern beforehand. Wind tunnel tests are therefore often necessary. The current form factors in NEN 6702 and the Eurocode are also based upon wind tunnel tests. The wind tunnel is essential for situations where the codes do not foresee. For buildings with a non-rectangular plan, buildings with rounded edges, complex details and buildings in a complex environment, the standards do not foresee and wind tunnel research is necessary.

With wind tunnel tests a complex built environment can be investigated by making a scaled model of the buildings and expose it to wind. The required wind is generated on scale with ventilators. To develop a velocity profile comparable to the atmospheric boundary layer, the air is led over a foreland before it reaches the scaled model. The foreland is an area where the roughness of the terrain in front is simulated. This terrain can be a flat landscape, open water or a built environment. The roughness of the terrain also influences the amount of turbulence. Turbulence is generated by blowing over a row of swords and a foreland of blocks. Figure 3.8 shows the setup of a wind tunnel test with the scaled model at a rotating platform, a foreland of blocks and a row of swords.

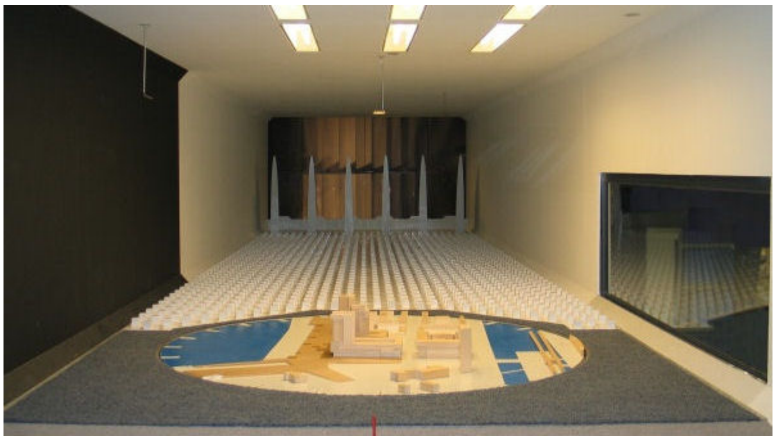


Figure 3.8: Setup of a wind tunnel test (Van der Ven, [28])

In wind tunnel studies most often an area with a radius of 300 m around the building or region of interest is modeled. By placing the scaled model on a rotating platform all wind directions can be investigated. The amount of detail required in the model depends on the purpose of the wind tunnel research. For the determination of forces on the main structure, small details are less relevant than for the investigation of local forces at fasteners of façade elements for example.

During a wind tunnel test all forces have to be determined that can occur during the lifespan of a building. According to CUR Aanbeveling 103 [27], two measurements have to be carried out in order to cover future changes of the surroundings that can influence the wind loads on a building in a negative way. Such a situation can occur when some buildings of the environment are demolished for example. In the first measurement, all surrounding buildings are modeled according to the actual situation. In the second measurement, the surrounding buildings are chopped off to a maximum height of 15 m. The maximum values obtained from these two measurements are normative.

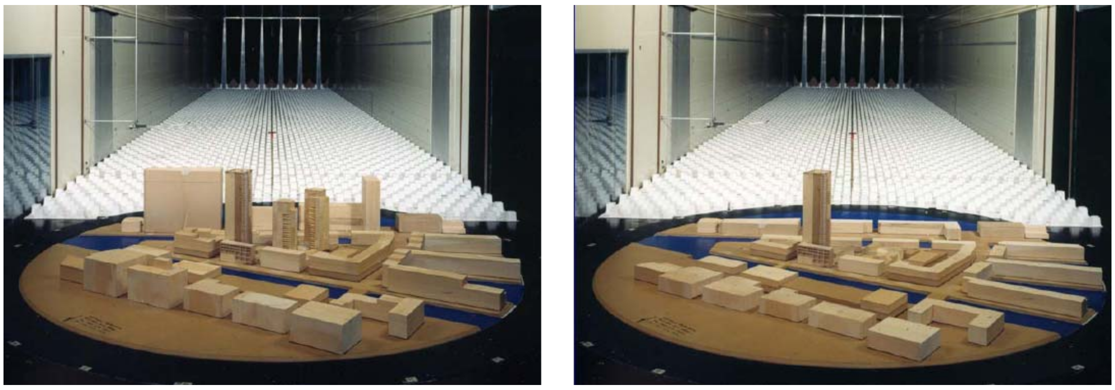


Figure 3.9: In wind tunnel tests two measurements have to be carried out: one measurement where the surroundings are modeled as the actual situation and another measurement where the surrounding buildings are chopped off (Geurts, [11])

For wind tunnel tests the model of the reality must satisfy some modeling rules. In a wind tunnel, vortexes release at a different location from rounded buildings than in reality. This is caused by the small Reynolds numbers that are used in wind tunnel tests. Therefore results of tests with round buildings or rounded corners might not be representative. Another aspect is 'blocking' in the wind tunnel. The projected surfaces of the buildings are limited in relation to the dimensions of the wind tunnel. When the model becomes too large, the velocity of the air will increase too much due to the smaller distance between the model and the ceiling of the wind tunnel. Therefore the profile of the tested object might be 5% of the tunnel profile at maximum. For the visualization of the flow pattern in wind tunnel tests, usage is made of smoke or fibers. With thermistors the average wind velocity can be determined in a certain measure point. The wind pressure on buildings and structures can finally be obtained by static pressure gauges on the model.

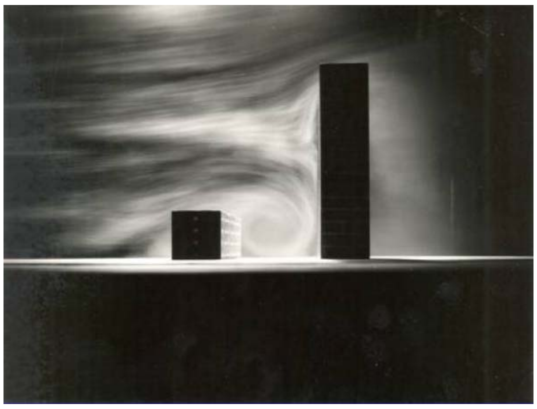


Figure 3.10: Visualization of the flow pattern

3.4 Computational Wind Engineering

3.4.1 Introduction

Computational Fluid Dynamics is in general a technique in which equations describing the fluid flow are solved numerically on a computer. Computational wind engineering (CWE) is the application of CFD in wind engineering. Topics of CWE are flows around off-shore structures, buildings, bridges, towers and effluents dispersion into the environment. Wind effects were traditionally investigated in wind tunnels. However, with the increasing computer capacity nowadays, the use of computational fluid mechanics to investigate the wind effects is growing.

Computational wind engineering is a difficult process because the flow obstacles, the so-called bluff bodies, often have sharp edges at their corners, where the flow separates. The flow field around bluff bodies is highly complicated compared to the flow fields traditionally treated in the field of CFD, such as channel or pipe flows. A very fine grid is required to analyze these flow fields with high precision.

CWE can be used to investigate the interaction between the wind and a structure. Global forces and moments can be determined, as well as local forces considering the fastening of façade elements. The dynamic behaviour of a structure can also be investigated. Besides, predictions of pollution dispersion around buildings or in city blocks are also important subjects in CWE. Finally, pedestrian comfort around high-rise buildings can also be investigated. Strong wind is often observed near the ground around these buildings. Since pedestrians are hindered considerably by this wind, architectural design to avoid such strong winds is also an important subject of CWE.

The use of CFD for wind engineering problems has some advantages over the use of wind tunnel tests. It is possible to calculate various model configurations in a relatively short amount of time and at low cost. The costs of a wind tunnel test are much higher, what discourages engineers to perform such a test in the early design phase. Wind tunnel tests are only performed at the final stage of the design process, to verify the calculations. With CFD simulations the wind effects on a building or structure can be investigated in an early stage of the design process. The results of these simulations can be used to optimize the design. Another advantage is that CFD programs determine all aspects of the fluid field. In every arbitrary point the velocity, fluid direction, turbulence and pressure can be calculated and visualized in various ways. An important disadvantage of CFD is that the user needs quite some much knowledge and insight of fluid dynamics, which the average structural engineer does not have. The user has to define a lot of parameters and make a lot of choices to setup a calculation. To verify the calculations it remains necessary to do some wind tunnel tests.

The process of a CWE analysis is more or less the same as a CFD analysis. The process is only more aimed at buildings and structures. The computational domain is the volume in which the flow has to be computed. This domain has to be discretized by the computational mesh which defines the spatial resolution of the numerical solution. The solution of the discretized equations is gained by iteration. After analyzing the solution some steps might be repeated if necessary to adapt the solution. Some typical aspects of the CWE process are discussed further in the next paragraphs.

3.4.2 *Computational domain*

Domain size
To examine the wind effects on a building dependently of its surroundings, a CAD model of the built environment has to be generated. The size of the computational domain depends on the built area to be investigated and the boundary conditions. The Cost Action C14 workgroup [8] gives recommendations for the size of the computational domain and on the use of CFD in wind engineering in general. Figure 3.11 gives the recommended size of the entire computational domain, depending on the maximum height H_{max} of a building inside the built area.

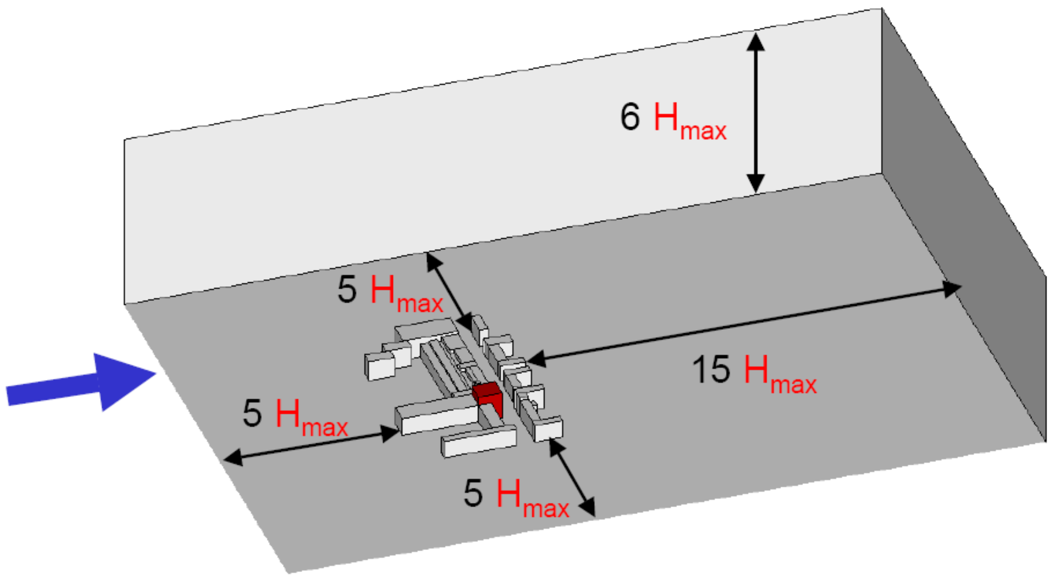


Figure 3.11: Recommended dimensions for a computational domain by the CAC14 workgroup (Franke, [8])

The inlet and side boundaries of the domain should be 5 H_{max} away from the area to be investigated. The outlet boundary should be 15 H_{max} away to allow flow development, as a fully developed flow is normally used as a boundary condition at the outlet. The top of the computational domain should be at least 6 H_{max} away from the tallest building to prevent an artificial acceleration of the flow over this building. The extent of the built area that is represented in the computational domain may be similar as the built environment in wind tunnel simulations (Franke, [8]). Most often an area with a radius of 300 m around the building or region of interest is modeled there.

Details
The central building at which the wind effects are of main interest requires the highest level of detail. The level of detail depends on the effects being studied. If for example pressures on the surface of a roof are of interest, details on the roof need to be presented. The level of detail may be limited by the computational mesh required to resolve the details. In general it can be said that features larger than 1 m should be presented (Franke, [9]). For individual buildings the level of detail required depends on the distance from the building of interest. Buildings further away may be presented as simple blocks.

Boundary conditions
When investigating the flow over for example an airplane, car or building, the flow equations are the same. It is the boundary conditions that are defined in the model that determines the behaviour of the flow. When the further located surroundings of the region of interest are cut off by the computational domain, the boundary conditions represent the influence of these cut off surroundings.

3.4.3 Computational grid

The results of the calculations highly depend on the used grid to discretize the computational domain. The resolution of the grid should be fine enough to capture all important physical phenomena. In regions where high gradients occur in the flow, the grid should be more refined. With regard to the shape of the computational cells, hexahedrals are preferred over tetrahedrals as they lead to smaller errors and better convergence.

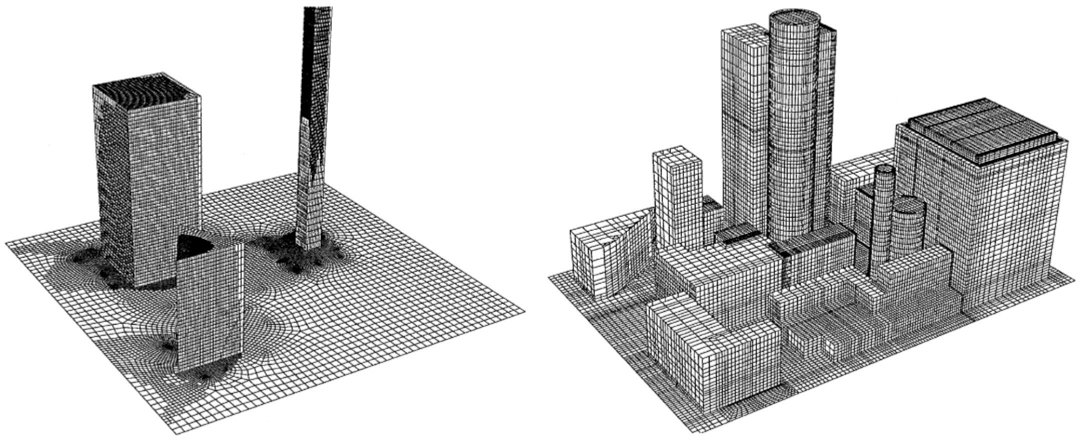


Figure 3.12: Unstructured hexahedral meshes for typical building groups (Kim, [14])

3.4.4 Solving the system

The system of algebraic equations is finally solved on a computer using iterative methods to reach a steady state solution. The solving process starts with an initial guess of the flow variables after which the variables are recalculated in every iteration. The user has to specify a permitted error and the iteration process continues until each equation is solved up to this error. Every solution depends on the grid that is used. The quality of the grid must be high, especially in the region of interest.

4. Geographic Information Systems

4.1 Introduction

Geographic information is information about specific places and objects on the surface of the earth (Bernhardsen, [1]). This information can be obtained through measurements and observations or interpreted from data analysis. Geographic information is usually represented in the form of maps, photos and images. Such a map or image is an effective medium for presentations and storage of geographical data.

A Geographical Information System (GIS) is a system which can process geographical information in various ways. It is a computer-based capability for saving, editing and analyzing geographical data. Compared to maps, GIS has the advantage that data storage and data presentation is separate. As a result, data may be presented and viewed in various ways. From a single data source many useful presentations can be created, for example of buildings, roads, the environment, utilities or property records. Many GIS systems can integrate data from a wide range of resources, including maps, images, statistical data and data from computer-aided design (CAD) systems.

A primary purpose of GIS is to analyze the relationships among objects in space. That is why all data in a GIS are linked to a specific location on the earth's surface through a system of coordinates. By using a GIS a wide variety of qualities and characteristics can be attached to the geographical locations. These characteristics are called attributes and they are the power of the GIS technology.

GIS is used in many technologies nowadays, including:
- Electronic navigation systems
- Analysis and planning systems for the design of road and transport systems
- Information systems for land ownership and building authority
- Urban planning systems

In this chapter some methods to convert the real world into a GIS model and some GIS datasets that are used in this thesis are discussed in detail. In Appendix A a historical overview and some recent developments in GIS technology are included as additional information.

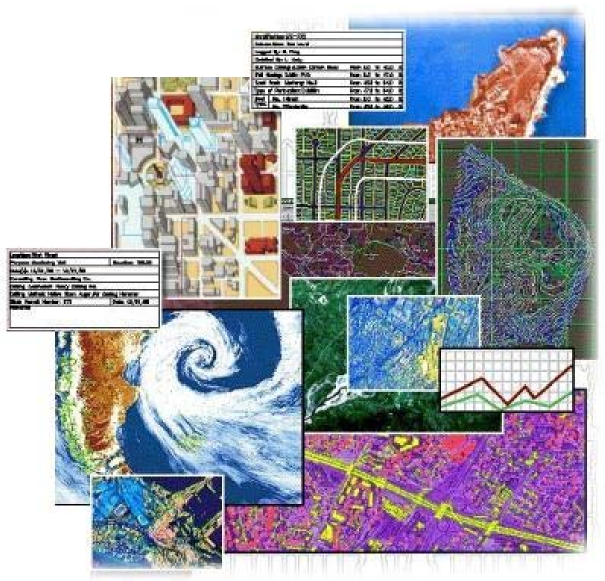


Figure 4.1: GIS application areas (Internet, [1])

4.2 Creating a GIS model of the real world

4.2.1 Introduction

The real world is far too complex to model it completely, so only specific areas and characteristics of interest should be selected for a GIS application. Simplified models of the real world have to be used to bring it into a GIS.

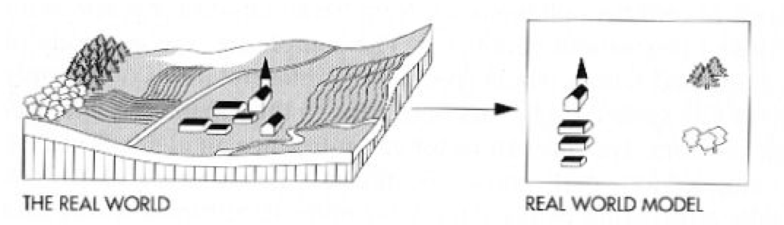


Figure 4.2: Transformation of the real world into GIS (Bernhardsen, [1])

Maps are graphic representations of the real world. Such a representation is always an abstraction of reality, as it is impossible to capture all complexity. Topographic maps for example abstract the real three-dimensional world at a reduced scale on a two-dimensional paper. Standard topographic maps can show a variety of information, for example roads, land use, elevation, rivers and boundaries. The objects represented on maps are called features. These features have a location, a representative shape and a symbol that represents one or more of its characteristics. With attributes, non-spatial information which describes the characteristics of a feature can be added to the features. In a GIS, attributes are stored in tables that can be viewed by selecting the object. There are standard attributes that indicate the position of the object and connections to other objects. There may also be specific attributes such as land use and population. Attributes can be used to select certain features or to perform mathematical calculations.

There is a very wide range of techniques available to obtain the data for a GIS model. They vary from digitizers which extract spatial information from photographs and maps, to land survey instruments and very sophisticated global positioning systems (GPS) that use satellites. However, data collection is by far the most time consuming part of creating a GIS model and forms the bottleneck of the process. In the next paragraph some techniques to acquire data for GIS applications are discussed more in detail.

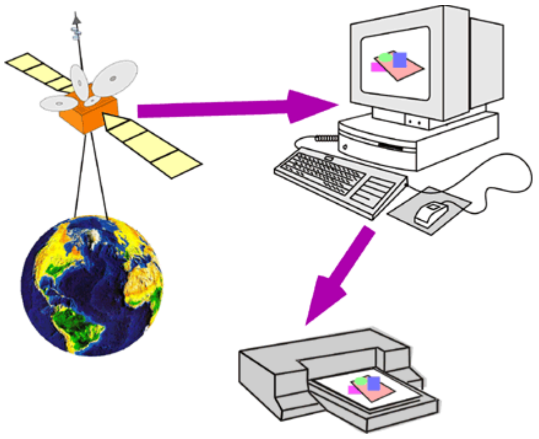


Figure 4.3: Real world data acquisition using satellites (Internet, [4])

4.2.2 Spatial data acquisition

There are several methods available for entering spatial data into a GIS. Examples are:

- Global Positioning Systems (GPS)
- Remote sensing
- 3D laser scanning

Global Positioning Systems (GPS)
A GPS is a set of hardware and software to determine accurate locations on the earth using signals received from satellites. With GPS, location data and associated attribute data can be transferred to a GIS. GPS operate by measuring the distances from multiple satellites orbiting the earth to compute the x, y and z coordinates of the location of a GPS receiver. For an accurate determination of the receiver location, four satellites are required. The use of more satellites simply adds confirmation to the location of the receiver. There are 24 satellites in the total GPS system from which 21 are active. There are 3 reserve satellites in case of unexpected circumstances. At least five satellites are visible from any location on earth. As buildings, hills, trees and other objects may block the signal of a satellite, there is usually still a sufficient amount of satellites available to determine the location of the receiver. For each satellite the location in space is accurately known. The orbits are carefully planned and constantly updated by the satellites. For each satellite the receiver can see, the distance to the receiver is calculated by comparing the time the signal was sent with the time it was picked up by the receiver. When for four satellites at minimal these distances are known, the exact location of the receiver can be determined. Examples of GPS equipment that is currently available are car navigation systems, small hand-held units and mobile telephones.

Remote sensing
In the broadest sense, remote sensing is the measurement of information of an object by a recording device that is not in physical contact with the object (Internet, [20]). In the context of GIS technology, remote sensing is the utilization of for example aircrafts and satellites to gather information about the environment. Without direct contact between the recording device and the object, some means of transferring information through space must be utilised. Remote sensing instruments record information about an object by measuring the object's transmission of electromagnetic energy from reflecting and radiating surfaces. Remote sensing is mainly used for earth observations. Using satellites, planes or balloons, information about the surface of the earth is acquired.

3D laser scanning
A 3D laser scanner is a device that analyzes a real-world object or environment to collect data of the shape of the object. The scanning device scans the surface of an object with a focused beam and records the reflected light. The collected data can then be used to construct digital, three dimensional models for a wide variety of applications. The purpose of the 3D laser scanner is to create a point cloud on the surface of the object. The points can then be used to reconstruct the shape of the object into a 3D surface model. For most situations, a single scan will not produce a complete model of the object. Multiple scans from different directions are required to obtain information about all sides of the object.

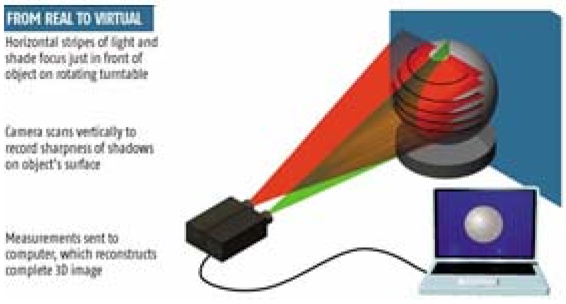

FROM REAL TO VIRTUAL
Horizontal stripes of light and shade focus just in front of object on rotating turntable
Camera scans vertically to record sharpness of shadows on object's surface
Measurements sent to computer, which reconstructs complete 3D image

Figure 4.4: Principle of 3D laser scanning (Internet, [4])

4.3 GIS data structures

In a GIS model the geographical data can be divided into spatial data and attribute data. The spatial data contain the objects representing the real world, such as buildings, roads and rivers, and gives the location of these objects. The attribute data contain the characteristics of these objects and indicate what the object is. The spatial data in GIS models can be presented in two ways: as raster data in the form of uniform cells, or as vector data in the form of points, lines and areas. These two data types will be discussed in the following paragraphs.

4.3.1 Raster data models

The raster model represents the real world through uniform, regular cells. Raster data can be visualized as a regular grid lying over the terrain, where the area is divided into rows and columns. The spatial data is therefore not continuous but divided into discrete units. The area which each cell represents varies from a few meters to kilometers and is known as the resolution of the grid. Each grid cell contains location coordinates and an attribute value, such as land use or terrain type. For the description of the terrain type several codes are used. For example a forest has code 1, a road has code 2 and a house has code 3. Within each cell the terrain is generalized to be a flat surface of constant elevation.

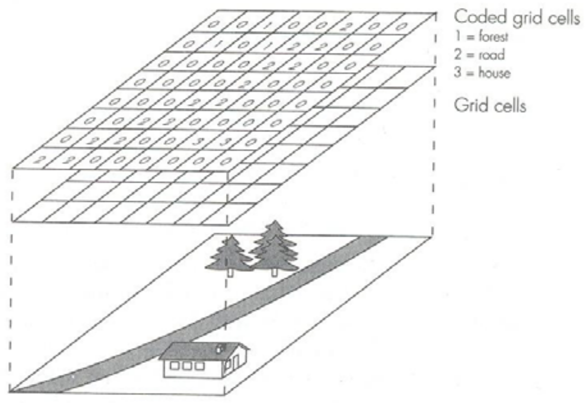


Figure 4.5: Each grid cell has a code describing the terrain within that cell (Bernhardsen, [1])

The attribute values in a raster model can represent numerous phenomena, for example codes for topography, urban districts, land use or distances from a given object. Only one attribute value may be assigned to a single cell, so objects that have several attributes are therefore represented with a number of raster layers, one for each attribute. Raster databases may therefore contain hundreds of thematic layers.

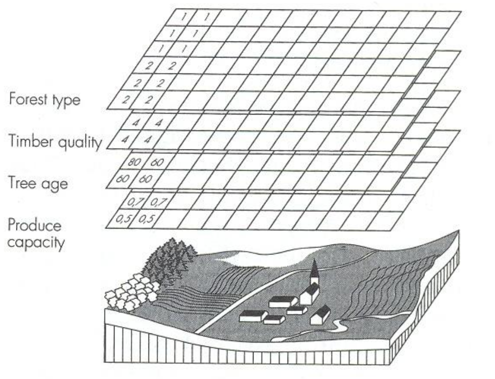


Figure 4.6: Objects with several attributes are represented with a number of layers, one for each attribute (Bernhardsen, [1])

Raster data models are usually used in digital imaging, where just the color is recorded for each cell. There is no distinction between dots that are part of a note or dots that are part of a line segment for a geometric entity, such as a line or a circle. There are no X and Y values in the raster data that define where a line starts or where it ends. Files such as TIFF, GIF, JPEG and BMP are typically raster data formats.

4.3.2 Vector data models

In the vector data model the elements that represent the real world objects are described as points, lines or polygons. There is a mathematical definition for the parts of the drawing that represent the object. There are real X and Y values for the geometric entities. An object shape is represented by dots which are located where the shape of the object changes. The dots are joined by straight lines. Examples of vector elements are property boundaries for houses represented as polygons and locations represented as points. Attributes are assigned directly to the objects. For 3D vector data models, the Boundary representation is the most used representation. This method is discussed further in Paragraph 5.2.3.

4.3.3 Raster VS vector data models

Raster data models are less accurate than vector data models in defining the spatial location, as the smallest entity is always a cell, which represents an area and not a coordinate. But it is more expensive in terms of data storage and computing power (Delaney, [7]).

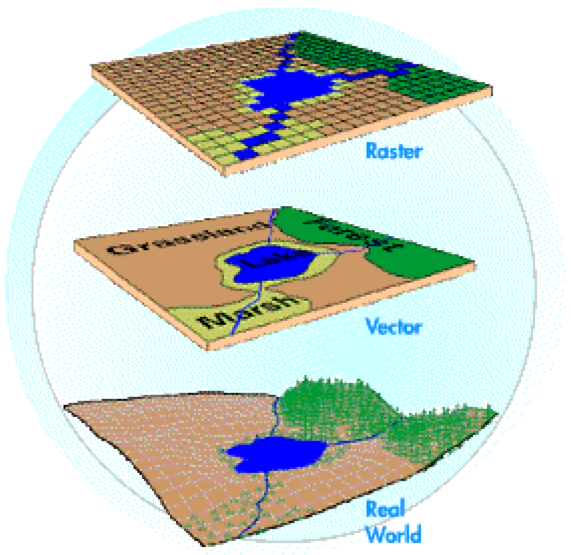


Figure 4.7: From the Real World to Vector and Raster models (Internet, [5])

4.4 GIS applications

GIS is used in a wide variety of applications nowadays, varying from navigation systems to planning systems for the design of roads and urban districts. Many other disciplines can also benefit from GIS techniques. Two important GIS datasets that will be used during this thesis are the Top10vector and the Actueel Hoogtebestand Nederland (AHN). The Top10vector dataset is a digital 2D map of the Netherlands. The AHN is an elevation model of the Netherlands where the height of the surface and all kinds of objects like buildings and roads is stored. When these two datasets are linked, a 3D model of a certain region of the Netherlands can be generated. These models can be used in the Virtual Wind Tunnel to simulate the flow pattern around a building that is placed in a built environment. The Top10vector and AHN datasets will be discussed further in the following paragraphs.

4.4.1 Top10vector

As one of the first countries the Netherlands have a very detailed digital topographic map of the whole country in 2D. This Top10Vector dataset is based on air photography and terrain exploration and has a scale of 1:10.000 (Internet, [11]). The Top10Vector contains closed polygons that represent different elements on the earth's surface, like buildings and roads. Out of the built-up area the individual buildings are visible. Within the built-up area only building blocks are visible. The Top10Vector contains the following topographic elements:

- Buildings
- Roads
- Railways
- Vegetation
- Hydrographs
- Relief
- Boundaries between regions

For the Netherlands the dataset is divided in 675 files that correspond to the size of real topographic maps. The files are available in different formats including DXF, which makes it very suitable for use in CAD applications.

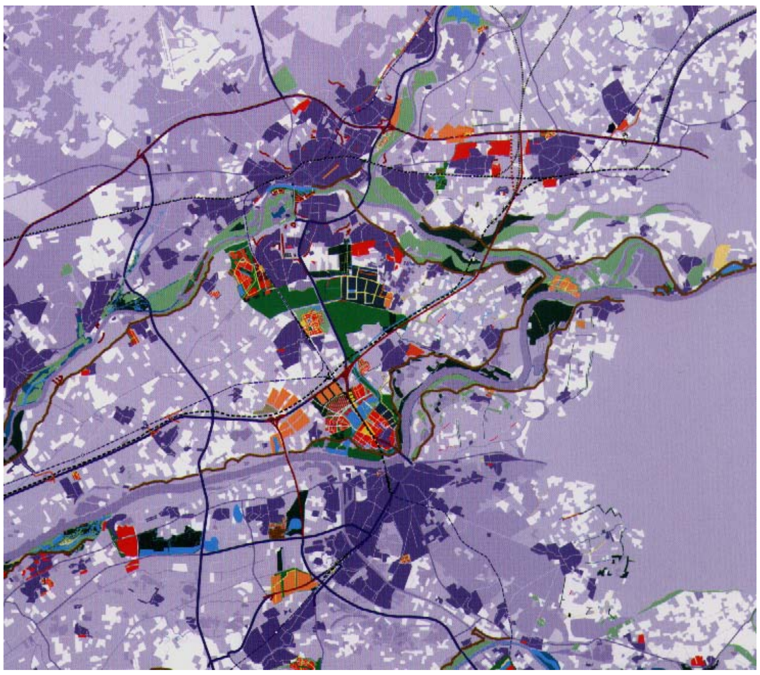


Figure 4.8: Top10Vector file for the area between Arnhem and Nijmegen (Internet, [12])

4.4.2 Actueel Hoogtebestand Nederland (AHN)

The Survey Department of the Dutch Ministry of Transport, Public Works and Water Management (Rijkswaterstaat) has built a detailed elevation model of the Netherlands. The project was initially meant to provide information for the management of coasts, dykes, polders and higher-level areas which seem to be drying out. But also for the construction of infrastructural works the data are of great importance (Internet, [13]). The heights of the dataset are stored in points that lie on a rectangular grid. The AHN has an average point density of 1 point per 16 m^2 and contains next to information about the ground level, also elevation data of built areas, dykes and roads.

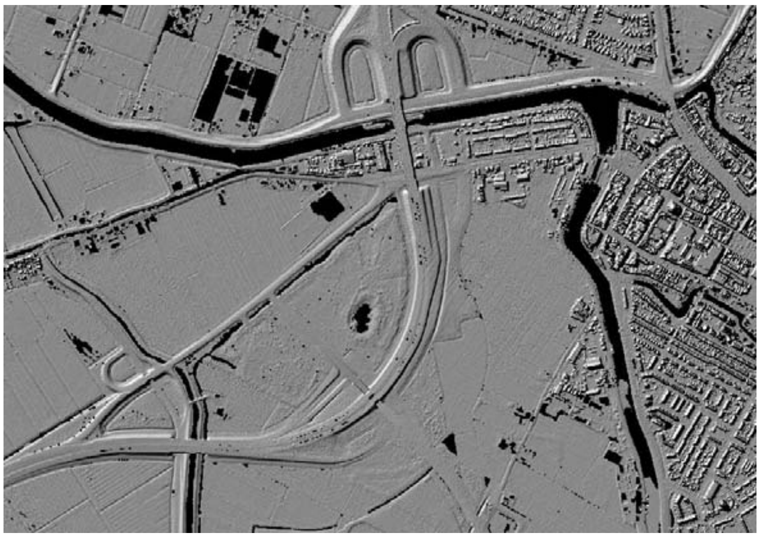


Figure 4.9: Graphical representation of the AHN dataset (Internet, [13])

The technique used to obtain data for the AHN is called *airborne laser altimetry*. An airplane using a laser scanner flies across the area that must be scanned. The plane transmits pulses of laser light to the ground and measures the time of pulse return. The time span between transmission and return indicates the relative elevation of the measured point. To determine the absolute elevation of the plane relative to NAP, the position of the plane must be known. Via GPS the location of the plane is determined. The laser data are checked with reference measures on a flat surface of approximately one hectare, frequently a football field. From these measurements if follows that the height of the measured points differ 5 cm at average from the real heights with a standard deviation of 15 cm (Internet, [13]). The laser scanner does not measure only terrain surfaces but also objects like buildings, cars and vegetation. To produce digital maps of a surface from the scanned data it is necessary to classify the measured points and distinguish the points on the ground from the other points. This process is called filtering and the result is shown in Figure 4.9. Usually it is not possible to recognize the difference between small objects and little fluctuations in height of the surface automatically. These small objects have to be filtered manually then.

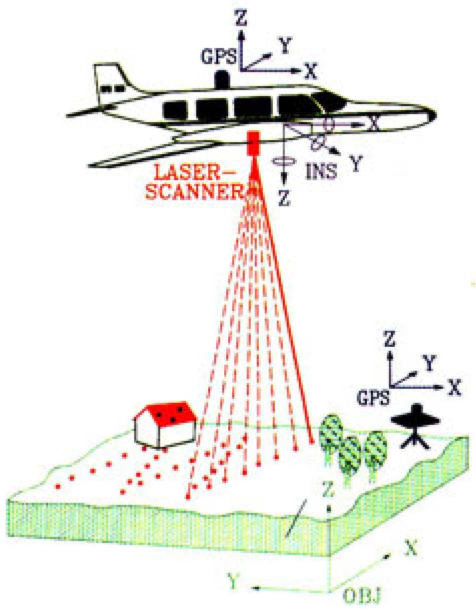


Figure 4.10: Principle of Airborne Laser Altimetry (Internet, [13])

5. Computer-Aided Design

5.1 Introduction

Computer Aided Design (CAD) is the use of a wide range of computer-based tools that assist engineers, architects and designers in their design activities. It is used in a wide range of applications, from producing designs for buildings, roads and industrial prototypes to producing plans for electrical supplies. CAD drawings can generally be categorized based on the engineering discipline they are related to. There are specialized CAD software packages that are aimed at specific disciplines, but many CAD packages can be used for a wide variety of disciplines.

Before the computer era the use of paper drawings was the only means of documenting a design. These drawings took various forms ranging from small handmade sketches to large sheets of paper many meters long using drafting machines (Schoonmaker, [21]). The overall format was standardized across entire nations. First commercial applications of CAD were in automotive and aerospace industry. In those days CAD was limited to producing drawings similar to hand-drafted drawings. With the rise of computer technology, CAD systems to generate, edit and store drawings electronically have developed over a number of decades and they became the standard method for producing drawings. Adapting the design is an important advantage of CAD systems compared to handmade drawings. Designs can be scaled, moved or rotated without the need of generating a new drawing. Today CAD is not limited to drafting and rendering; it ventures into more intellectual areas of designer's expertise.

In this chapter various methods to describe geometry in general are dealt with. Several CAD systems are discussed that are especially developed for the design of buildings. The design tools that will be developed in this thesis must be able to work with the various types of CAD models that can be generated with these systems.

5.2 Form representation

Geometric modeling deals with the mathematical representation of curves, surfaces and solids that is necessary in the definition of complex physical or engineering objects. The objects of concern in engineering range from simple mechanical parts to complex objects such as automobiles, ships, airplanes and buildings. These geometric shapes all have topology and geometry. Topology describes the relationship between faces, edges and vertices of an object that does not change when the object is deformed. Geometry describes the exact position of these faces, edges and vertices and gives shape to the topology.

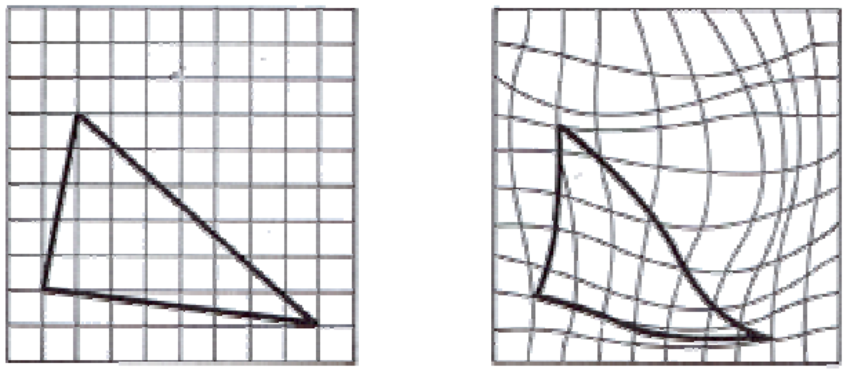


Figure 5.1: Topology contains the properties of geometric shapes that do not change when the shape is deformed. It gives coherence between the points (Tolman, [25])

There are several methods to represent the geometry and topology of an object. The common method is the use of traditional, technical 2D drawings with cross-sections and views from different sides. Although these drawings are sufficient, the geometry and topology can be represented in more advanced ways using 3D modeling:

1) Wire Frame modeling
2) Surface modeling
3) Solid modeling:
   a. Constructive Solid Geometry
   b. Sweeping
   c. Boundary representation

These methods will be discussed further in the following paragraphs.

5.2.1 Wire Frame modeling

Wire frame modeling is one of the earliest geometric modeling techniques and is the simplest way of representing 3D objects. It is constructed using points with 3D coordinates, connected with straight lines or curves. A wire frame model does not have surfaces; it contains only points and lines defining the edges of the modeled object.

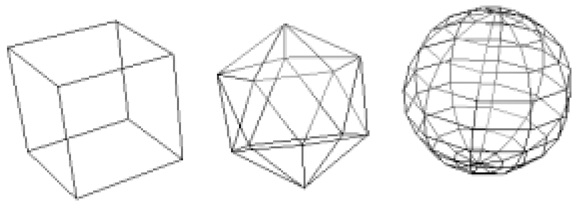


Figure 5.2: Wire frame representation (Internet, [10])

5.2.2 Surface modeling

The first extension of the wire frame model is the surface model. It contains surfaces in stead of points and lines and is used for modeling the shape of curved surfaces, such as shell structures and terrain models. Complex, double curved surfaces can be represented using B-Splines or NURBS (Tolman et al, [25]). These techniques will be discussed in Paragraph 5.3.

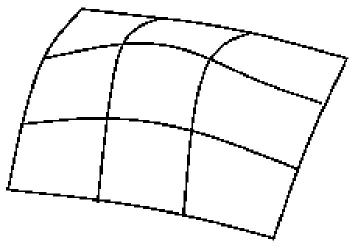


Figure 5.3: Surface representation (Internet, [10])

Similar to surface modeling, an object can be described with meshes. A mesh is a collection of vertices and polygons forming faces, which define the shape of an object. The faces are arranged in such a way that they form the outside surface of the object. Meshes usually consist of triangles or other simple convex polygons.

5.2.3 Solid modeling

In a solid model the object is defined by a closed boundary. The boundary of the object separates the interior and exterior of the model. The object is defined by the volume space within the defined boundary. An advantage of solid modeling is that information like weight, volume or moment of inertia can be calculated for an object. A number of techniques are developed to form a solid model, like Constructive Solid Geometry, Sweeping and Boundary Representation.

Constructive Solid Geometry
The Constructive Solid Geometry (CSG) combines simple geometric primitives to form more complex solid objects using so-called Boolean operations. Typical geometric primitives are cones, cylinders, spheres and blocks. Using a Boolean operation, a new solid is constructed from two intersecting solids. Typical Boolean operations are union, intersection and difference. Given two sets A and B, its union consists of all points from either A and B. Its intersection consists of all points in both sets and its difference, for example written as A-B, consists of all points in A but not in B.

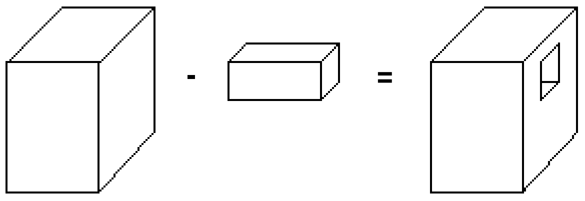


Figure 5.4: Constructive solid representation as a result of subtraction (Internet, [10])

Sweeping

The geometric shapes that can be built using the Boolean operations on the geometric primitives are limited. For complex objects it is more efficient to generate the shape using Sweeping. Sweeping can be carried out in different forms. The most commonly used are extrusion and revolving. With extrusion an object model can be generated from a 2D cross-section, given the direction of extrusion and a certain depth. By revolving a 2D cross-section that is specified by a closed curve around the axis of symmetry, an axially symmetric object can be formed. The sweeping technique is most convenient for solids with translational or rotational symmetry.

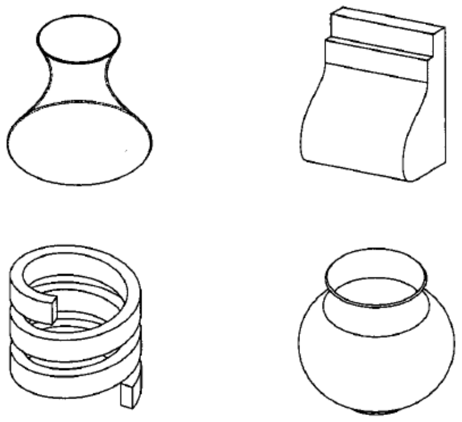


Figure 5.5: Some shapes are easier to create with sweeping techniques (Internet, 10)

Boundary representation

The most used representation in 3D modeling is the boundary representation. With this method, an object is represented as a collection of boundary surfaces. The boundary representation is an extension to the wire frame model by adding surface information to the model. Due to this surface information the method is very suitable for hidden surface removal and rendering.

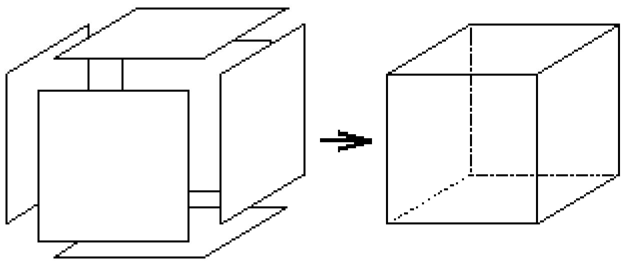


Figure 5.6: Boundary representation by joining the boundary surfaces (Internet, [10])

5.3 Freeform curves and surfaces

With Non Uniform Rational B-Splines (NURBS) a wide range of curves and surfaces can be parametrically represented, from straight lines and flat panels to precise circles and complex curved surfaces (Rogers, [19]). NURBS curves and surfaces are defined by a complex combination of mathematical equations, formulas and procedures. The development of NURBS began in the 1950's when engineers needed a mathematical representation of freeform surfaces, like car bodies and aircraft wings. Pierre Bézier was one of the pioneers of the development of NURBS and invented the Bézier curves, the precursors of NURBS. They are used for curve representation, but are not suited for describing any curve. Bézier curves were followed by B-spline curves and finally by NURBS, which can be used to describe any curve. B-spline curves and NURBS are generalizations of the Bézier curves and have become the standard for describing, constructing and modeling free formed curves and surfaces in computer aided design and interactive graphics. In this thesis NURBS curves will be used to simplify a model of the building of interest. This procedure is discussed in Paragraph 7.3.7.

5.3.1 Bézier Curves

Bézier curves are the precursors of NURBS. They are used for curve representation, but are not suited for describing any curve. A Bézier curve is determined by a control polygon, such as shown in Figure 5.7. The control polygon is defined by four points, from which two are the endpoints of the Bézier curve. The point B_0 is the origin endpoint. The point B_3 is the destination endpoint. The points B_1 and B_2 are so called control points and control the path of the Bézier curve. The curve does not pass through any of the control points; it only passes through the endpoints with tangent vectors of the same direction as the first and last polygon spans. Bezier surfaces are surfaces which are described by a set of Bézier curves.

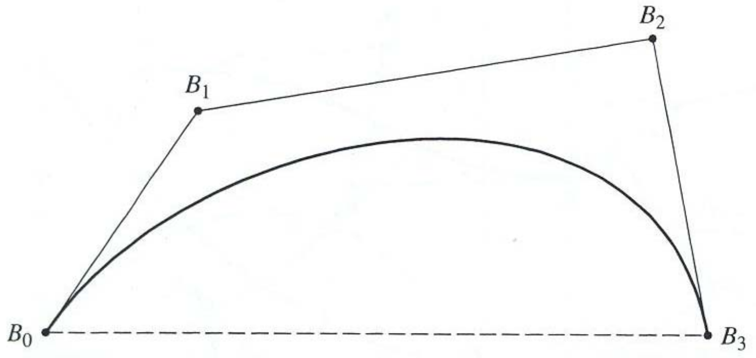


Figure 5.7: A Bézier curve is defined by a control polygon of four points (Rogers, [19])

The parametric Bézier curve function is defined by:

$$P(t)=\sum_{i=0}^{n} B_i J_{n,i}(t) \quad \text{for } 0 \le t \le 1 \ ; \ 2 \le i \le n+1 \tag{5.1}$$

Herein is n the degree of the curve and i the number of control points. The Bézier curves have at least one more control point than the degree of the curve. Point B=0 refers to the first point and B=n refers to the last point of the Bézier curve. The term $J_{n,i}(t)$ in this function is called the blending function:

$$J_{n,i}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i} \tag{5.2}$$

The blending function, also known as the basis function, influences the control points that form the Bézier curve.

In summary, the parametric Bézier curve function is built up of three different components:

- **Control points:**
  The control points of a Bézier curve define the vertices of the control polygon and are given by B_i. The curve follows the shape of the control polygon. By moving the control points of the polygon, the shape of the Bézier curve can easily be changed. Usually, the first and the last point of the control polygon are the same as the first and the last point of the Bézier curve.

- **Basis Function:**
  The basis or blending function $J_{n,i}(t)$ indicate the influence each control point has on the shape of the curve on that point. Figure 5.8 gives some cubic Bézier curves with their blending function. With Bézier curves the blending functions are symmetrical.

- **Degree:**
  Each Bézier curve has an associated degree that controls the smoothness of the curve. A degree of 1 will produce a straight line between the control points. A higher degree will result in a smoother curve. The higher the degree of the curve, the more points are needed to define it.

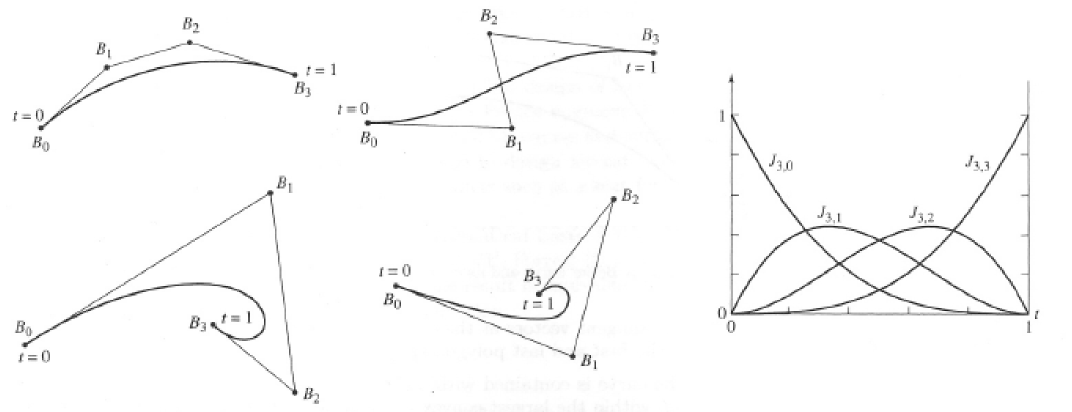


Figure 5.8: Examples of cubic Bézier curves and their basis function (Rogers, [19])

5.3.2 B-Spline Curves

B-Spline curves are a generalization of the Bézier curve. A spline curve is a sequence of Bézier curve segments that are connected together end to end to form a single continuous curve (Sederberg, [20]). The spline curve consists of several subintervals that correspond to the various Bézier curves of the spline. Each sub endpoint is called a *knot*. The set of sub endpoints is called a *knot vector*. Each subinterval of the spline is influenced by only *d* control points.

B-Spline curves can be classified by uniform or non-uniform spacing of the knots. With uniform spacing B-spline curves the distance between the knots is equal. Non-uniform spacing B-spline curves have various distances between the knots. The shape of these curves can vary more than the shape of uniform B-spline curves.

5.3.3 NURBS

A special form of B-spline curves are the Non-Uniform Rational B-Spline (NURBS) curves. The curves are non-uniform, so the distance between the knots can vary. Rational B-spline curves are specified as a ratio between two spline functions:

$$P(t) = \frac{\sum_{i=o}^{n} w_i \cdot B_i \cdot J_{n,i}(t)}{\sum_{i=o}^{n} w_i \cdot J_{n,i}(t)} \qquad (5.3)$$

Herein is B_i the set of control points and w_i the set of weight factors for each control point. The weighing factors determine the influence of the various control points. So, when the distance between the knots of a curve varies and the control points are weighted, the curve is a Non-Uniform Rational B-Spline curve. With NURBS, any shape from a simple line or circle to the most complex 3D surfaces or solids can accurately be described. By moving the control points of boundary curves or internal curves, almost any surface can be described. In CAD and CFD-programs, three-dimensional objects are generally described using NURBS.

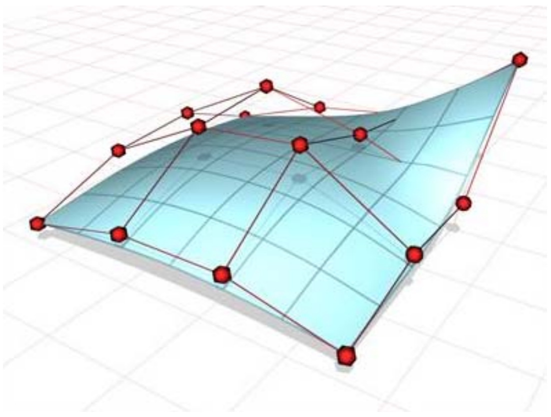


Figure 5.9: NURBS surface (Internet, [6])

5.4 Exchange of CAD Data Files

Most CAD file types are binary data files. This means that one can not directly view and manipulate the data in the file. Files which one can directly view are known as ASCII files. Most CAD systems have their own file format, custom-made for the particular program. A major disadvantage is that the files have to be read, written and viewed with the same CAD-system that created it. Therefore, CAD data files are not directly interchangeable from one CAD system to another. Neutral file formats have been developed to deal with this file incompatibility. The format is public domain to some degree and is almost always in ASCII format. The originating CAD system creates the neutral file after which it can be exchanged between two systems. The receiving CAD system reads or imports the neutral file. The most common neutral file formats are DXF, IGES and STEP (Schoonmaker, [21]).

The exchange between a CAD system and Gambit, the Fluent pre-processor, is also based upon neutral file formats. Gambit is based on the ACIS geometrical libraries, but it can also import IGES and STEP data files. However, there are some bottlenecks when CAD models are translated to CFD systems. The ACIS modeling algorithms require a high degree of precision and accuracy of the model, but the inaccurate way in which most models are drawn leads to errors and inconsistencies. Therefore, the geometry often needs to be repaired after it is imported in Gambit.

5.5 Developments in CAD

In the previous paragraphs the methodology how CAD is used in the traditional CAD systems is discussed. However, there are some important recent developments in CAD that deserve some attention. In this paragraph object-oriented and parametric modelers are discussed. In Appendix A.5 additional information about Industry Foundation Classes is included. The design tools that are developed in this thesis should support the different types of building models that originate from the various CAD systems and modelers.

5.5.1 Object-Oriented Modelers

A major disadvantage of the traditional CAD systems is that the models have no meaning for the computer. The drawings contain lines and points, but the interpretation is done by man. In the last decade a new technology is developed, called object-oriented modeling (Blaha, [5]). With this technology a model of the reality is described in terms of objects that are used in reality, such as walls, columns and girders. The model contains essential characteristics of these objects and also information about the relationships between the objects. If, for example, a wall is replaced in the model, the windows, doors and channels in that particular wall are also replaced automatically. This is very useful in the design process.

The structure of the object-oriented models is based on so-called *information models*. With these models information can be modeled that can be interpret by man as well as the computer. Object-oriented models are a collection of objects that are organized into classes with common features like attributes and procedures. Attribute values, such as width, height, length or material, represent properties of the object itself or associations with other objects. Procedures are operations that the object can perform, such as move or stretch, and describe the behaviour of objects. The structure of object-oriented models can be represented using a graphical modeling language. In this case the Unified Modeling Language (UML) will be used (Internet, [17]).

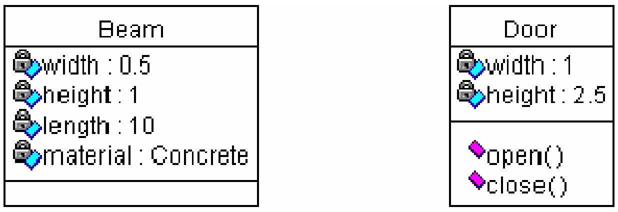


Figure 5.10: Objects in an object-oriented model

In Figure 5.10 two examples of objects in an object-oriented model are given. The objects have attribute values, like width and height, and the object *door* also has two procedures; the door can be opened or closed. Objects are derived from so-called classes. A class describes the type of an object. Figure 5.11 gives an example of the class *structural element*. In a class the properties of the objects of a certain type are described. The object *beam* in Figure 5.10 is an instance of the class *structural element*. Each object has its own value for each attribute, but shares the attribute names with other instances of the class. Other instances of the class *structural element* can be objects like columns or plates for example.

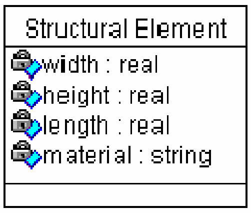


Figure 5.11: Class in an object-oriented model

One of the powerful tools of object-oriented models is the ability of hierarchical classification and inheritance. Attributes and operations can be shared among classes based on a hierarchical relationship. A so-called *superclass* has general characteristics that *subclasses* inherit. Each subclass inherits all features of its superclass and adds its own unique features. Figure 5.12 gives an example of inheritance. The subclasses *beam* and *column* inherit the attributes width, height, length, weight and material of the superclass *structural element*. The subclasses can add their own attributes, like bending moment or axial force.

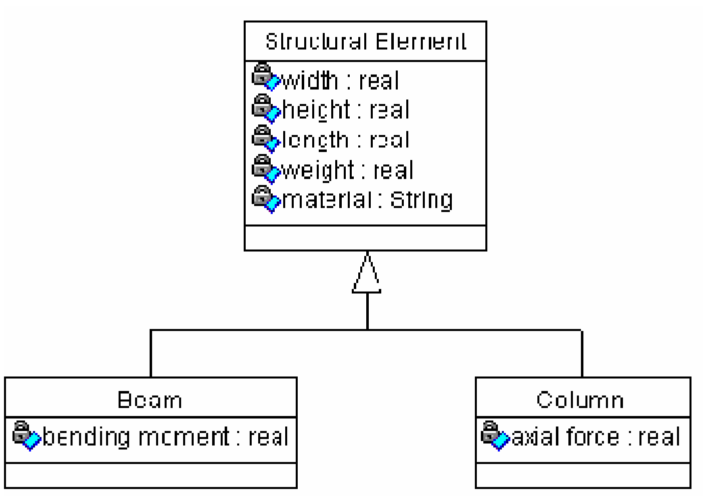


Figure 5.12: The subclasses Beam and Column inherit attributes from a superclass

The examples in the preceding pictures can be extended to form an entire room. Figure 5.13 gives a model of a bathroom that is enclosed by walls, a ceiling and a floor. The bathroom contains a toilet, shower, bath tub and a wash basin.

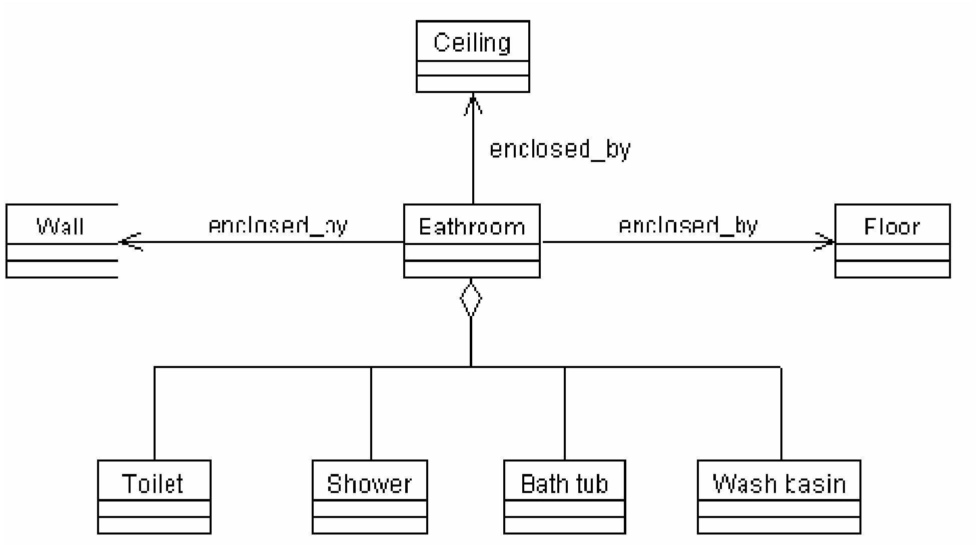


Figure 5.13: UML model of a bathroom (Kerklaan, [13])

Figure 5.14 shows a class model of an entire house. The house can have several rooms, like a hall, kitchen and living room, and each room has its own orientation. The rooms are separated by one or more dividing elements, like floors, walls or a roof. The dividing elements can have cavities with a door or window. The house can also have a garden and it is built on a foundation. With this class model several instances for different house designs can be made.

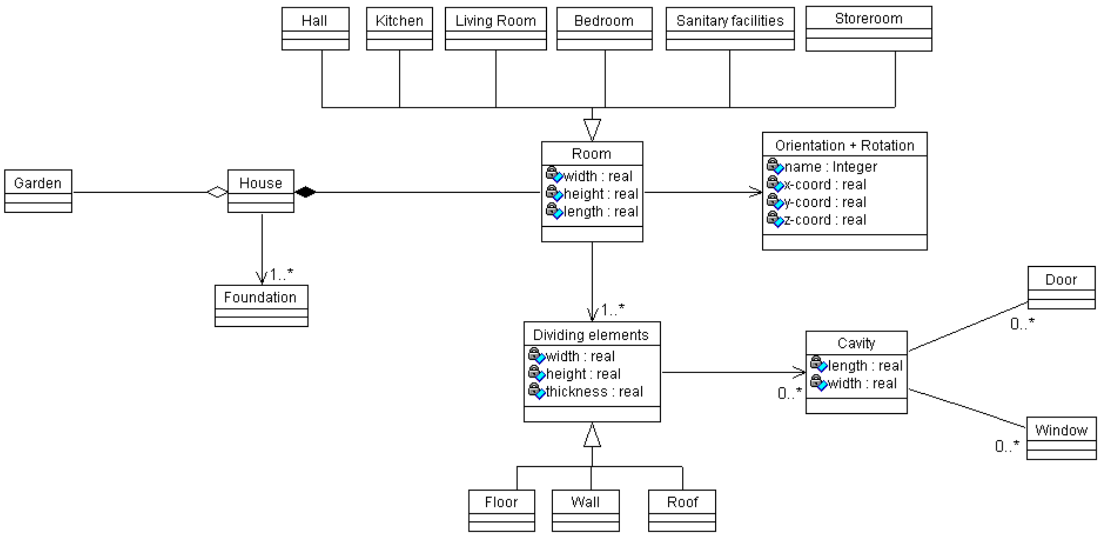


Figure 5.14: UML model of a house (Kerklaan, [13])

An important advantage of object-oriented modeling is the transparent structure and relative simplicity of the models. The above discussed method of object-oriented modeling, which is explained with the graphical modeling language UML, is often used in the next generation architectural and structural CAD systems. Figure 5.15 shows cross-sections of simple instances of the class model of the house, generated in ArchiCAD (Internet, [18]). All objects, structure or furniture, have sense for the computer; they are not only points and lines. The computer recognizes the various elements. By replacing an element, all accompanying elements are also replaced.

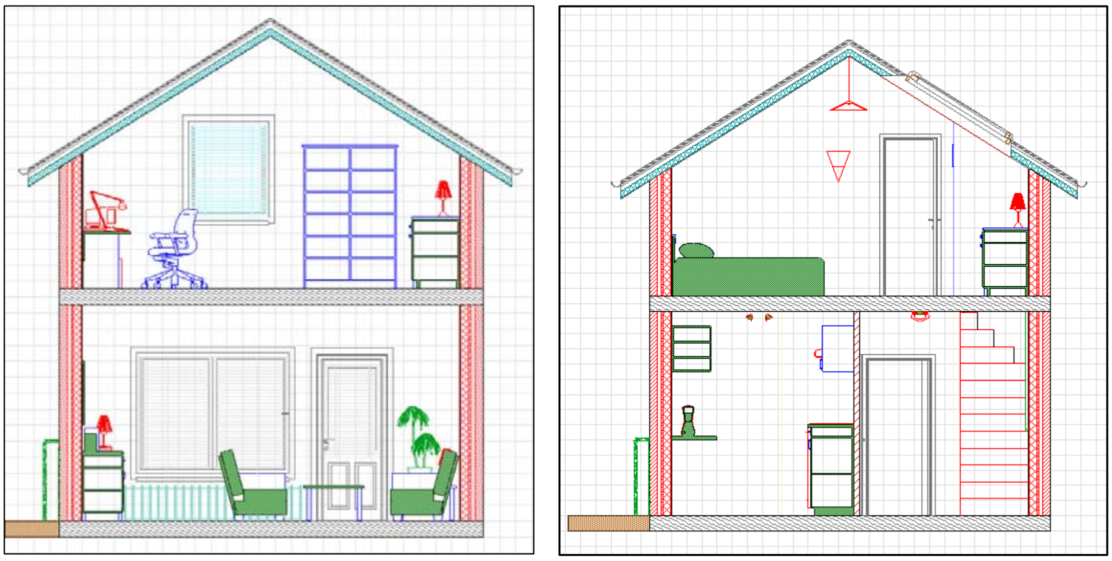


Figure 5.15: Instances of the class House (Kerklaan, [13])

5.5.2 Parametric modelers

Some other modern CAD systems have the ability to create parametric models. In the original CAD systems, coordinate-based geometry is used to create graphic entities. Editing these entities is difficult and can easily lead to errors. In a parametric model, each entity, such as a line, arc or Boolean primitive, has parameters associated with it. These parameters control the geometric properties of the entity, such as the length, width and height or radius. They also control the location of the entities within the model. By changing some parameters, the model can quickly be adapted. Within some parametric modelers also equations can be added to the models. These equations can be used to construct relationships between the parameters. This is very useful when a certain parameter depends on the values of others.

A parametric modeler is also aware of the characteristics of components and the association between them. It maintains constant relationships between elements as the model is manipulated. A door in a wall for example maintains its relationship with the wall when the wall is replaced. This is also the case in object-oriented modeling.

Examples of parametric objects in Building Engineering are prefab elements, façade elements or hollow core slabs. Although the composition of these elements can be quite complicated, there are only a couple of parameters required to describe the element. Buildings are often unique products that are constructed with standard, serial manufactured elements. Because parametric modeling is very suitable for designing with standard elements, it is very useful for Building Engineering.

5.6 Conclusion

In this chapter various types of CAD models are discussed. The traditional CAD models consist of points, lines, surfaces and solids and have no meaning for the computer. The next generation CAD systems work with advanced object-oriented models and parametric models and do have sense for the computer. The design tools that are developed in this thesis should support the different types of building models that originate from the various CAD systems. For use in the Virtual Wind Tunnel, it also has to be possible to exchange the different CAD models with the CFD software. Chapter 7 deals with this problem.

6. Thesis approach

The theory discussed in the previous chapters will be used to develop the various design tools to setup the geometry for CFD calculations. In this chapter an overview of the options to determine the wind loads will be given first, just as the current state of the Virtual Wind Tunnel. Then the purpose of the design tools and their contribution to the Virtual Wind Tunnel is given. Finally the strategy of the various tools to setup the geometry will be given.

6.1 Wind load determination

In Figure 6.1 an overview is given of the options to determine the wind loads on a building or structure. Calculations according to the building code and wind tunnel studies are generally applied methods and their results are widely accepted. As there is still contradiction on the use of CFD in wind engineering, CFD results are not generally accepted yet. The Virtual Wind Tunnel is therefore proposed to develop a general approach for setting up a flow problem. In recent graduation studies a computational domain is developed in which the CFD calculations can be performed. Some recommendations of these studies are used in this thesis to develop the various design tools. The scheme is discussed more extensively on the next page.

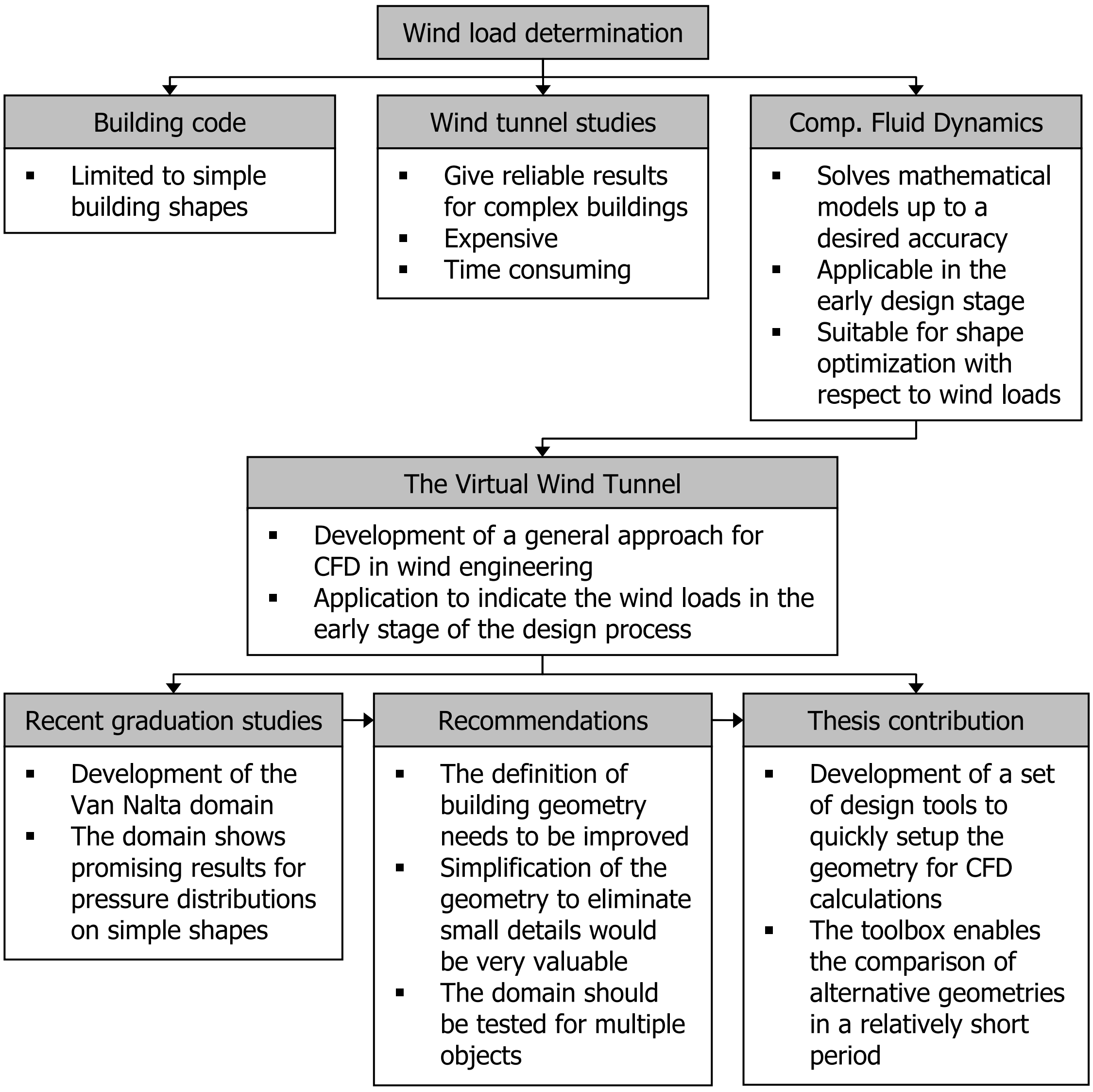

Wind load determination
Building code
▪ Limited to simple building shapes
Wind tunnel studies
▪ Give reliable results for complex buildings
▪ Expensive
▪ Time consuming
Comp. Fluid Dynamics
▪ Solves mathematical models up to a desired accuracy
▪ Applicable in the early design stage
▪ Suitable for shape optimization with respect to wind loads
The Virtual Wind Tunnel
▪ Development of a general approach for CFD in wind engineering
▪ Application to indicate the wind loads in the early stage of the design process
Recent graduation studies
▪ Development of the Van Nalta domain
▪ The domain shows promising results for pressure distributions on simple shapes
Recommendations
▪ The definition of building geometry needs to be improved
▪ Simplification of the geometry to eliminate small details would be very valuable
▪ The domain should be tested for multiple objects
Thesis contribution
▪ Development of a set of design tools to quickly setup the geometry for CFD calculations
▪ The toolbox enables the comparison of alternative geometries in a relatively short period

Figure 6.1: Options to determine the wind loads and the purpose of the Virtual Wind Tunnel

Until now the structural engineer has to rely on the building code or real wind tunnel studies to determine the wind loads on a building or structure. As the Dutch building code NEN 6702 and the Eurocode are limited to simple shapes, real wind tunnel tests are necessary to determine the wind effects on complex building shapes. Although these studies give reliable results for situations where the codes do not foresee, designers are discouraged to perform such a study in the early design process as they are very expensive and time-consuming. A third option to determine the wind loads on a building or structure could be the use of Computational Fluid Dynamics. In other industrial sectors CFD has become a very suitable tool for analyzing flow problems for which no analytical solution is available. CFD can solve the mathematical models up to a desired accuracy and changes in the geometry are relatively easy to perform. This can make CFD a very powerful tool to determine the wind effects in the early design process. Shape optimization with respect to wind loads could be possible then.

However, as there is still contradiction on the use of CFD in wind engineering, results of CFD calculations are not widely accepted yet. To develop a general approach for CFD in wind engineering, a computational wind tunnel has been proposed at the Structural Design Lab of Delft University of Technology. The goal of the so-called *Virtual Wind Tunnel* is an application that can be used in the early stage of the design process to determine the wind loads. In the recent graduation studies by Van Nalta [16] and Snijders [23] a computational domain has been developed that shows promising results for the pressure distribution on simple shapes, like cubes and cylinders. However, for more complex building geometry the computational demand seems too high for the current desktop computers. Nevertheless, anticipating on future developments of computer resources, there are still some fields that require further research. From the recent graduation studies it is concluded that the definition of geometry has to be improved. The conversion of a detailed CAD model of a building to a useful CFD model seems to be a major problem that seriously affects the applicability of the Virtual Wind Tunnel. To perform parameter studies and to improve the design, alternative geometries must be compared. Methods that are able to automatically create the geometry could save valuable time. Finally it is concluded that the computational domain should be tested for more complex geometry and for an environment that contains a number of buildings. In the recent studies only single objects of a simple shape were studied. The consequences of placing more complex or several building shapes in the domain should be examined.

From the conclusions and recommendations of the recent graduation studies it appears that certain tools to setup the geometry for CFD calculations are desired. The design tools that are developed in this thesis are to a large extent able to automatically generate the building geometry. With the toolbox it will be possible to setup a CFD calculation and compare several alternative geometries in a relatively short time. This makes shape optimization with respect to wind loads possible. As less interference of the structural engineer is required, valuable time will be saved as well. The purpose of the various design tools and their order in the geometry generating process is discussed in the next paragraphs.

6.2 Approach for the generation of geometry

During this Master's thesis several design tools will be developed to generate the geometry for CFD calculations. Looking at the entire CFD process, the tools concentrate on the first step, the description of geometry. In the previous graduation studies by Van Nalta [16] and Snijders [23] some other steps of the CFD process are investigated and the results of these studies will be used in this thesis to perform some CFD calculations. In Chapter 8 the results of some calculations for several test cases will be discussed.

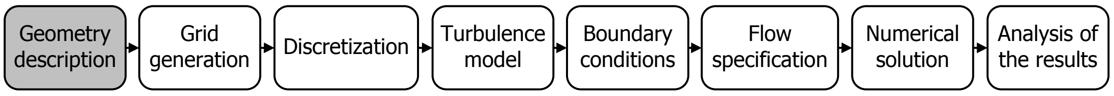


Figure 6.2: The design tools concentrate on the first step of the CFD process

In Figure 6.3 a general approach for the generation of building geometry for CFD calculations is given. The straight orange boxes represent the models or datasets from which the required geometry can be generated. The blue diamonds represent the procedures that have to be carried out to adapt the various models. The yellow straight boxes finally represent the results that follow from the various procedures. As the flow pattern around a building is influenced by its surroundings, it can be desired to take the environment into account in the calculations. Next to a model of the building of interest, a 3D model of the environment is required then as well. However, at the moment there are hardly any useable 3D models of a certain area available and a method has to be developed to generate such models. Examples of data from which the 3D models of the environment can be constructed are GIS datasets, separate building models or 3D scans of a certain area or building. Next to a model of the environment, a 3D CAD model of the building of interest is required as well. As calculations for global wind load determination do not require highly detailed building models, the CAD models can be simplified by neglecting small details to quicken the calculation process. The simplified model of the building of interest can then be joined with the 3D model of the environment to create an integrated model of the research area. The next step is to mesh the model to be able to perform a CFD calculation. In Paragraph 6.3 the strategy of the various design tools and their position in the geometry generation approach is discussed in detail.

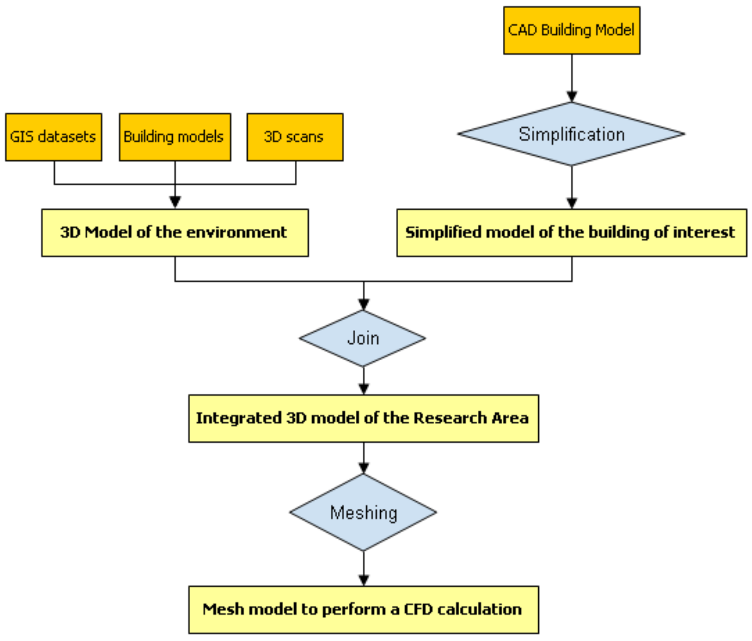

CAD Building Model
GIS datasets
Building models
3D scans
Simplification
3D Model of the environment
Simplified model of the building of interest
Join
Integrated 3D model of the Research Area
Meshing
Mesh model to perform a CFD calculation

Figure 6.3: General approach for the generation of building geometry

6.3 Strategy of the design tools

With the design tools that will be developed in this thesis, three types of geometry can be generated. The first design tools can be used to generate a 3D model of the environment for a certain area. Another design tool can be used to simplify the geometry of the building of interest and to remove little details. Joining the results of these tools gives an integrated model of the research area with a simplified model of the building of interest, surrounded by the buildings of the environment. A final design tool can be used to create the computational domain in which the CFD calculations are performed. In Figure 6.4 the strategy of the various design tools to generate the geometry for a CFD calculation is given, according to the general approach as discussed in Paragraph 6.2. The rounded green boxes represent the various tools and give their order in the total process. With the design tools the calculations to predict the wind loads on a building or structure can be setup in a relatively short time and alternative geometries can be compared without much interference of the user. This would make shape optimization with respect to wind loads possible. The purpose of the various tools is discussed more extensively in the rest of this paragraph; the development process of the tools is discussed in Chapter 7.

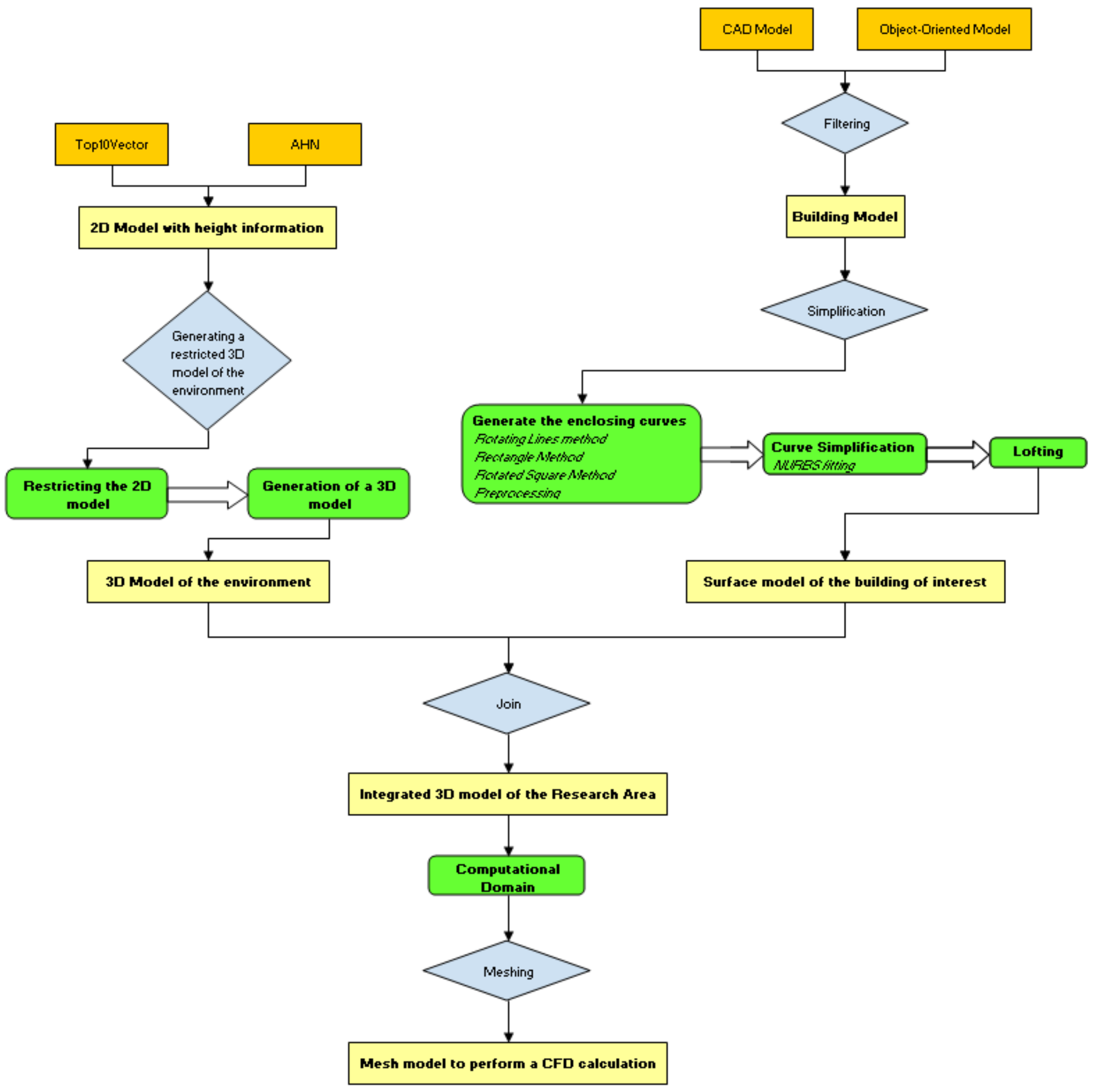


Figure 6.4: Strategy of the various design tools to setup the geometry for CFD calculations

Modeling the environment

GIS technology offers two datasets for the Netherlands that can be used to generate a 3D model of a certain area. The Top10Vector dataset provides a digital topographic map of the Netherlands in 2D which is composed of closed polygons that represent various objects on the earth's surface. Such objects are for example buildings, roads, water and vegetation. The other dataset is the *Actueel Hoogtebestand Nederland* (AHN), which provides an elevation model of the Netherlands. In this dataset the height of the country is stored on a regular grid with a density of one point per 16 m^2, where each point represents the height of that particular location. Joining the two datasets gives a 2D digital topographic map for a certain area with the accompanying height information.

Because the amount of built environment that has to be taken into account for the CFD calculations is limited, the first design tool can be used to create a restricted 2D research area from the Top10Vector dataset. For real wind tunnel studies a research area with a radius of 300 m around the building of interest seems sufficient. Buildings further away do not have a significant influence anymore on the flow pattern around the central building. For the CFD calculations the extent of built area that is represented will initially be similar as for real wind tunnel studies. By assigning the center location of the building of interest and the radius of the research area, the first design tool will be able to create a bounded 2D research area with height information that originates from the AHN dataset.

With the second design tool a 3D model of the environment can be generated by extruding the various elements of the 2D research area over their concerning height. The extrusion heights are derived from the AHN dataset for that particular area. Within the generated 3D model of the environment, the model of the building of interest can be placed to create an integrated 3D model of the research area. With this model it is possible to take the influence of the surroundings into account in the calculation of the wind loads on the building of interest.

Simplification of the building of interest

To determine the wind loads on the building of interest, a CAD model of the building is required. Nowadays these models usually originate from traditional CAD or object-oriented systems. However, CAD models of a certain building often contain lots of information that are not relevant to determine the global wind loads. Little details have hardly any influence on the wind effects, but a very complex grid is required to mesh all these small details. This increases the calculation time of the CFD software considerably. Inner geometry is also not relevant as only the surfaces that come in direct contact with the flow are necessary for CFD analysis. Nevertheless, deriving just the exterior surfaces of a building model is very hard as façades for example are usually constructed from several elements, like bricks, window-frames, glass and awnings. It is not possible to subtract a single surface from the façade then. A final aspect is that building models have to be fully airtight for CFD analysis. Drawing inaccuracies can introduce small gaps through which air can flow into the building, which is off course not permitted.

The purpose of the third design tool is to simplify the building of interest by wrapping an airtight, fitting surface around the model. The wrapped surface replaces all internal and external geometry and excludes small details. It is only this wrapped surface model that will be used for the CFD calculations. The procedure to wrap a surface around the building model consists of several steps. First enclosing curves are generated around the model at different height levels. Several methods will be developed to create these curves. Detailed façades can result in highly fluctuating enclosing curves, which can be simplified by fitting a NURBS curve through it. Finally a surface can be lofted through the curves that wraps the original building model.

As discussed in Paragraph 5.5, recent developments in CAD have resulted in new technologies, like object-oriented and parametric modeling systems. As architects and structural engineers work with these various systems nowadays, the design tool to simplify the building of interest must support the different models that originate from the various CAD systems. To quicken the simplification process, the CAD building models can be filtered first by manually deleting some inner geometry that is not relevant to determine the global wind loads. One can filter a building model on element size or on element type for example. As most models are constructed in several layers to distinguish the various building elements, the irrelevant parts can be removed by manually deleting the elements of a specific layer.

The simplified model of the building of interest and the 3D model of the surrounding buildings together will form an integrated 3D model of the research area. All procedures to setup this area with the various design tools will be executed in the 3D CAD environment Rhinoceros (Internet, [1]). As both the model of the environment and the simplified building model are generated in Rhinoceros, it is the intention to manually place the building model in the 3D model of the environment. However, the Top10Vector and AHN datasets originate from GIS technology and all data correspond to the global coordinate system. As building models are usually generated in a local coordinate system, both models use different systems to define the coordinates. When the models are joined, they are placed at different locations and some operations are required to move the building model to the right place in the model of the environment. Although it will not be developed in this thesis, an additional design tool could be generated that links the local coordinate system with the global coordinate system. When the user knows the global coordinates in the model of the environment where the building of interest has to arise, a design tool should be able to place the building model with the right orientation at the desired location.

Generation of the computational domain
The last design tool that will be developed can be used to generate the computational domain in which the calculations are performed. The domain must be large enough to avoid that the flow pattern at the boundaries of the domain is influenced by the buildings in the research area. The Van Nalta domain (Nalta, [16]), which is developed at the Structural Design Lab of Delft University of Technology, forms the base for the computational domain that is used in this thesis. The Van Nalta domain was developed to investigate the wind effects on a cube with dimensions of 60x60x60 m^3. However, for calculations with a research area containing several building models, the dimensions of the original domain are not sufficient. For each calculation the domain has to be adapted, as the dimensions depend on the maximum height of the buildings in the research area and the radius of the research area.

The purpose of the fourth design tool is to create the computational domain, depending on the dimensions of the research area. The domain is divided in two parts, a *wind environment* and a *building environment*. In the building environment the model of the research area will be placed and this part of the domain has to be meshed manually. The wind environment contains the regions around the building environment and will be meshed automatically during the generation of the domain. The structure of the Van Nalta domain is discussed more extensively in Paragraph 7.4. After the entire domain is meshed and boundary zones are defined, the calculation can be setup in Fluent. The guidelines that were developed in the previous graduation studies will be used for this procedure. The results of some calculations with models that are generated with the various design tools are discussed in Chapter 8. By going through the entire CFD process it will be investigated if the various design tools work for CFD applications. Comparing the results of a CFD calculation with the Dutch building code or the Eurocode would be very interesting then.

7. Tools to setup the geometry for CFD calculations

In this chapter the design tools to setup the geometry for CFD calculations are discussed in detail. The first and second tools to generate a 3D model for a certain area are demonstrated for a part of the Delft University of Technology district. For the third tool that simplifies the model of interest, several methods are developed. These methods are discussed extensively and explained on the basis of some simple models. The fourth tool can be used to generate the computational domain for a certain research area, depending on the dimensions of that area. The Van Nalta domain, which was already developed at the Structural Design Lab, forms the base for the generation of the domain. All developed procedures, methods and scripts originate from own ideas and work. Only the method that directly finds an intersection between a mesh object and a line is developed with some outer help. This method is then implemented in the own developed methods that are discussed in Paragraph 7.3.

7.1 Tool 1: Generation of the 2D research area

7.1.1 Purpose

The flow pattern around a building is strongly influenced by its surroundings. That is why the surroundings are of great importance for the CFD calculation. The amount of built environment that has to be taken into account can be limited. In real wind tunnel studies a circular area with a radius of 300 m around the building of interest is usually modeled (Franke, [8]). Buildings further away have no influence of concern on the flow pattern around the region of interest. In accordance to real wind tunnel tests, the extent of built area that is represented in the CFD calculations will initially be similar.

Information about the built environment can be obtained from GIS technology. But until now there are no useable 3D models of certain regions of the Netherlands available. However, as mentioned in Chapter 4, research is done to create a digital topographic map of the whole country in 2D. Besides, a detailed elevation model of the Netherlands is built where the height of the ground level, built areas, dykes, rivers and roads are modeled. With these two models it should be possible to create a 3D model of the built environment for a certain area.

The purpose of the first design tool is to generate a 2D research area from the Top10Vector dataset for use in the Virtual Wind Tunnel. When the user specifies a certain location of the building of interest, the design tool should be able to automatically create a circular research area with a certain radius around the central building. All objects outside this area can be removed, only the inner objects will remain. The 3D CAD environment Rhinoceros is applied to create the tool. Rhinoceros is able to read both the digital map of the Netherlands and the elevation model of the country. Furthermore, it is possible to script with Rhinoceros, what makes it a very suitable environment to create the design tool. For scripting, Rhinoceros uses a specialized version of Microsoft's Visual Basic language, called Rhinoceros Visual Basic. This version adds over 200 methods to the native Visual Basic language, which enables the programmer to work with specialized geometry, like meshes and NURBS curves and surfaces.

7.1.2 GIS data

The datasets that are used to generate the 3D model of the built environment are the Top10Vector and the AHN. These datasets are obtained in DXF format from research institute OTB in Delft. The Top10Vector dataset contains a map of the southern part of Delft at scale 1:10.000. The AHN dataset contains the heights of the surface and all kind of objects for a part of the Delft University of Technology district. The heights are stored in points that lie on a regular grid with a point density of 1 point per 16 m^2.

A 2D model of the university district, where the two datasets are joined together, is shown in Figure 7.1. Figure 7.2 gives a closer view of the district to show the rectangular grid of points of the AHN that lies over the Top10Vector dataset.

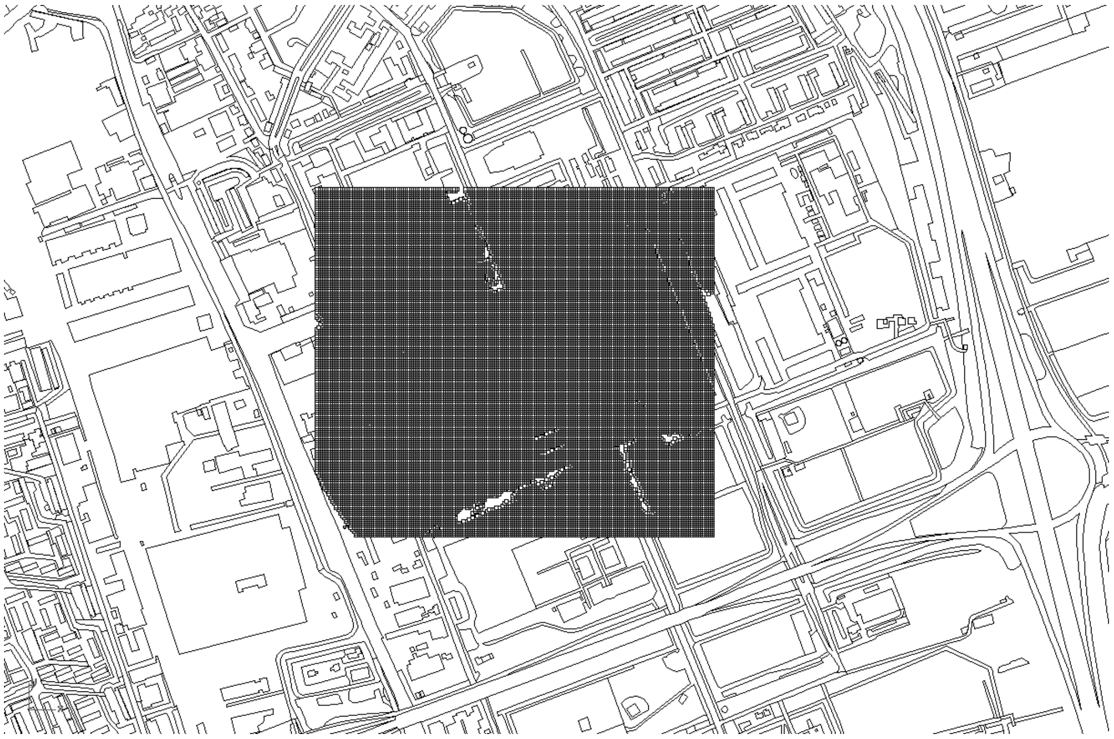


Figure 7.1: 2D model of the University of Delft district; Top10Vector and AHN together

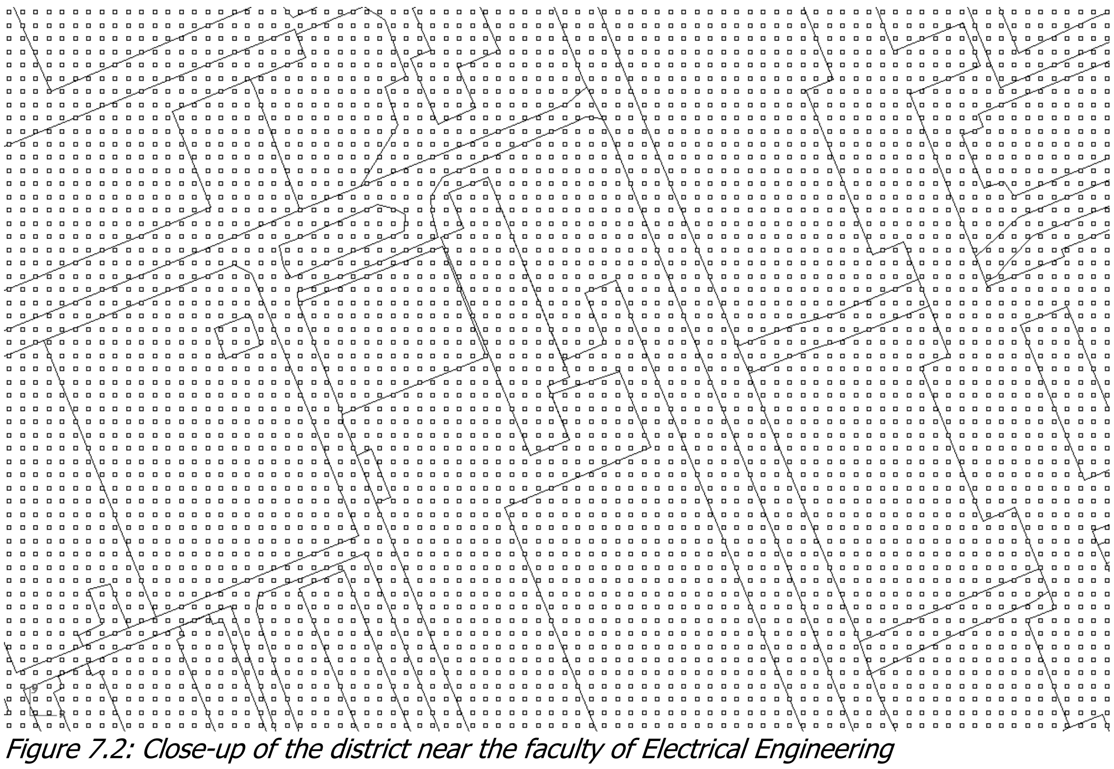


Figure 7.2: Close-up of the district near the faculty of Electrical Engineering

The points itself of the AHN dataset have no height, they all lie on a plane with zero elevation in z-direction. The height of the area they represent is stored in the layer of the points. Each point has its own layer, where the name of the layer corresponds to the concerned height.

The Top10Vector is a vector data model that is composed of only closed polygons that represent the various elements on the earth's surface. To distinguish these elements there is a distinction in layers of the polygons. Every element type has its own layer with a unique code, wherein all polygons are subdivided. There are for example separate layers for roads, buildings and vegetation. The several layers can be turned on and off separately. Appendix B gives a list of all layers in the Top10Vector dataset, the so-called TDN code (Internet, [2]).

7.1.3 Explanation of the tool

Before running the script, the user has to open both the Top10Vector and AHN datasets in Rhinoceros. When the script is loaded, the user is asked for the radius of the research area. This is 300 m by default, but the user is able to give any radius. After entering the desired radius the user is asked to pick the center location where the building of interest will arise. The user can do this by simply mouse clicking on a certain location in the map. After this the script will first turn off all layers in the Top10Vector dataset that do not represent any building type. The purpose is to create a 3D model of the buildings inside the research area only. Elements like roads, rivers or vegetation are not modeled, because this would complicate the grid generation process considerably. In a further phase of the script a flat ground surface will therefore be created that corresponds to the size of the research area.

When the script has turned off the various layers, a circular area around the center location of the building of interest will be created with the entered radius. The script will now check which polygons representing the built environment and which height points lie at the outside of the circular area. These will be deleted then. When a polygon crosses the boundary of the area, the polygon will be kept. Initially it was intended to trim the polygons at the boundary, where the part at the outside of the circular area would be deleted. However, scripting this procedure appeared to be very complicated. Besides, when a polygon is trimmed at the boundary, only a part of the building would be modeled then. This does not correspond with the reality and it also involves an abrupt change of the built environment at the boundary. Therefore it is decided to keep the complete building when it is located at the boundary of the circular area. To check if a polygon lies outside the research area, a Rhinoceros command is used that returns all control points of a polygon. For each control point it will be checked if its distance to the center location of the research area is larger or smaller than the user-defined radius of the research area. The mathematics for this procedure are:

$$\begin{aligned} \Delta x &= (control\ point)_x - (center\ location)_x \\ \Delta y &= (control\ point)_y - (center\ location)_y \\ distance &= \sqrt{(\Delta x)^2 + (\Delta y)^2} < radius \end{aligned} \tag{7.1}$$

If the distance for one or more control points of the polygon is smaller than the radius, the polygon lies completely or partly in the research area. This polygon must remain then. When the distance of all control points is larger than the radius, the polygon lies completely outside the research area. Subsequently, the script checks which height points that lie in the research area also lie in a building polygon. A special plug-in for Rhinoceros is used for this procedure, which automatically finds all points that are enclosed by a certain polygon. All height points that lie in the research area and are enclosed by a building polygon are kept; the remaining points are deleted. Because the polygons that cross the boundary of the circular area are kept, the actual research area is larger than the original area. That is why the script finally deletes the original circle and creates a new circle that fits all polygons. Figure 7.3 gives the result of the first tool for a part of the Delft University of Technology district. All polygons and height points at the outside of the research area are deleted. Within the research area, only the points that contain the height of the remaining building polygons are kept.

The variables which the user has to specify are summarized in Table 7.1.

Variable	User specified value
Radius of the research area	
Center location of the building of interest	

Table 7.1: Variables to specify when running the script of the first design tool

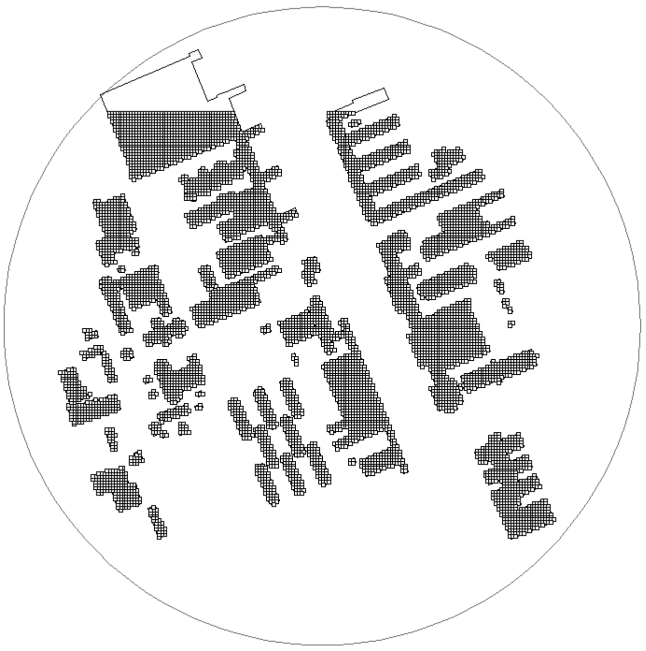
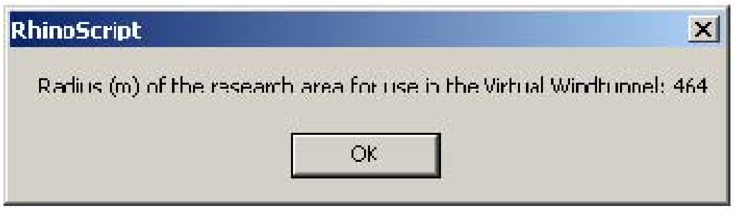


Figure 7.3: Result of the first tool: A 2D model of the research area with the necessary height points lying at the inside of the polygons

As can be seen in Figure 7.3, there are no height points at the top side of the upper polygons. This is caused by the relatively small amount of available height data of the university district. However, the method seems to work properly. When in the nearby future the Top10Vector and AHN datasets are available for other areas, the method will also be suitable to generate a research area from these datasets. This of course on condition that the datasets are received in the same format; that means the heights in the AHN dataset have to be stored in the layer of the various points. Because all polygons that cross the boundary of the research area are kept, the actual research area is larger. The script draws a new circle with a larger radius that fits all polygons. The center location of this new research area corresponds to the user-specified center location of the building of interest. In this way the center location of the building remains the center of the research area. The whole model is then moved from the center location of the research area to the origin of the coordinate system. As the computational domain that will be constructed around the research area to perform the CFD calculations is also generated around the origin, the model of the research area is directly in the right place when it is imported in the domain. Finally the radius of the research area is given to the user. The radius is required to determine the dimensions of the domain. In Appendix C the first tool is discussed extensively on the basis of the written script.


RhinoScript

Radius (m) of the research area for use in the Virtual Windtunnel: 464

Figure 7.4: The tool finally gives the radius of the research area

7.2 Tool 2: Extrusion of buildings inside the research area

7.2.1 Purpose

With the first design tool a 2D model of the region of interest is generated. The buildings inside the research area are represented as polygons. These polygons enclose several height points that contain information about the elevation height of the building that is represented by the polygon. For use in the Virtual Wind Tunnel a 3D model of the region of interest is required. So, the purpose of the second design tool is to generate an elevated model from the 2D model that is created by the first design tool. The script of the second design tool is again written in the Rhinoceros Visual Basic language.

As mentioned in Chapter 3 two measurements have to be carried out during a real wind tunnel test: one measurement where the surrounding buildings are modeled as the actual situation and a second measurement where the surrounding buildings are chopped off to a maximum height of 15 m. The maximum values obtained from these measurements are normative. The user of the Virtual Wind Tunnel must also be able to do these two measurements. The second design tool will therefore be able to extrude the buildings inside the research area over both the actual height and, if a building is higher than 15 m, over the truncated height.

7.2.2 Explanation of the tool

Before running the script of the second design tool, the user first has to open the 2D model of the research area in Rhinoceros. When the script is loaded, the user is asked to choose the extrusion height of the surroundings. The buildings can be extruded over their full height, over a truncated height if a building is higher than 15 m and over no height at all. With the last option the user is also able to simulate the flow pattern around a building without modeling the built environment. After an extrusion method is chosen, the script checks for all polygons in the model which height points are located in the area surrounded by a polygon. A special Rhinoceros plug-in is used to automatically determine which points are enclosed by the polygon. The layers of these points contain the height over which the concerned polygon has to be extruded. The script determines then the average height of all points that are enclosed by the polygon and extrudes the polygon over this averaged height. However, due to inaccuracies in the AHN dataset, some height points differ considerably from the real height. The divergences originate from for example inaccuracies of the laser scanning equipment. The dataset also contains blunders that are caused by disturbing objects on the earth's surface, like vegetation, water and mobile objects. Figure 7.5 shows some correct and divergent height points that lie in the polygon of the faculty of Electrical Engineering.

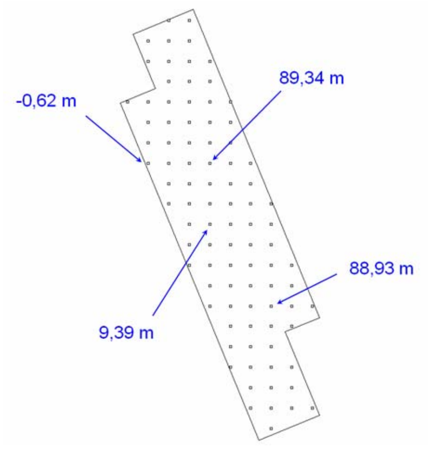
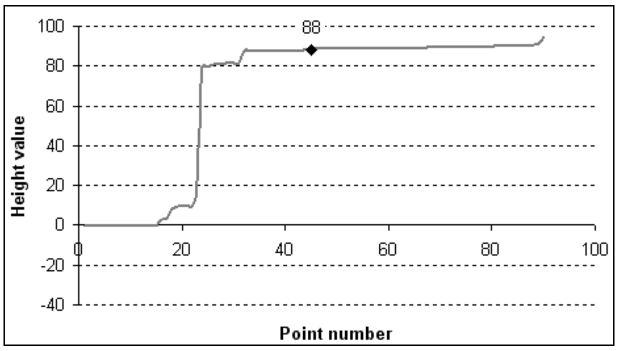


Figure 7.5: Correct and divergent height points in the faculty of Electrical Engineering polygon

The correct height of the building is about 89 m, but there are points of 9,39 m and even -0,62 m. By taking the median height of all points as extrusion height in stead of the average height, the divergent height points can be neglected. As the median is the middle of a distribution, the list of height values for a certain polygon has to be arranged first from the lowest value to the highest value. Sorting the list is easily done in Rhinoceros with one simple command. If a certain polygon contains an uneven number of height values, the median is just the middle value of the sorted list. If there is an even number of height values of a certain polygon, the median is the average of the two middle values of the list. In Figure 7.6 the values of all height points in the faculty of Electrical Engineering polygon are sorted and represented in a graph. The median of the values is 88 m and corresponds to the middle point of the sorted height points.



Uneven number of height values:

$$Median = \left(\frac{n+1}{2}\right)^{th} value\ of\ list$$

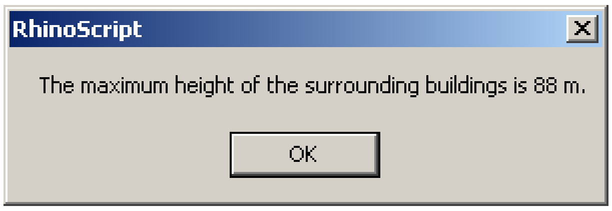
Even number of height values:

$$Average = (value\ below\ median\ + \\ value\ above\ median)/2$$

Figure 7.6: Median of the height values of the faculty of Electrical Engineering polygon

When the polygon is extruded over the median height, the result is a closed solid with the shape of the underlying polygon. If the user has chosen the option to chop off the buildings that are higher than 15 m, the script checks if the averaged height of the points is larger than 15 m. If so, the concerned polygon is extruded over 15 m. Smaller buildings are just extruded over their actual height.

When all buildings are extruded, the script finally extrudes the circular ground plane that was created by the first design tool. The circle is extruded over a constant height and forms a flat ground surface that corresponds to the size of the research area. Finally the script informs the user about the maximum height over which a polygon is extruded. This corresponds to the highest surrounding building in the research area. The maximum height of surrounding buildings and the building of interest determines the size of the computational domain that will be used for the CFD calculations. In Paragraph 7.4 the generation of the computational domain is discussed further. Figure 7.8 on the next page shows the result of the second design tool for a part of the Delft University of Technology district.



The maximum height of the surrounding buildings is 88 m.

Figure 7.7: The tool finally gives the maximum height of the surrounding buildings in the research area

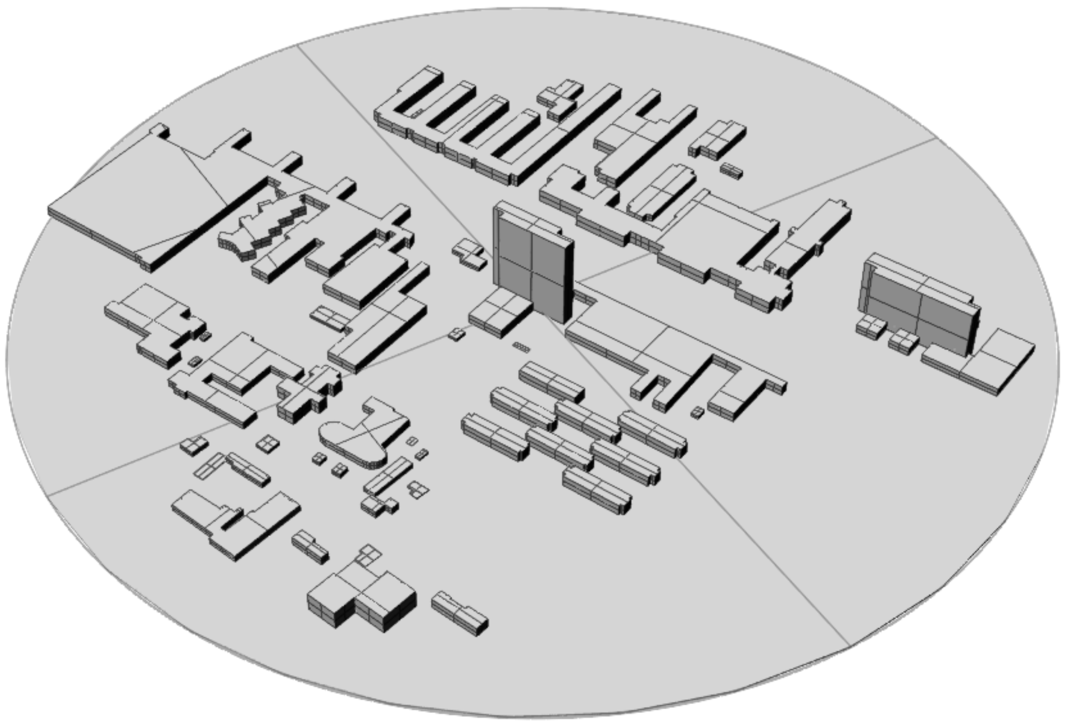


Figure 7.8: Result of the second design tool: A 3D model of the research area with a flat, circular ground surface

The 3D model of the research area can be saved in a wide variety of formats, so it can be used in many applications. To investigate the wind forces and pressures on a new building, the user can manually add a 3D CAD model of the building of interest into the model of the research area. This can be done in Rhinoceros, but when the modeled area is saved in the right format, it should be possible to do it in other CAD environments as well. Adding a new building to the research area is not definitely necessary. The model could also be used to investigate the wind effects on an existing built environment, for example to determine the most suitable location for a specific new building. The user could also use the 3D model to investigate the effects for the surrounding buildings when a particular building in the research area is removed. Finally, the model could also be used as input for scaled model manufacturers to create a real scaled model of a certain area. These models could for example be used for real wind tunnel research.

In Appendix D the second design tool is discussed extensively on the basis of the written script. The variables which the user has to specify are summarized in Table 7.2.

Variable	**User specified value**
Extrusion method: - Extrusion over the full height - Extrusion over the truncated height - No extrusion at all	

Table 7.2: Variables to specify when running the script of the second design tool

7.2.3 Points of concern

Although the second design tool seems to work properly, there are some points of concern. The first problem is that an entire building is represented by only one polygon. When a building consists of several building parts with a different height, the polygon is extruded over one height, which is the average of all building parts. The left picture of Figure 7.9 shows the extruded faculty of Civil Engineering. In reality the building consists of several parts with their own height. The lecture halls for example are about 9 m high, the Stevin labs about 12 m and the main building about 33 m. However, because the building is represented as one polygon, it is extruded over the average height of all building parts, which is 12,5 m.

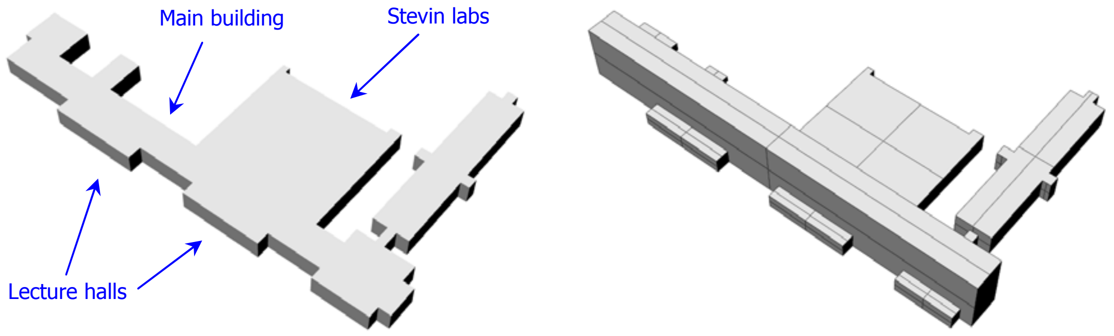


Figure 7.9: all building parts of Civil Engineering extruded over the average height (left)
all building parts of Civil Engineering extruded over their concerning height (right)

There seems to be methods available that solve these problems. For the points of the AHN dataset a very complicated algorithm can recognize a large change in height. It seems possible to create a new polygon automatically around the divergent height points. These new polygons could be extruded over the corresponding height then. However, due to the relative coarse point density of the AHN dataset, the shape of these new polygons will probably not exactly match the real shape of the building parts. With more height points per m^2 a more accurate result could be obtained. A better method seems to manually create polygons in the Top10Vector dataset for areas with divergent height points and separate these from the original polygons. The different building parts could then be extruded over their concerning height. In the right picture of Figure 7.9 this method is implemented for the faculty of Civil Engineering. As all building parts are extruded over different heights, the model describes reality much better. However, a thoroughly knowledge of the buildings in the environment is necessary for this method, which is usually not the case.

Another aspect is that only the polygons that represent buildings are extruded. All other polygons are turned off, so roads, water and vegetation for example are not modeled. In stead of these elements, a flat ground surface is modeled to simplify the grid generation process. A complex surface with lots of small height differences would cause a very complex grid, which results in less cells being available for modeling the primary buildings. For use in a design application a very high accuracy of the ground surface might not be that necessary. However, trees for example could still be modeled. If for example a certain tree form is modeled, it can be placed in the model of the research area at the concerning locations that follow from the tree layers in the Top10Vector dataset. With the elevation data that follow from the AHN dataset for these locations, the tree models could be scaled to the right dimensions. In this way the influence of large vegetation can be taken into account in the model of the research area. This option could be a useful extension of the second design tool.

A final point of concern is that still no slope surfaces like roofs can be modeled. All polygons are extruded straight up and have a flat top side. Other surfaces, like for example the slope underside of the Aula Congress Centre of the Delft University of Technology can not be modeled as well. The building is represented as a solid block with straight walls. However, as the models of the environment are used in a design application, it might not be that necessary for now to have a very high accuracy of the surrounding buildings. At GIS technology, research is done at the moment to develop more accurate and airtight 3D models of certain environments. If these methods come widely available in the future, more accurate models of the built environment can be generated for use in the Virtual Wind Tunnel. However, one must keep in mind that the purpose of the Virtual Wind Tunnel is a design application with which the global wind forces can be calculated as an indication. Highly detailed models of the environment are therefore not that necessary.

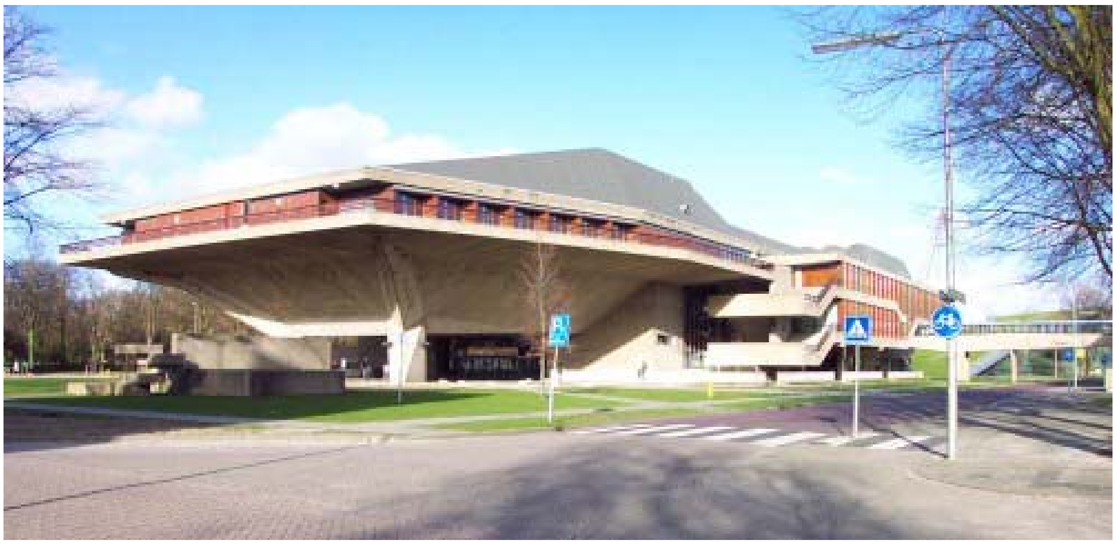


Figure 7.10: Slope surfaces can not be modeled with the current design tools (Internet, [15])

7.3 Tool 3: Simplification of the building of interest

7.3.1 Purpose

CAD models of a building usually contain lots of information that are not relevant to determine the global wind loads on a structure. Modeling all little details, like window frames, door handles and railings will cause a very complex grid that increases the calculation time of the CFD software tremendously. For CFD analysis only the surfaces that come in direct contact with the flow are necessary. But most building models also contain the interior structures and several furniture objects in high detail. These objects are certainly not relevant to determine the wind loads. A method that automatically removes the interior geometry and simplifies the external geometry to a certain degree by neglecting small details would be very valuable as this can save a lot of calculation time. For meshing, also a lot of cells can be saved when not all little details are modeled. This is also valuable, as the amount of cells that can be used is limited due to the actual computer capacities. Meshing large building models that are placed in a built environment can therefore be very difficult.

Figure 7.11: Highly detailed external and internal geometry in CAD models (Internet, [18])

For CFD analysis the provided input models have to be airtight, meaning that all lines and surfaces have to connect exactly. However, this is not always the case as edges, surfaces and solids do not always meet in a CAD model of a building design. These gaps are generally introduced by inaccuracies of the drawers and give problems with the grid generation as the meshing algorithms require a perfect model description as input. Gaps in a model could also cause air flowing through the building, which is of course not permitted. By wrapping a surface around the building model an airtight model could be created that is very suitable for CFD simulations. If the shape of the model is followed to some degree by the wrapped surface, a simplified building model can be generated that excludes all little details.

The purpose of the third design tool is to simplify the external geometry of the building of interest by wrapping a fitting surface around the model. The wrapped surface heals eventual gaps, which makes the model airtight, and replaces the external and internal geometry of the building. It is only this wrapped surface that will be used for the CFD analysis. The following paragraphs discuss four methods that have been developed to create the wrapped surface. The methods are again scripted in the Rhino Visual Basic language and are based on finding the most outer points of the model for different height levels. If for a certain level the outer points are found, a polygon can be generated through the points, which encloses the model at that level. If this process is repeated for the full height of the building, where the level of concern is elevated each time over a certain distance, the model is enclosed by an amount of polygons. A surface can then be generated through these polygons, which wraps the model. For all three methods the process is divided in several parts. First the enclosing polygons for the several height levels are generated. When the model is highly detailed this will lead to very fluctuating polygons. To simplify these curves and get rid of little details, a smooth NURBS curve can be fitted through the fluctuating curve. Finally a surface can be generated through the enclosing curves that wraps the model.

7.3.2 Rotating Lines method

The first method to generate the enclosing curves around a model is called *Rotating Lines*. The method is based on finding the outer points of the model. The method is first developed in 2D and later extended to 3D. When the building model is opened in Rhinoceros and the script is loaded, the user is asked to pick the center location of the model with the mouse. From this location, lines will be drawn with a user-specified length. For each line it will be checked if there is an intersection with the various elements of the model. The script recognizes intersections with lines, surfaces and solids. If there is more than one intersection for a certain line, the script will search for the intersection point with the largest distance from the center location of the model. At this location the script adds a point to the model. This procedure is now repeated for the next line, where each new line will be rotated over a certain angle around the center location of the model. The user has to specify an amount of parts from which the rotation angle can be calculated. If, for example, the amount of parts is 90, the angle over which the lines will be rotated is 360/90 = 4 degrees. The following pictures explain the method on the basis of a very simple 2D model.

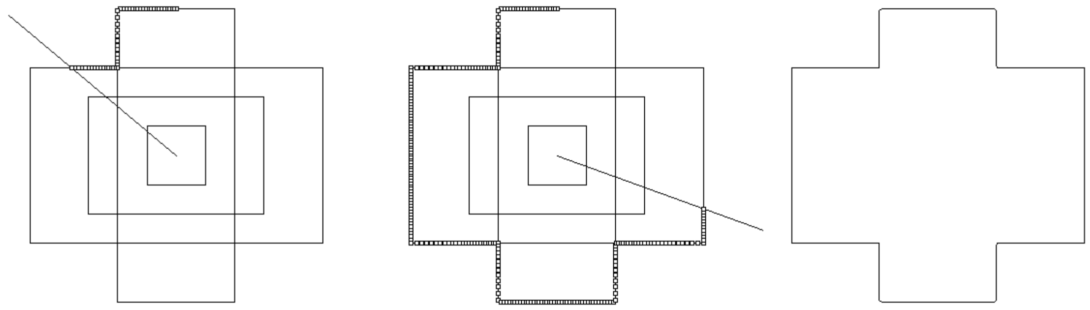


Figure 7.12: Rotating Lines method; 2D example

In Figure 7.12 a simple model that consists of rectangular shapes is given. The method starts with drawing a line from the user-specified center location of the model, with a user-specified length in the north direction. The length of the line must be higher than the largest distance between any point of the model and the center location. Else the line won't reach these points. The script uses some Rhinoceros commands that automatically determine the intersection between a line element and other lines, surfaces or solids. The distance between the intersection point and the user-defined center location of the building is then calculated.

$$\begin{aligned}\Delta x &= (intersection\ point)_x - (center\ location)_x \\ \Delta y &= (intersection\ point)_y - (center\ location)_y \\ distance &= \sqrt{(\Delta x)^2 + (\Delta y)^2}\end{aligned} \quad (7.2)$$

For a certain line element, a point is added to the model at the intersection with the largest distance to the center location of the building. The procedure is repeated for the next line, which is rotated around the center location. In this example the amount of parts that is specified is 360, what means that the rotation angle is 1 degree. So, for each degree the most outer point of the model will be found. When the lines are finally rotated over 360 degrees, the outer points in all directions are found. By adding a polyline through these points and removing the inner geometry, the model is enclosed by a surrounded curve. The result is shown in the right picture of Figure 7.12.

The procedure can now be extended to the third dimension. The level at which the outer points are determined is raised each time in order to generate the enclosing curves at different heights. When this is done for the full height of the building, a surface can be lofted through the various curves. The lofting process however is a separate method and will be discussed in Paragraph 7.3.8. Figure 7.13 explains the *Rotating Lines* method for a simple 3D model.

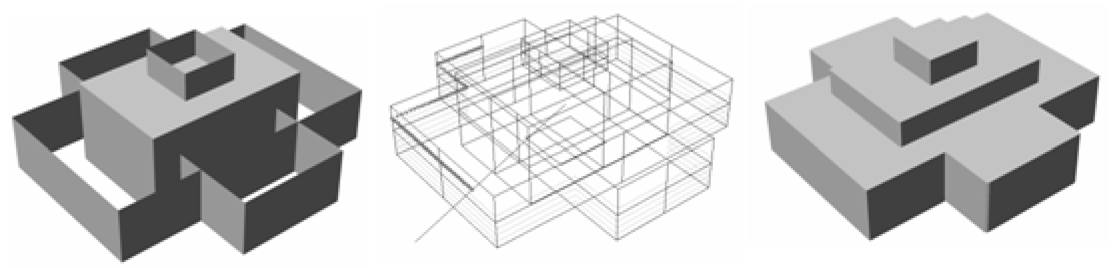


Figure 7.13: Rotating Lines method; 3D example

In Figure 7.13 a 3D model is given, based on the 2D model of Figure 7.12. The model consists of surfaces and solids. After loading the script, the user is again asked to pick the center location of the model. He is also asked for the length of the rotating lines and the amount of parts for which the outer points have to be determined. Because the script now works in three dimensions, the user is also asked for the height of the building and the step size over which the concerning level has to be elevated. Finally the user is asked for the height where the script has to start. Usually this is the bottom of the model. In the middle picture of Figure 7.13 some already generated enclosing curves can be seen. The total height of the model is four units; the step size is 0.25 units. When for all steps the outer points are found and the curves are generated, a lofted surface can finally be generated with the *Loft* method that is discussed later. The result is shown in the right picture. All underlying model parts are removed, what is left is a hollow surface model. The variables which the user has to specify are summarized in Table 7.3.

Variable	**User specified value**
Center location of the model of interest	
Length of the rotating lines	
Amount of parts to determine the outer points for	
Height of the model	
Step size to elevate the level of concern	
Starting height of the script	

Table 7.3: Variables to specify when running the script of the Rotating Lines method

The method also seems to work properly on more complicated models. However, complex building models will also contain more detail. As a consequence, the enclosing curves are not smooth anymore. To simplify the fluctuating curves and get rid of the little details, special fitting algorithms can be used. In Paragraph 7.3.7 the NURBS fitting technique is discussed that can be used to simplify the enclosing curves.

In Appendix E the *Rotating Lines* method for 3D models is discussed extensively on the basis of the written script.

7.3.3 Rectangle method

Another method that is developed to find the outer points of a model is a method called *Rectangle*. A disadvantage of the *Rotating Lines* method is that the distance between the outer points increases as the distance between the boundaries of the model and the center location of the model increases. For very large models with a rectangular shape, the shape of the enclosing curve will differ from the real shape. This phenomenon can best be demonstrated with an example.

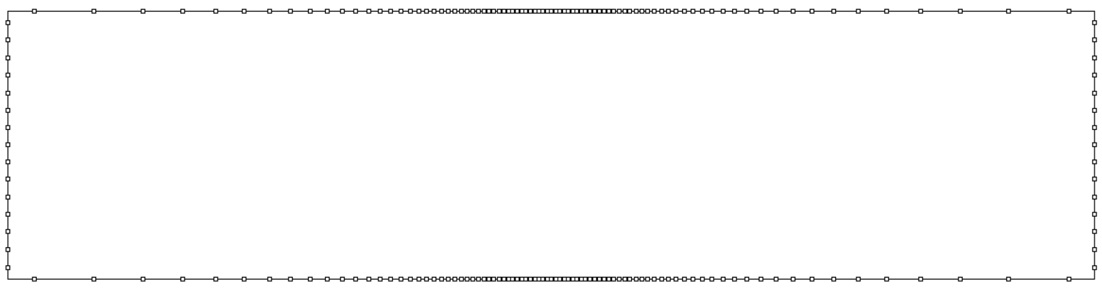


Figure 7.14: Disadvantage of the Rotating Lines method for large models: when a curve is generated through the points, the curve differs from the original model

As a consequence of the rotation of the line, the distance between the outer points increases when the intersection locations are further away from the center location. When a polyline is created through these outer points, the enclosing curve will differ at the corner points from the original model. A method that deals with this problem is the *Rectangle* method.

The *Rectangle* method is also based on finding the outer points of a model by the intersection of a line with the various elements of the model. In stead of lines that rotate around a center location, a rectangle that is drawn by the user and encloses the model will now be used as basis for the lines. Starting from the lower left corner of the rectangle, a line will be created to the upper left corner of the drawn rectangle. After having checked if there is an intersection with the model elements, the line will be moved to the right over a user-specified distance. For this new location of the line it will again be checked if there is an intersection with the various elements. For all intersection points of a certain line element the distances are compared and the minimum and maximum values are derived. At these locations points are added to the model then. This process is repeated until the line element is finally arrived at the right side of the rectangle. The procedure now continues with a line that is created from the lower left to the lower right corner of the drawn rectangle. After having checked if there are intersections with the model elements, the line is moved upwards over the same user-specified distance. This process continues until the line is finally arrived at the top side of the rectangle. The method is explained with pictures on the following page on the basis of the simple 2D model.

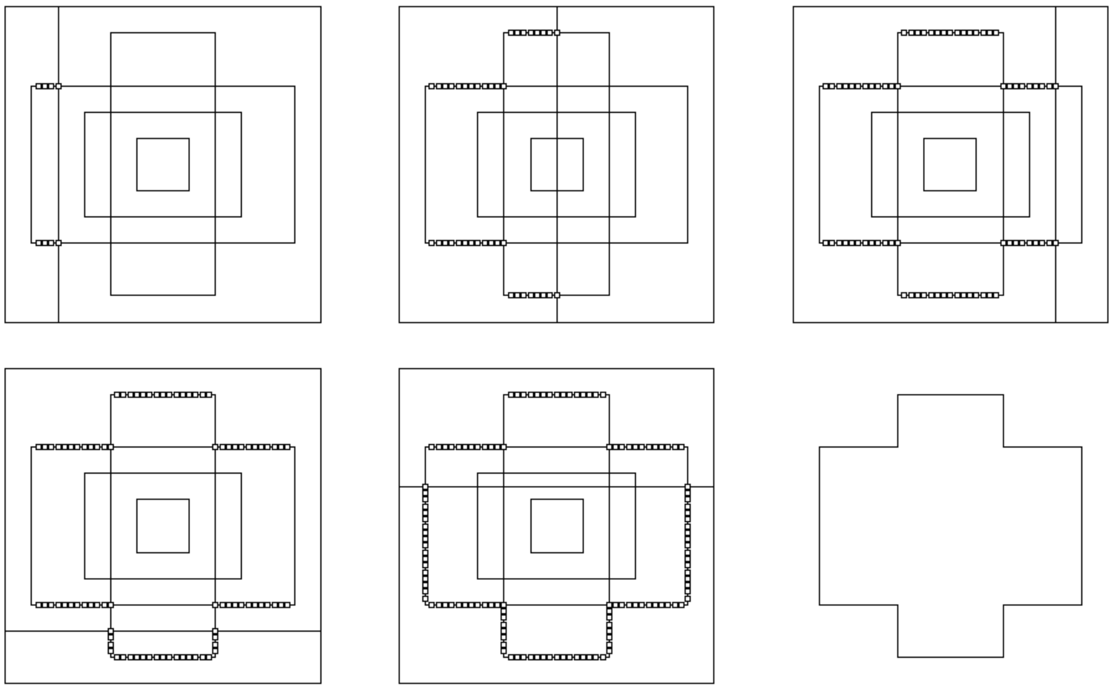


Figure 7.15: Rectangle method; 2D example

In Figure 7.15 the simple 2D model is given again. When the script is loaded, the user is first asked to draw a rectangle around the model. Then a line is created from the lower left corner of the rectangle to the upper left corner. The script checks if there are intersections between the line and the model parts and, in case of several intersections, automatically places a point at the nearest and most far away intersection location from the bottom of the rectangle. The next line that is drawn is moved to the right over the user-specified distance. When the line is finally arrived at the right side of the drawn rectangle, the process is repeated, but now in vertical direction. Starting from the lower left to the lower right, lines are drawn and moved upwards over the user-specified distance. The process ends when the line is arrived at the top side of the rectangle. All outer points are found now.

The next step is to draw the enclosing polyline through the outer points. However, this procedure is more complicated in comparison with the *Rotating Lines* method. For the *Rotating Lines* method, all outer points lie in the right sequence in the outer points array because of the rotation of the lines. The neighbouring outer points lie next to each other in the array. With the *Rectangle* method, the outer points do not lie in the right sequence in the array. The nearest and most far away intersection points for a certain location of the line are placed next to each other in the array. When directly creating a polyline through the points of the array, a zigzagging curve would arise. So, the points in the array have to be ordered first before the enclosing curve can be drawn through it.

The procedure of ordering the array of outer points is based on finding the closest point for a certain outer point. Starting from the first point of the array, the distances from all other points in the array to this first point are calculated. The closest point is now placed in a new array. From this point the distances to all other points are calculated again to find the closest point for that one. The closest point that is found now is again placed in the new array. Points that are already placed in this new array are ignored by the determination of the closest points. The reason is that it could else be possible for a certain point that the distance to a point in the new array is smaller than the distance to a point that is not already placed in this array. When all points are gone through, a new array is originated with the outer points lying in the right sequence. Neighbouring outer points now lie next to each other in the array. After adding a polyline through these points and removing the inner geometry, the model is enclosed by a surrounded curve. The result is shown in the lower right picture of Figure 7.15.

The procedure can again be extended to the third dimension. The level at which the outer points are determined is raised each time to generate the enclosing curves at different heights. Figure 7.16 demonstrates the method for the simple 3D model.

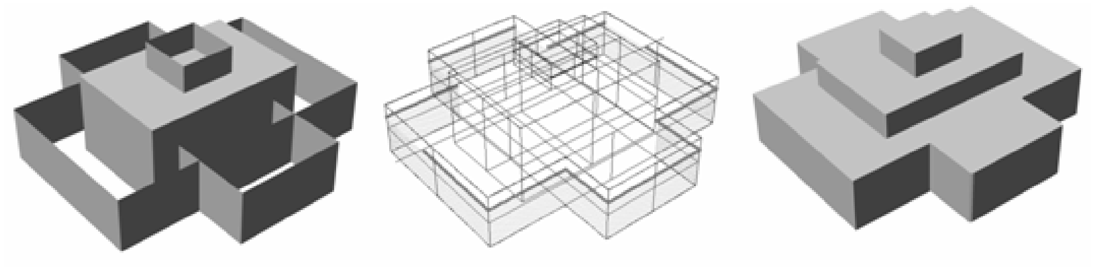


Figure 7.16: Rectangle method; 3D example

The model in Figure 7.16 consists of surfaces and solids. After loading the script, the user is asked to draw a rectangle around the model. The script uses the dimensions of this rectangle to determine the length of the curves and the boundaries between which the curves will move. The user is also asked for the distance between the line elements over which the lines have to be moved. If the user wants a very accurate enclosing of the model, the distance between the line elements must be small. However, a small distance between the line elements will lead to more outer points and the process of finding the closest nearby point to generate a polyline will take longer. The user is also asked for the height of the building and the step size over which the rectangle has to be elevated each time. Finally the user is asked for the height where the script has to start. The total height of the model in Figure 7.16 is four units. The step size is 0.25 units. The distance between the line elements is set to 0.1. With the Loft method, a lofted surface can finally be created between the curves when for all steps the enclosing curves are generated. After removing all underlying model parts, a hollow surface model will remain. The variables which the user has to specify are summarized in Table 7.4.

Variable	User specified value
Rectangle around the model of interest	
Distance between the line elements	
Height of the model	
Step size to elevate the level of concern	
Starting height of the script	

Table 7.4: Variables to specify when running the script of the Rectangle method

Some results of the *Rectangle* method for some more complicated models are discussed in Paragraph 7.3.9. A method to simplify the enclosing curves is discussed in Paragraph 7.3.7. In Appendix F the *Rectangle* method for 3D models is discussed extensively on the basis of the written script.

7.3.4 Rotated Square method

The third method that is developed to find the outer points of a model is a method called *Rotated Square*. The method is based on the *Rectangle* method that is discussed in Paragraph 7.3.3. The method is developed to find the outer points in case of in-built surfaces. The original *Rectangle* and the *Rotating Lines* method do not give proper results for such situations. The following pictures demonstrate the problem.

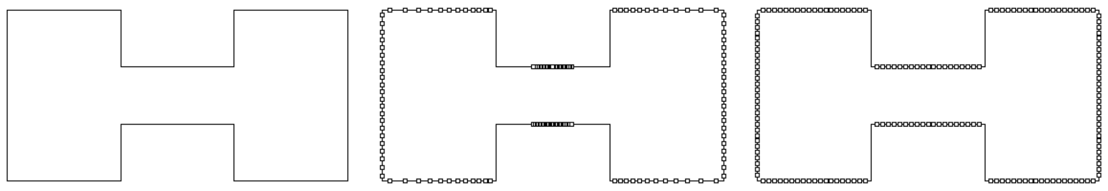


Figure 7.17: Limitations of the Rotating Lines and Rectangle method: outer points at in-built façades are not completely found

If, for example, the floor plan of a model has a shape like the left picture of Figure 7.17. The middle picture gives the result of finding the outer points with the *Rotating Lines* method. The right picture gives the result for the *Rectangle* method. As can be seen, most outer points on the in-built façades are not found. Creating the enclosing polyline through these points will certainly give a divergent curve. This is especially the case for the *Rotating Lines* method.

The developed *Rotated Square* method deals with this problem. In this method, the user is still asked to draw a rectangle around the model of interest. The corner points of this rectangle are now used to generate a square that is rotated over 45 degrees. The square fits the original rectangle exactly. Figure 7.18 demonstrates the principle.

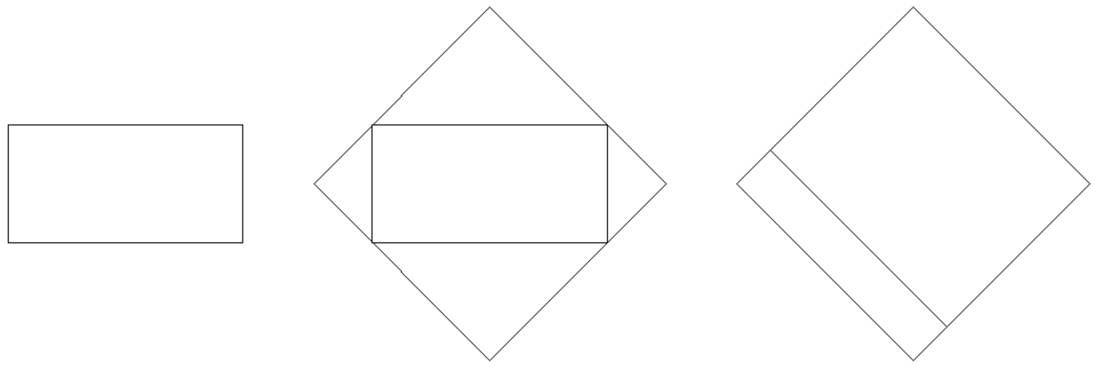


Figure 7.18: The Rotated Square method: the square fits the drawn rectangle exactly

The left picture of Figure 7.18 shows a user-defined rectangle that can be drawn around the model of interest. The corner points of this rectangle are used to define a square that is rotated over 45 degrees and fits the rectangle exactly. The rotated square will now be used as basis for the lines that will search for intersections with the various elements of the model. Starting from the lower left edge of the square, a line will be drawn and moved to the upper right edge of the square over user-specified intervals. For each interval it will be checked if there is an intersection between the line and the various model elements. In case of more than one intersection for a certain location of the line, the script will search for the closest and most far away intersection related to the startpoint of the line. At these locations points will be added to the model. If the line has finally arrived at the upper right edge of the square, the procedure continues with lines moving from the upper left edge of the square to the lower right edge of the square. The method is explained more extensively on the basis of a 2D model on the following page.

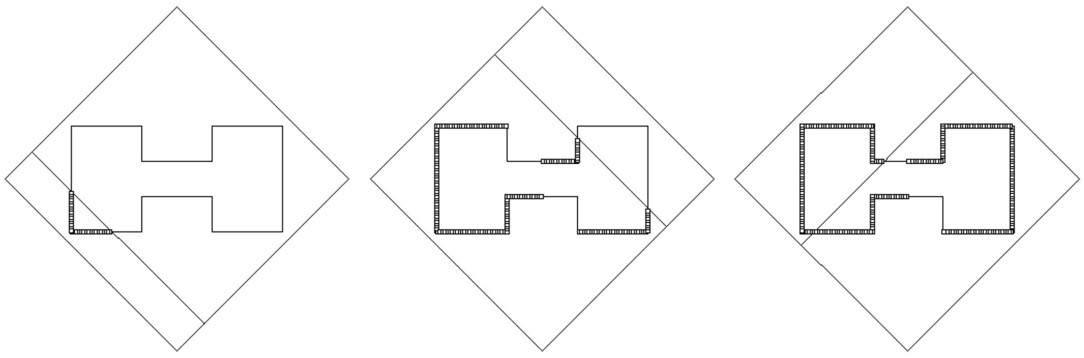


Figure 7.19: Rotated Square method; 2D example

In Figure 7.19 the model with the in-built façades is given again. When the script is loaded, the user is first asked to draw a rectangle around the model. A rotated square is created from the corner points of the original rectangle. Then a line is drawn from the bottom corner of the square to the left corner of the square. The script checks if there are intersections between the line and the model parts and, in case of several intersections, places a point at the nearest and most far away intersection location related to the startpoint of the line. The next line that is drawn is moved diagonally in upwards and right direction over a user-specified distance. When the line is finally arrived at the upper right edge of the square, the process is repeated, but now with lines moving from the upper left edge to the lower right edge of the square. Then all outer points of the model are found.

The next step is to draw the enclosing polyline through the outer points. As in the *Rectangle* method, the outer points do not lie in the right sequence in the array. Before the polyline can be created, the outer points have to be ordered first. The procedure of ordering the array is again based on finding the closest point for a certain outer point. Another consequence of this method is that some outer points are placed twice in the array, because for both directions wherein the lines move, the same outer point can be found. While ordering the array it will also be checked if a certain point is placed twice in the array. The procedure of ordering the array is already discussed in Paragraph 7.3.3.

The procedure of finding the outer points with the *Rotated Square* method can again be extended to the third dimension. The level at which the outer points are determined is raised each time to generate the enclosing curves at different heights. The following pictures demonstrate the method for a simple 3D model.

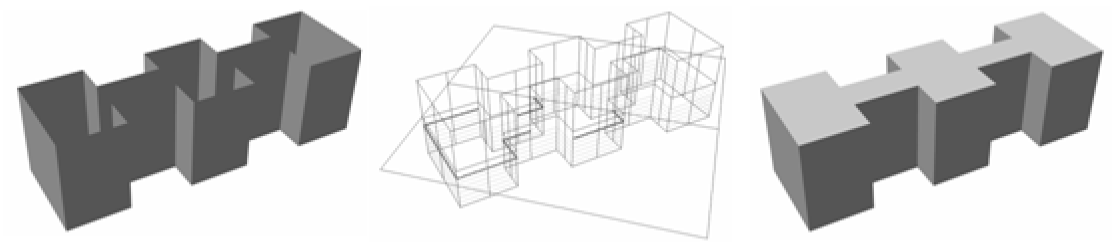


Figure 7.20: Rotated Square method; 3D example

After loading the script, the user is asked to draw a rectangle around the model. The script again uses the corner points of the rectangle to create the rotated square. Then the user is asked for the distance between the line elements over which the lines will be moved. He is also asked for the height of the building and the step size over which the rotated square has to be elevated each time. Finally the user is asked for the height where the script has to start. When for all steps the outer points are found and the curves are generated, a lofted surface can finally be generated with the *Loft* method that is discussed in Paragraph 7.3.8. After removing all underlying model parts, a hollow surface model remains. The variables which the user has to specify are summarized in Table 7.5.

Variable	**User specified value**
Rectangle around the model of interest	
Distance between the line elements	
Height of the model	
Step size to elevate the level of concern	
Starting height of the script	

Table 7.5: Variables to specify when running the script of the Rotated Square method

Results of the *Rotated Square* method for some more complicated models are discussed in Paragraph 7.3.9. A method to simplify the enclosing curves is discussed in Paragraph 7.3.7. In Appendix G the *Rotated Square* method for 3D models is discussed extensively on the basis of the written script.

7.3.5 Integrated method

The discussed methods to find the outer points and generate the enclosing curves seem to work properly for rectangular building models. However, additional tests showed that the methods sometimes fail in finding all outer points of a model, especially for multiple curved models. Figure 7.21 gives an example of a freeform, organic shape. Such organic shapes are very popular in the actual Blob architecture and it is desired that the various tools support such models as well. However, none of the developed methods to generate the enclosing curves succeeded in finding all outer points of the given model. If the *Rotating Lines* method for example gave the outer points for a certain location of the model, it failed in finding the outer points at other locations. Those points were found then with the *Rectangle* method, but this method failed in finding the points at other locations again. Joining the several methods to one integrated method appeared to be the solution. Several tests with various curved models showed that the integrated method succeeded each time in finding all outer points of the various models.

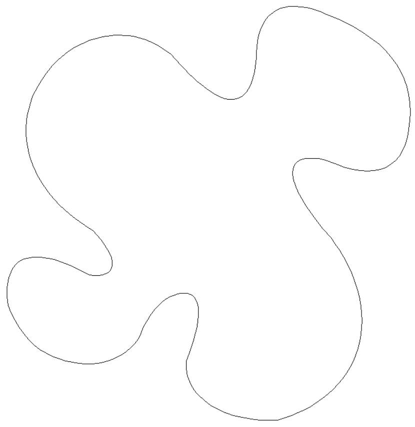


Figure 7.21: Freeform organic shape

A rectangle that has to be drawn around the model forms the base of the integrated method. From this rectangle the rotated square can be constructed and the middle of the rectangle forms the centre point of the rotating lines method. The coordinates of the centre point are calculated from the coordinates of the corners of the drawn rectangle:

$$x = \left(lower\ left\ corner\right)_x + 1/2 \cdot \left(\left(lower\ right\ corner\right)_x - \left(lower\ left\ corner\right)_x\right) \qquad (7.3)$$

$$y = \left(lower\ left\ corner\right)_y + 1/2 \cdot \left(\left(upper\ left\ corner\right)_y - \left(lower\ left\ corner\right)_y\right) \qquad (7.4)$$

After the rectangle is drawn around the building model, the user is asked for the distance between the line elements for the *Rectangle* and *Rotated Square* method and for the amount of parts for the *Rotating Lines* method. From the amount of parts the angle over which the lines are rotated can be calculated. The process of finding all outer points then starts with the *Rectangle* method and is directly followed by the *Rotating Lines* method. The length of the rotating lines is equal to the length of the diagonal between the lower left and upper right corner of the rectangle. In this way the lines are always long enough to reach all model parts. The process to find the outer points is then finished with the *Rotated Square* method. The final step is to draw a polyline through the points that encloses the model. However, as a result of the execution of the three methods in one continuous process, quite a lot of points are found that do not lie in the right sequence. Before the enclosing curve can be generated, the array that contains the outer points has to be ordered first. This procedure is again based on finding the closest point for a certain outer point. On the next page the results of the three separate methods and the integrated method are given for the organic shape of Figure 7.21. Because the script of the integrated method is very long as it is just a sequence of the various separate methods with only some small adaptations, it is not given in this report.

In Figure 7.22 the results of the *Rotating Lines* method (left) and the *Rectangle* method (right) are given for the freeform shape. Both methods are not able to find all outer points on the boundary of the model. Generating the enclosing curve through the points will give a curve that reasonably differs from the original model.

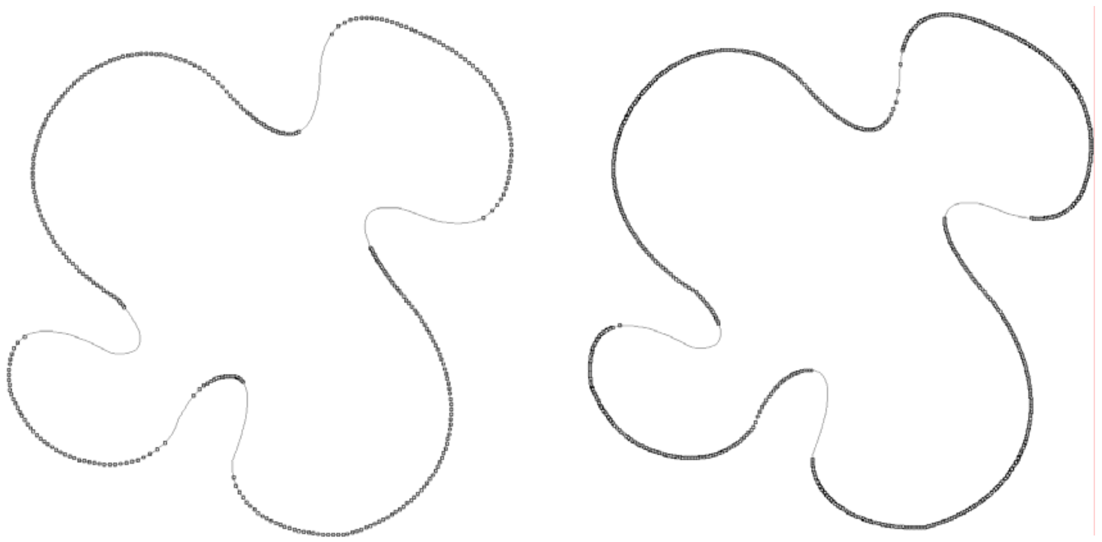


Figure 7.22: Results of the Rotating Lines and the Rectangle method for the given shape

In the left picture of Figure 7.23 the result of the *Rotated Square* method is given for the freeform shape. Comparing the results of the three methods, it can be concluded that each method finds certain points that are not found with the other methods. Joining the methods together gives an integrated method that is able to find all outer points on the model. The result of the integrated method is given in the right picture of Figure 7.23. As all outer points of the model are found, the enclosing curve will perfectly match the original model.

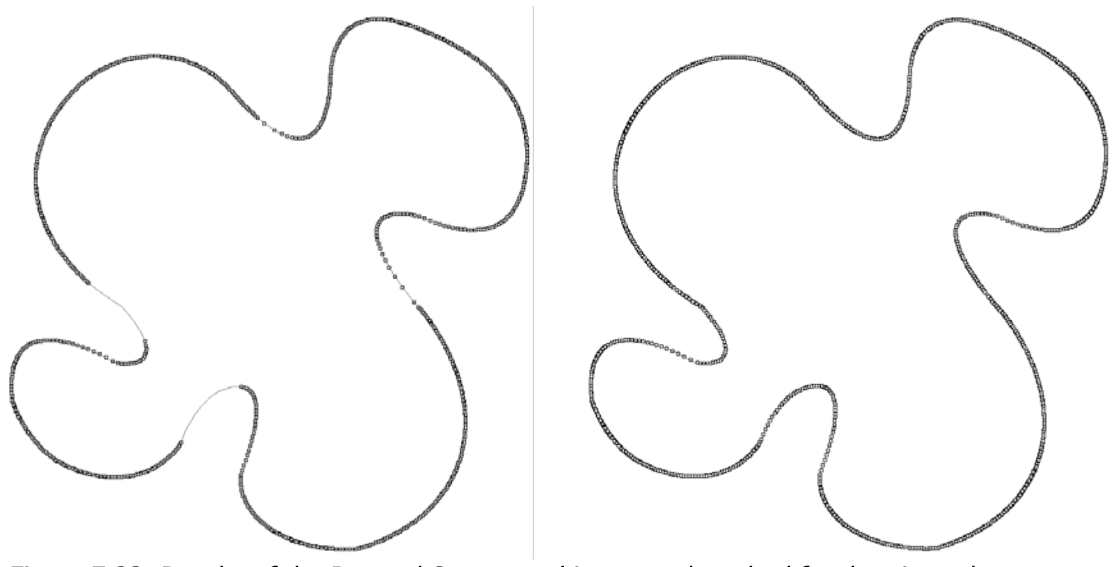


Figure 7.23: Results of the Rotated Square and integrated method for the given shape

Additional tests with other complex, multiple curved models showed that for each model the integrated method was able to find all outer points. When the original methods fail in finding those points on a certain model, the integrated method seems to be a proper alternative. However, as three procedures have to be executed in this method, it will take quite some time before all points are found and the enclosing curves are generated. This is especially the case for extended building models with many elements and a considerable height. A method that quickens the process to some extent would therefore be very valuable.

7.3.6 Meshes

As architects and structural engineers work with various CAD systems nowadays, the purpose of the design tool to simplify the geometry is to support the diverse model types. When traditional models, originating from systems like AutoCAD, are imported in Rhinoceros, the models are still constructed from lines, surfaces and solids. The various developed tools work very well on these models. However, as mentioned in Paragraph 5.5, recent developments in CAD have resulted in technologies like object-oriented modeling and parametric modeling. Models that are created with these technologies are not constructed from lines or surfaces anymore, but of objects with attributes and procedures. The several objects represent the elements from which a building or structure is built of, like walls, floors, beams and columns. The objects have sense for the computer; they are not only points and lines. When such models, made in systems like ArchiCAD or Autodesk Revit, are imported in Rhinoceros, the model contains no lines, surfaces or solids anymore. Because Rhinoceros is not an object-oriented or parametric environment, the objects are converted to meshes. A mesh is basically a collection of vertices and faces connected together to represent a surface.

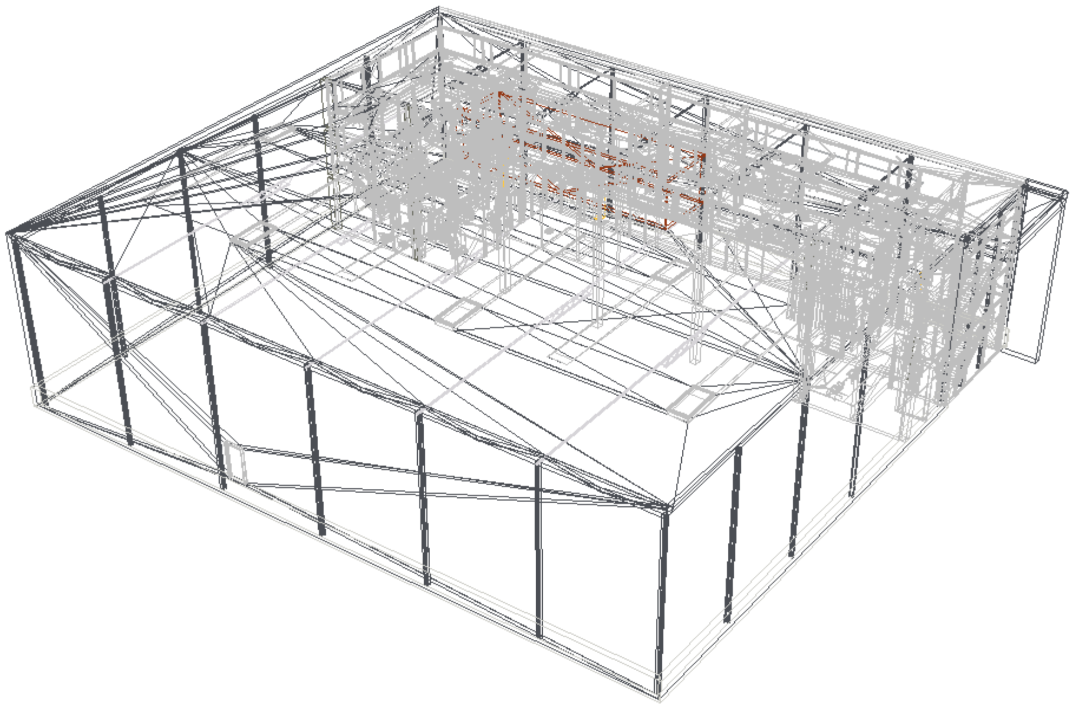


Figure 7.24: Object-oriented models imported in Rhinoceros are constructed from meshes

In Figure 7.24 an example is given of an object-oriented model of an office building that is imported in Rhinoceros. The original model is made in ArchiCAD (Internet, [18]). In Rhinoceros, the meshes are built up from vertices and faces, forming triangles. The problem that arises here is that the current version of Rhinoceros is not able to determine the intersections with meshes. The developed tools to find the outer points of a model appear not to work for these models. However, two methods are developed to deal with meshes and which are able to find the intersection points. The first method converts the meshes to surfaces by deriving the vertices and faces from which the mesh is built. Through these entities a surface can be created, for which it is possible to determine the intersection point. Another method makes use of a new version of Rhinoceros, which is however still in development. With the newest beta edition it seems possible to directly determine the intersection with a mesh. On the following pages these methods are discussed further.

The first method that deals with meshes is based on returning the face vertices of a mesh object. Through these vertices a surface can be created, for which it is possible to determine the intersection point with a line element. When this is done for all meshes of the model, a model is obtained that is constructed from only surfaces. For these surfaces it is possible to determine the outer points and generate the enclosing curves with the developed methods. The principle is explained in Figure 7.25.

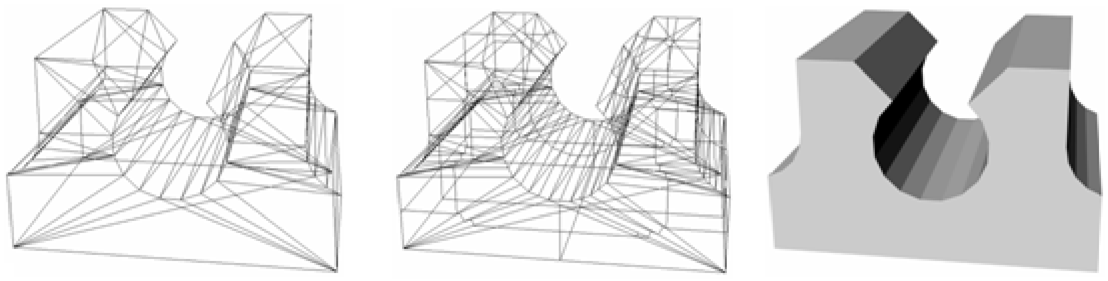
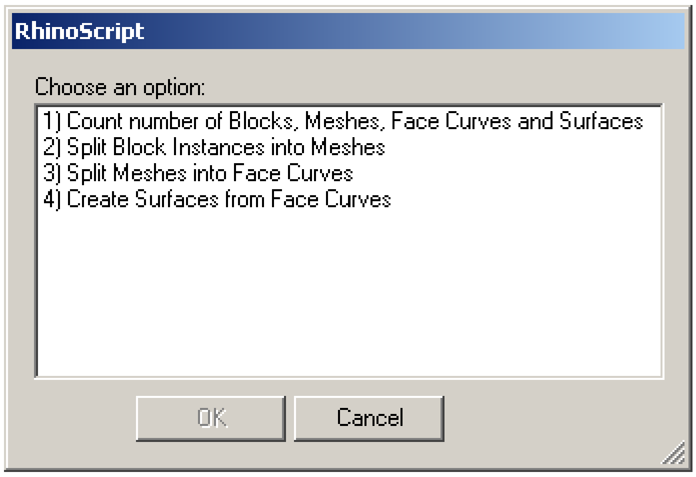


Figure 7.25: Conversion of meshes to surfaces

In the left picture of Figure 7.25 a mesh of a mechanical device is given. The mesh is constructed from vertices forming triangles. The method returns the vertices of the triangles and uses these as boundaries between which surfaces can be created. In the middle picture all mesh triangles of the model are conversed to surfaces. The right picture finally shows a rendered picture of the model.

Because the above discussed method has to be carried out before one of the methods to find the outer points is applied, the method to convert the meshes into surfaces is called *Preprocessing*. The preprocessing process can be seen as a method to prepare the models for the methods to generate the enclosing curves. Next to meshes, a model can also contain block instances. These instances are some kind of superiors of meshes. For block instances it is not possible to determine the intersection between a line element and the instance as well. Exploding a block instance gives the meshes from which the instance is built of. For cases where a model contains block instances, the *Preprocessing* method also gives an option to derive meshes from the instance.

The *Preprocessing* method gives the user four options. First the number of blocks, meshes, face curves and surfaces can be count, to control if the model contains one of these entities. Then the user has the option to split the block instances into meshes. The next option is to split the meshes into face curves and finally surfaces can be created from the face curves. The user interface of the method is given in Figure 7.26.



RhinoScript

Choose an option:

1) Count number of Blocks, Meshes, Face Curves and Surfaces
2) Split Block Instances into Meshes
3) Split Meshes into Face Curves
4) Create Surfaces from Face Curves

Figure 7.26: Preprocessing method

Figure 7.27 demonstrates the process of converting a mesh to surfaces with the *Preprocessing* method and generating the enclosing curves with the *Rotating Lines* method for a chess pawn. Finally a surface is lofted through the curves with the *Loft* method, which is discussed in Paragraph 7.3.8.

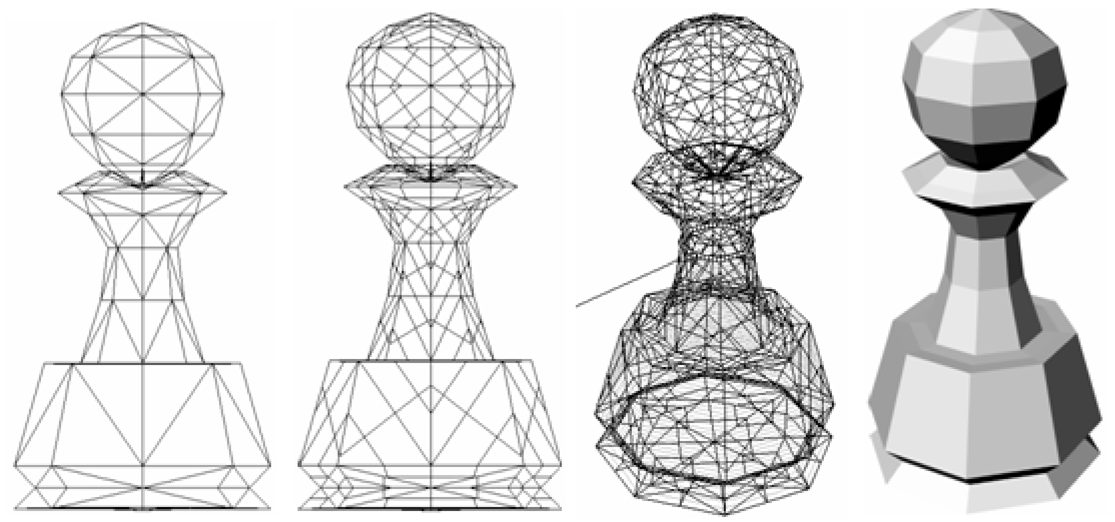


Figure 7.27: The process of wrapping a surface around a mesh object

The first picture of Figure 7.27 gives the mesh model of a pawn. The second picture shows the result of converting the mesh model to a surface model. For an object that is built up from surfaces, the outer points over the height of the model can be found with the *Rotating Lines* method. The third picture shows this procedure. After lofting a surface trough the curves, a hollow surface model of the pawn is generated that replaces the original geometry. The result is shown in the fourth picture. The method also seems to work properly on more complicated models. The results of some test cases are discussed in Paragraph 7.3.9. In Appendix H the *Preprocessing* method is discussed extensively on the basis of the written script.

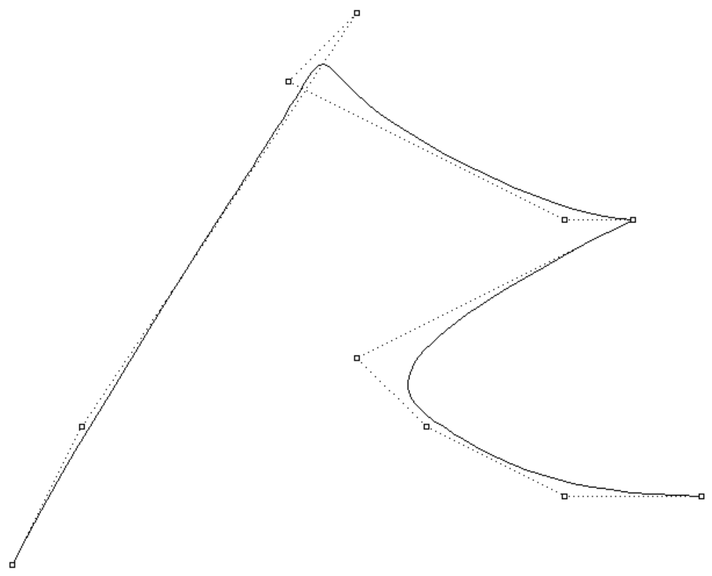
Another way of finding the intersection point with a mesh object is using the newest Beta release of Rhinoceros 4. For all previous scripts and methods, Rhinoceros 3 is used. This is also the version that is installed on the computers of the faculty of Civil Engineering of Delft University of Technology. Rhinoceros 3 is not able to determine the intersection between a line element and a mesh, but Rhinoceros 4 is. In this latest version, some commands and functions are implemented that are not supported by Rhinoceros 3. With these commands it is possible to directly determine the intersections. However, Rhinoceros 4 is still in development and no final versions are available yet. Although Beta releases are free to download with the license key of Rhinoceros 3, it is not allowed to install Beta software on the faculty's computers. Nevertheless, with some help of David Rutten (Internet, [3]) a method is developed for Rhinoceros 4 to find the intersection with a mesh. This method is then implemented in the *Rotating Lines*, *Rectangle* and *Rotated Square* methods, leading to two variants of each method: one variant to determine the outer points for a model that is built up of lines, surfaces and solids, and another variant to determine the outer points for a model that is built up of meshes. Both variants give the same results, but with the meshes variant for Rhinoceros 4 the whole *Preprocessing* process can be skipped. This saves time, especially when the model contains hundreds of meshes. The scripts were tested on an ouster system where Rhinoceros 4 was installed and they all seem to work properly. Appendix I gives the implementation of the method to find intersections with a mesh, in the *Rotating Lines* method. The implementation in the *Rectangle* and *Rotated Square* method is also performed, but as the procedure is the same, the implementation for these methods is not given in this report.

7.3.7 Curve simplification

Applying one of the three methods to determine the outer points of a model give the enclosing curves of the model for several height steps. The smaller the distance between the line elements is chosen, the more outer points are found. When the model is highly detailed, the enclosing curves will fluctuate considerably. To simplify the curves and get rid of the little details, special fitting algorithms can be used. In this paragraph the NURBS fitting technique is discussed to simplify the enclosing curves. However, it must be said that it is not always necessary to simplify the enclosing curves. When a building model has smooth façades for example without much detail, the enclosing curves perhaps do not have to be simplified. The user must choose for himself if he wants to simplify the curves or not.

The NURBS fitting technique will use the outer points of the model where the enclosing curves are constructed from, as control points to fit a simplified curve through it. By assigning parameter values to the control points, a curve can be created that fits the points at the given parameters. Depending on the parameter values, the curve can follow the control points exactly or more approximately. As the original enclosing curves are constructed from the outer points of the model, they automatically form the control points of the enclosing curves. The enclosing curves go straight through the control points; no parameter values are defined to fit the curve through the points. Here lies the difference with the NURBS fitting technique as the fitting curves do not go exactly through the outer points. The method uses the outer points as control points to fit a smooth curve through the points dependent on the given parameter values.

NURBS fitting
NURBS curves are curves that are parametrically represented. The curve is defined by its degree, the control points, knots and weight values. The degree of the curve can be any positive whole number, but it is usually 1, 2 or 3. If the degree is 1, the curve is linear and it will consist of straight sections. Quadratic curves have degree 2 and cubic curves have degree 3. For fitting a smooth curve through the control points, degree 3 is recommended. Another factor that defines a NURBS curve is the knots. The knots determine the smoothness of the curve. The amount of knots on a curve is the degree of the curve plus the number of control points minus 1. For a NURBS curve of degree 3 and with 11 control points, the list of knots can be for example: 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 8, 8. The knot values in the list must be of increasing order. The number of times a knot value is duplicated in the list is called the knot's multiplicity. In this example the knot values 0 and 8 have multiplicity 3 and the knot values 1 to 7 have multiplicity 1. If a knot value is duplicated by the degree of the curve it is said to be a full-multiplicity knot. With a degree 3 curve, the knot values 0 and 8 have full-multiplicity for this example. A knot value that appears only once is called a simple knot. Duplicate knot values in the middle of the knot list make a NURBS curve less smooth. A full-multiplicity knot in the middle of the knot list means that the curve is bent into a sharp kink there. When the knot list starts and ends with a full-multiplicity knot, the curve starts exactly in the first control point and ends in the last control point. If not, the curve can start or end at a different location to construct the best fit. So, a knot list that starts and ends with a full-multiplicity knot and has simple knots between them, results in a smooth curve that goes exactly through the first and last control point and fits the other control points. The last factor that defines a NURBS curve is the weight of the control points. The weight values determine the influence of the control points and can be seen as variables that indicate the accuracy of the fitting process. In general the first and last weight should be set to 1. If the weight value of the control points is larger than 1, the points are closer approximated by the curve. If the weight values are smaller than 1, the points are less approximated. On the next page some examples are given of the construction of NURBS curves.



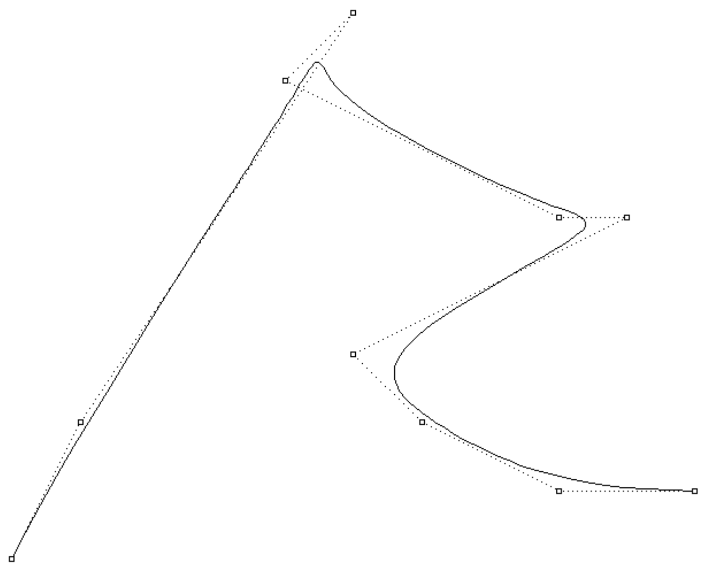
Coordinates and weight factors of the control points:

	X	Y	Z	W
1	0	0	0	1.0
2	1	2	0	1.0
3	5	8	0	1.0
4	4	7	0	1.0
5	8	5	0	1.0
6	9	5	0	1.0
7	5	3	0	1.0
8	6	2	0	1.0
9	8	1	0	1.0
10	10	1	0	1.0

Knot vector:
0, 0, 0, 1, 2, 3, 3, 3, 4, 5, 5, 5

Figure 7.28: Kinked NURBS curve due to full-multiplicity at the 6th control point

In Figure 7.28 a NURBS curve with its control points is given. The coordinates of the control points, together with the weight factors of the points, are given in the table next to the figure. Also the knot vector for this curve is given. The knot list has a full-multiplicity knot at the beginning and end, resulting in a curve that starts and ends exactly in the first and last control point. The knot list also contains a full-multiplicity knot in the middle, what results in the sharp kink in the middle of the NURBS curve. The curve goes exactly through the control point there. In Figure 7.29 the same curve is given again, only the knot vector is changed. The list only has full-multiplicity knots at the beginning and end of the list; all other knot values are single knots. This results in a smooth curve that goes exactly through the start and endpoint and fits all other control points.



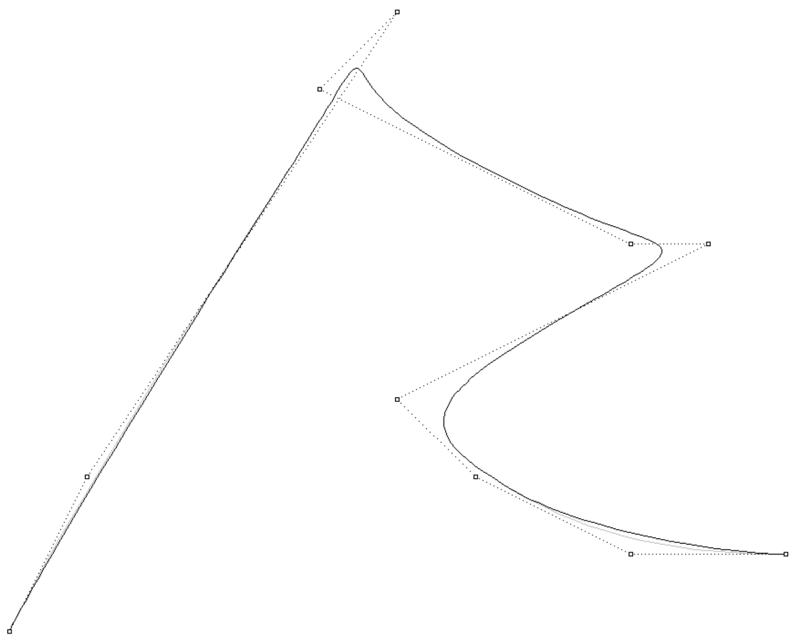
Coordinates and weight factors of the control points:

	X	Y	Z	W
1	0	0	0	1.0
2	1	2	0	1.0
3	5	8	0	1.0
4	4	7	0	1.0
5	8	5	0	1.0
6	9	5	0	1.0
7	5	3	0	1.0
8	6	2	0	1.0
9	8	1	0	1.0
10	10	1	0	1.0

Knot vector:
0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 7, 7

Figure 7.29: Smooth NURBS curve

Figure 7.30 shows two curves with various weight factors for the control points.



Weight factors of the control points:

	W_1	W_2
1	1.0	1.0
2	1.0	0.3
3	1.0	0.3
4	1.0	0.3
5	1.0	0.3
6	1.0	0.3
7	1.0	0.3
8	1.0	0.3
9	1.0	0.3
10	1.0	1.0

Figure 7.30: Weight factors determine the influence of the control points

The influence of the weight factors is small for the curve in the given example. For the black NURBS curve in Figure 7.30 the weight factors for the control points are set to 0.3. For the original grey curve the weight factors were set to 1.0. The lower weight values result in a smaller approximation of the control points. Only at the end of the curve a small difference can be seen between the two NURBS curves. However, as follows from the given figures, the NURBS curve is very smooth in relation to its original curve that is drawn directly through the control points. Simplifying the enclosing curves by fitting a NURBS curve using its control points looks very promising.

The script that is developed for the *NURBS fitting* method returns the control points of the enclosing curves. The first and last control points get a full-multiplicity knot; the rest of the knot list is filled with single knot values. This results in a NURBS curve that starts and ends exactly in the first and last control point. The user must specify a certain weight value that is set to all control points, except the first and last one. The weight value for these control points is set to 1 by default. Then the script is able to create a NURBS curve from the derived control points. The process is repeated for all enclosing curves in the model, after which the original curves are deleted. The result is a model that contains only NURBS curves. With the *Loft* method that is explained Paragraph 7.3.8, finally a surface can be generated through the curves. The script of the NURBS fitting method is given in Appendix J.

7.3.8 Lofting

After the enclosing curves for several height levels are generated and eventually simplified, a lofted surface can be created through the curves to form a closed surface model. The lofting process has become a separate procedure and is not directly implemented in the methods to find the outer points and generate the enclosing curves. By separating these procedures, the user has the possibility to adapt or eventually remove some enclosing curves that do not fully satisfy. This could be desired if the methods to find the outer points do not give a satisfactory result for a certain height level. For the models that are tested and discussed in Paragraph 7.3.9 this was not the case and the methods returned accurate curves. However, if for a certain model the methods somehow do not give a satisfying result, the user just has the opportunity to intervene. By returning the control points of a curve it can be adapted manually to the desired result, or it can be removed from the model.

After the enclosing curves are checked by the user, the script to loft a surface through the curves can be loaded. The script searches for the lowest curve and from there it creates a straight surface to the curve that lies one level higher. From there on, the next surface is lofted to the curve that lies one level higher again. The procedure continues until through all curves the surfaces are lofted. In vertical direction the model is now closed by the surfaces. However, the bottom and top of the model are still open. To close the model completely, a planar surface is generated through the lowest and topmost enclosing curve. The lofted surfaces and the planar surfaces at the bottom and topside of the model are finally joined together, to form one single surface model. The procedure is explained with the following pictures on the basis of a 3D example.

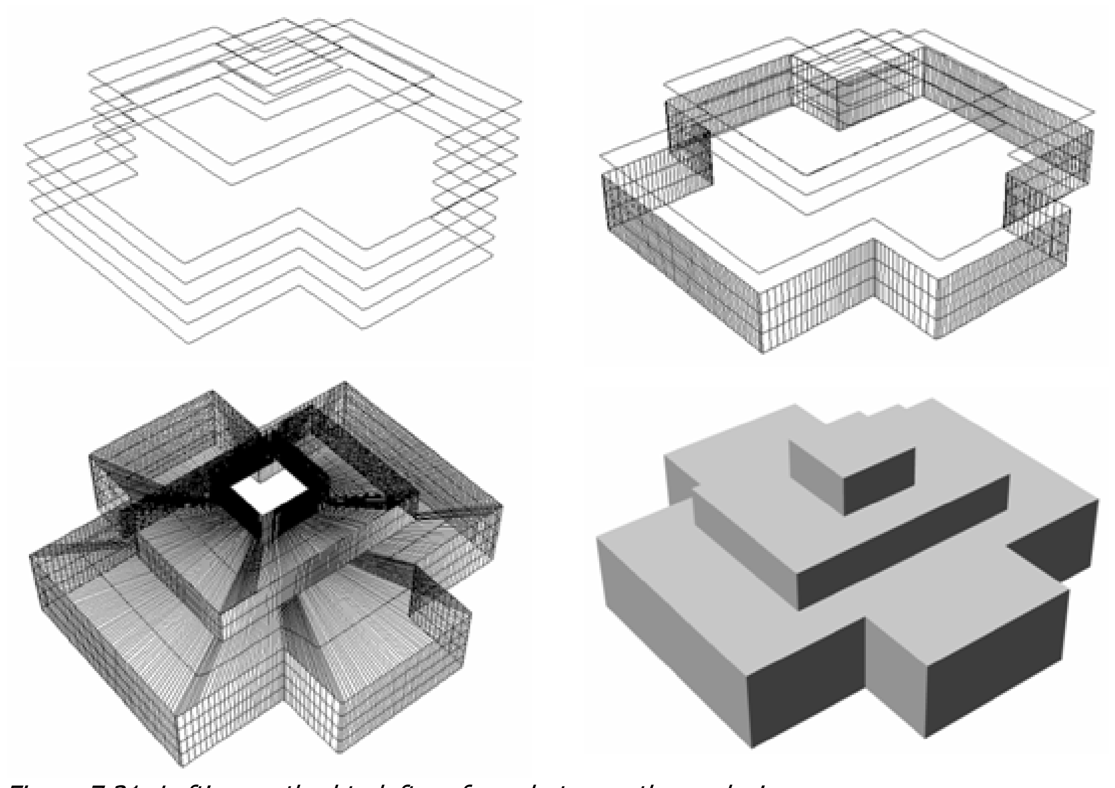


Figure 7.31: Lofting method to loft surfaces between the enclosing curves

In Appendix K the lofting method is discussed extensively on the basis of the written script.

7.3.9 Results

After all methods are gone through for a certain building model, a lofted surface that has wrapped the model is left. All inner and outer geometry of the original building model are removed, what remains is a closed, hollow surface model. This simplified model of the building of interest can manually be placed in a 3D model of the research area then. In this paragraph some results are discussed of the various methods to create a wrapped surface around a building model. The several building models originate from various CAD systems, like SketchUp and ArchiCAD.

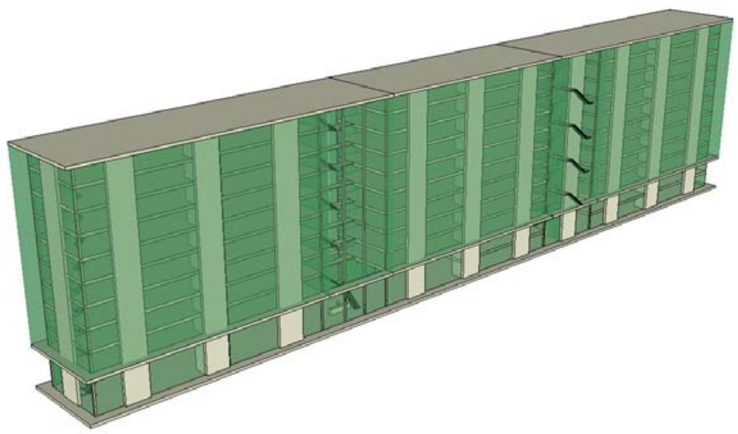
Dream and Realization (SketchUp)
The first building model that is used to test the various methods is obtained from students of the faculty of Architecture of Delft University of Technology. The model is the result of a design exercise called *Dream and Realization*. It is a design of an office building that consists of three separate blocks that are connected through atria. The model was generated in SketchUp and exported to Rhinoceros in the dwg file format. When opened in Rhinoceros, the model contained only block instances. Because the intersection between the line elements and a block instance can not be determined in Rhinoceros, the block instances were first converted to meshes with the *Preprocessing* tool. The same tool is then used to return the face curves of the meshes and create surfaces through these face curves. From now on it was possible to determine the intersection between the line elements and the model parts. As the building had a nice, rectangular shape, the *Rectangle* method was used to determine the outer points and generate the enclosing curves over the full height of the building. No curve simplification was applied, as the enclosing curves were quite smooth. The facades of the building model did not contain small details.

The variables that were specified when running the *Rectangle* method are summarized in Table 7.6. As a result of the conversion from meshes to triangular surfaces, the amount of elements that the script had to go trough to determine the outer points was large. Where the original model contained about 1.200 elements, the conversed model contained 18.599 elements. As for each line of the method all elements have to be checked if it has an intersection with the line, it takes quite some time before all outer points are found. To quicken the process, a larger distance between the line elements or a larger step size to elevate the level of concern can be chosen. Less line elements have to be gone through then. However, this is only permitted when the model is relatively straight and when it does not contain many details. Another method to quicken the process is to manually delete all inner geometry before the script is executed. Most architects work with several layers to distinguish the various building elements. When for example all objects in the various furniture or internal structure layers are removed, the model contains far less elements. As the script has to go through fewer objects then, the process can be quickened considerably. Finally it is also possible to use the *Rectangle* method variant that is adapted for Rhinoceros 4. With this version it is directly possible to determine the intersection between a line element and a mesh. It is not needed to convert the meshes to surfaces what results in less elements. This saves time twice: the process of converting the meshes can be skipped and the script has to check fewer elements.

Variable	User specified value
Rectangle around the model of interest	-
Distance between the line elements	1.000 mm
Height of the model	42.000 mm
Step size to elevate the level of concern	500 mm
Starting height of the script	0 mm

Table 7.6: Specified variables for the Rectangle method

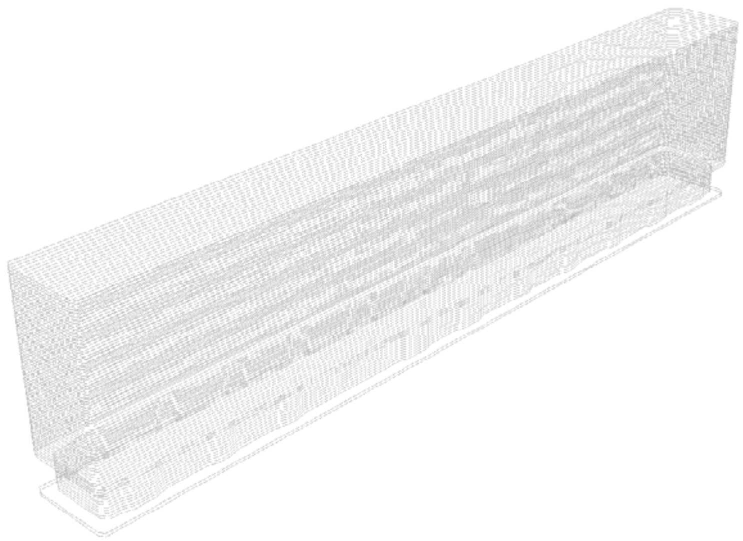
On the next page the results of the *Rectangle* method are given for the office building.



Original building model:

- 18.599 elements
- Model contains internal and external geometry

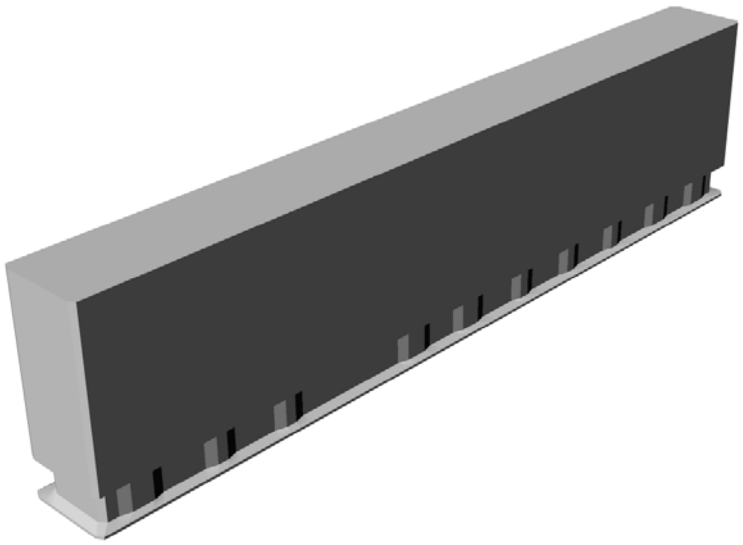
Figure 7.32: Design of the office building



Enclosing curves model:

- 85 curves
- Distance between curves = 500 mm
- Original geometry is removed

Figure 7.33: Enclosing curves over the full height of the building



Lofted model:

- 1 single polysurface
- Model is empty
- Surface model replaces internal and external geometry

Figure 7.34: Result of the methods: a wrapped, hollow surface model of the office building

The figures on the previous page show the original building model, a model containing the enclosing curves and finally the wrapped surface model of the office building. Before the scripts were executed, some internal geometry was deleted to quicken the process. Especially the stairs with all its steps contained a lot of elements. After the *Rectangle* method was executed for the building model, 85 enclosing curves were generated. Through these curves a surface is lofted that wraps the original model. The result that is shown in Figure 7.34 matches the original building model accurately. If there were any gaps in the model, they are now closed. All inner and outer geometry is removed; what remains is a single, hollow surface model that consists of one part. The next step is to eventually place the model in a 3D model of the environment and export it to the CFD software.

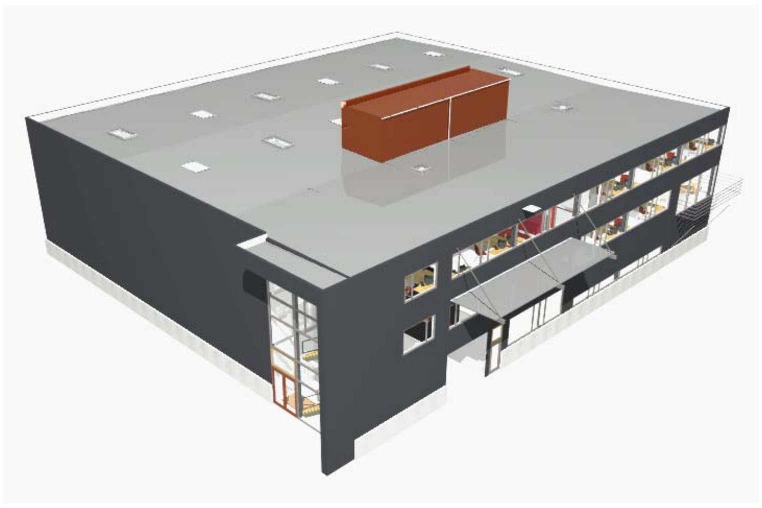
Office building (ArchiCAD)

The next building model that is used to test the various methods is an office building that is drawn with ArchiCAD. The model was placed as an example on the ArchiCAD installation CD. As ArchiCAD is an object-oriented CAD system, the model elements are converted to meshes when imported in Rhinoceros. As the building model had a rectangular shape, the *Rectangle* method was used again to find the outer points of the model. In stead of converting the meshes to triangular surfaces, the Rhinoceros 4 variant of the *Rectangle* method was used. With this variant it is possible to directly determine the intersection between the line elements and the meshes, so no preprocessing was needed. The building model contained office rooms at the front side of the model. The offices were fully furnished with desks, chairs, computers and carpets. The backside of the model contained a large hall with some structural elements. To quicken the process, all furniture and internal structures were manually removed from the model before the scripts were executed.

The variables that were specified when running the script are summarized in Table 7.7. As the façades of the building model were straight without much detail, the chosen distance between the line elements was relatively large. The step size was chosen smaller, as the office building contained some canopies with a limited thickness that had to be modeled. In the figures on the following page the result of the *Rectangle* and *Loft* methods are given for the office building. The surface model looks a little strange because the facades and roof do not contain any texture, but it reflects the form of the original model accurately.

Variable	User specified value
Rectangle around the model of interest	-
Distance between the line elements	1.000 mm
Height of the model	13.000 mm
Step size to elevate the level of concern	500 mm
Starting height of the script	0 mm

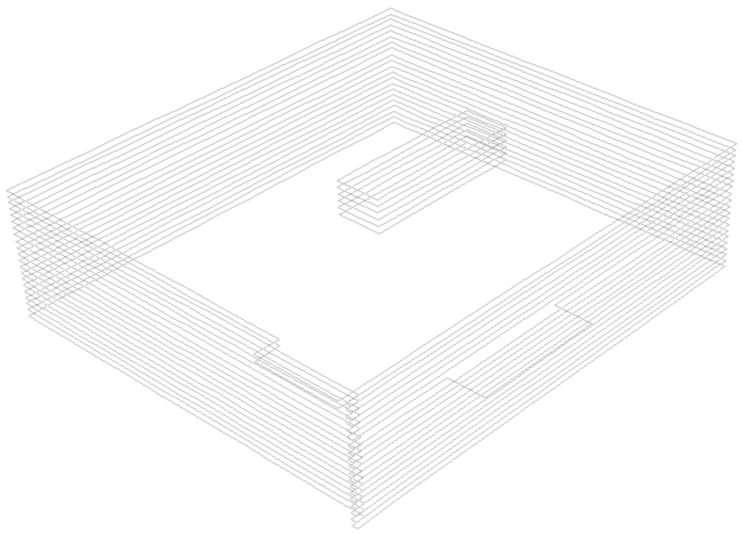
Table 7.7: Specified variables for the Rectangle method



Original building model:

- 2.059 meshes
  (= 11.860 surfaces)
- Model contains internal and external geometry

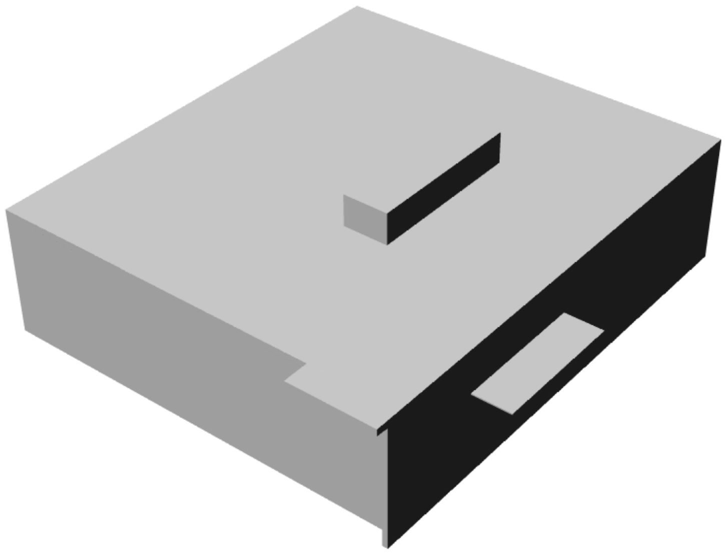
Figure 7.35: Design of the office building



Enclosing curves model:

- 27 curves
- Distance between curves = 500 mm
- Original geometry is removed

Figure 7.36: Enclosing curves over the full height of the building



Lofted model:

- 1 single polysurface
- Model is empty
- Surface model replaces internal and external geometry

Figure 7.37: Result of the methods: a wrapped, hollow surface model of the office building

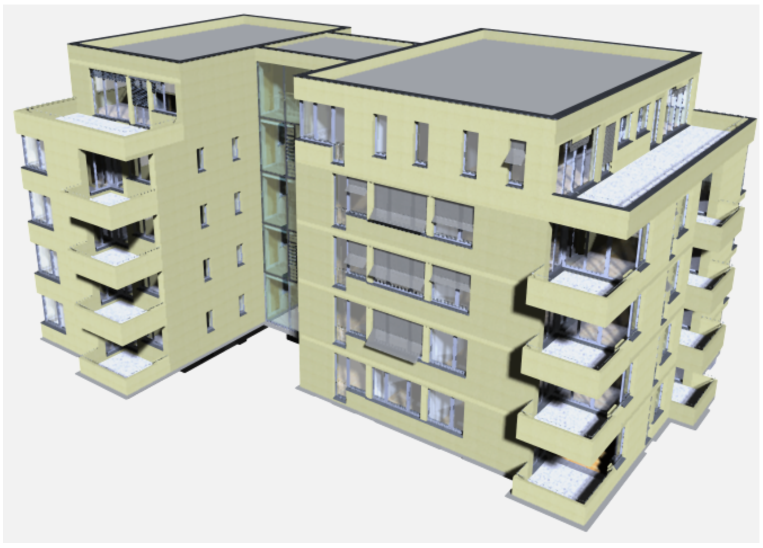
Apartment complex (ArchiCAD)

Another building model that is used to test the various methods is an apartment complex that is again modeled in ArchiCAD. Because the building is modeled in an object-oriented system, the various elements were again converted to meshes when imported in Rhinoceros. As the building model contains in-built façades, like the glass staircase in the middle of the building, the *Rotated Square* method is used to find the outer points of the model. To quicken the process, the Rhinoceros 4 variant of the script is used that directly finds the intersection between a line element and a mesh. Some furniture, inner structures and window screens were deleted to decrease the amount of elements. This also shortened the process. The variables that were specified when running the script are summarized in Table 7.8.

Variable	**User specified value**
Rectangle around the model of interest	-
Distance between the line elements	1.000 mm
Height of the model	19.000 mm
Step size to elevate the level of concern	500 mm
Starting height of the script	0 mm

Table 7.8: Specified variables for the Rotated Square method

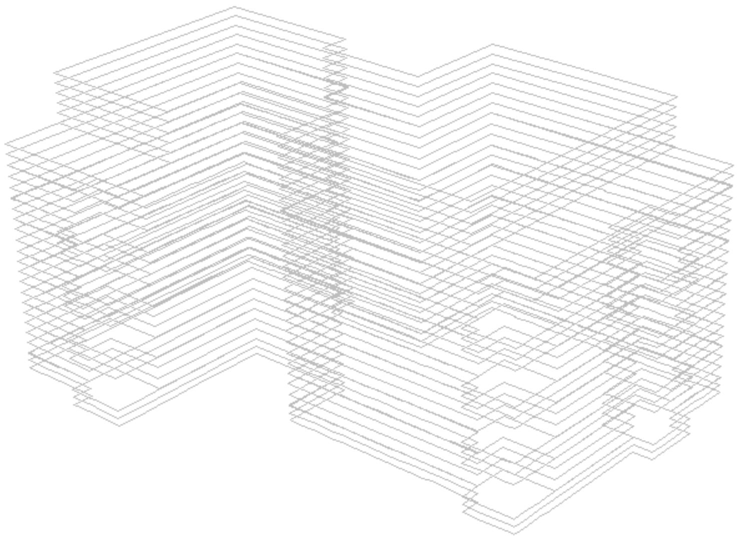
The figures on the next page give the result of the *Rotated Square* method and the *Loft* method to generate a lofted surface around the building model. Again the shape of the original building model is reflected accurately. Only the balconies are not modeled precisely. Due to the operating procedure of the methods, horizontal surfaces that are placed behind standing surfaces are not found. Because the outer points are determined on a horizontal plane that is leveled each time, the border of the balconies prevent that the floors behind are modeled. The balconies in the surface model are just represented as box volumes. However, for the determination of the global wind loads, this would probably be not a major bottleneck.



Original building model:

- 6.331 meshes
  (= 37.765 surfaces)
- Model contains internal and external geometry

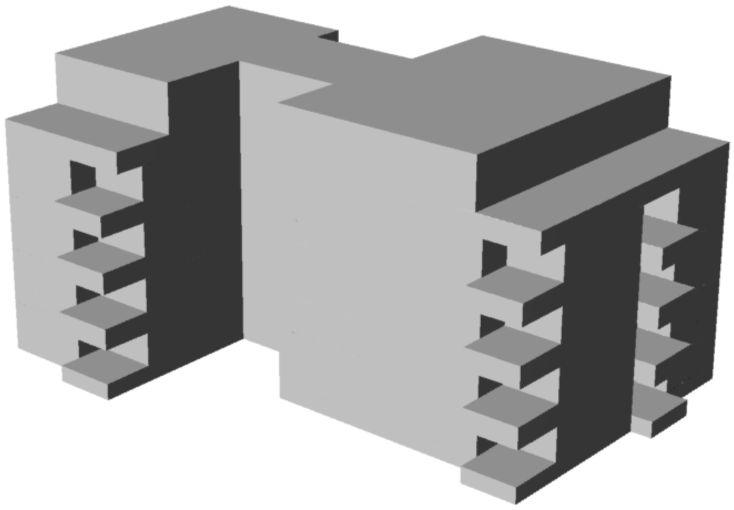
Figure 7.38: Design of the apartment complex



Enclosing curves model:

- 39 curves
- Distance between curves = 500 mm
- Original geometry is removed

Figure 7.39: Enclosing curves over the full height of the building



Lofted model:

- 1 single polysurface
- Model is empty
- Surface model replaces internal and external geometry

Figure 7.40: Result of the methods: a wrapped surface model of the apartment building

7.3.10 Restrictions

Building parts of different height

The results of the tests in the previous paragraph show that the developed methods work and give good results for the given building models. Especially for rectangular and circular models the three methods work very well. If a building model has multiple curved façades the original methods might fail, but several tests with the integrated method showed that this method is capable of finding the outer points for such models. Although it takes some time then to generate the enclosing curves, accurate results are obtained. However, there still are some restrictions on the various methods to find the outer points and to create the enclosing curves. The principle of the methods is to create a curve from the first outer point that is found, through all next outer points, to the last outer point. As the last outer point is the same as the first outer point, the result is a closed curve. Problems arise when a building model consists of various parts with different heights. In Figure 7.41 an example is given of a Blob model that consists of two higher parts at the outside and a lower part in the middle of the model. When the level of concern of the various methods is raised above the height of the middle part, only points at the outer model parts are found. As one single enclosing curve is generated through these points, the curve also passes the lower middle part of the model.

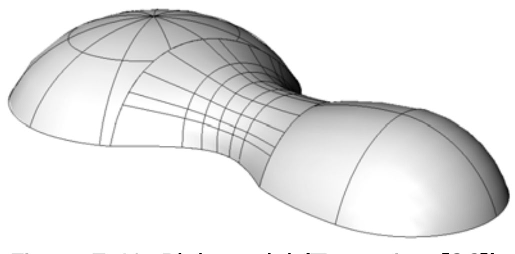


Figure 7.41: Blob model (Toussaint, [26])

In Figure 7.42 this principle is explained further. The figure gives a side view of the model with the enclosing curves that are generated with the *Rotated Square* method. At a certain level the left and right model parts are just separated by air as there is no construction in between. At that level only the outer points at the outside model parts are found. As for each level only one curve is created through all points, the curve crosses the middle area.

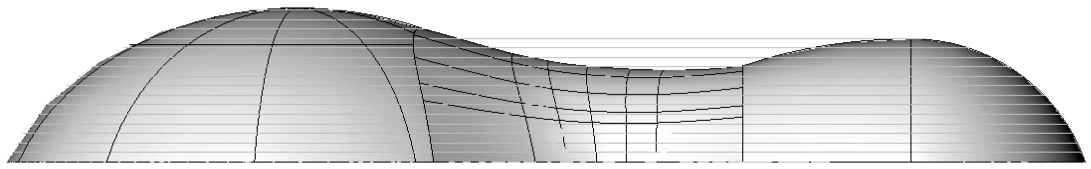


Figure 7.42: Side view of the model with the generated enclosing curves

In Figure 7.43 a 3D view of the enclosing curves of the Blob model is given. When a surface is lofted through the curves, the air in the middle would be modeled as a building part, which is off course not desired. Ultimately the method should recognize building parts of different heights. In stead of creating one single enclosing curve, it should create multiple enclosing curves at a certain height level in case of several building parts. Joining the separate lofted building parts would give a single building model again. At the moment the various methods are only able to find the outer points in a horizontal plane and the level of concern is raised each time to create the curves at several heights. The methods should be adapted to find the outer points on a vertical plane as well. In addition to a horizontal plane that is raised each time, a vertical plane that moves in a horizontal direction along the model could be used to find some other outer points. This can help to find the various building parts of different height.

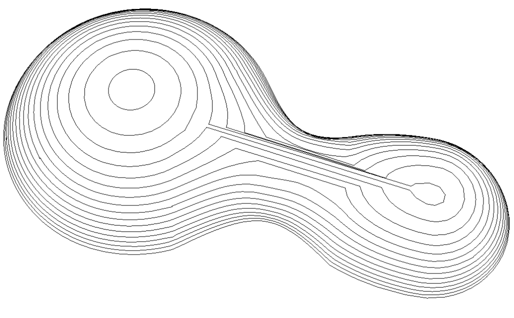


Figure 7.43: Enclosing curves of the model

When the methods to find the outer points would be adapted to use a vertical plane as well that moves in horizontal direction, it is not recommended to create enclosing curves through the points. It will be difficult to automatically create a surface through the various horizontal and vertical curves then. However, using both a horizontal and a vertical plane to find the outer points results in a point cloud on the boundaries of the model. For a point cloud it is possible to directly construct a surface through the points that will enclose the model.

Processing time
Another restriction of the various methods to simplify the building of interest is the time required to find the outer points of the model. Especially when the *Integrated* method is used it takes quite some time before all enclosing curves are generated, as the *Rotating Lines* method, the *Rectangle* method and the *Rotated Square* method are all executed in a continuous process. The total time required to wrap the building model with a surface can increase up to several hours then. The Rhinoceros Visual Basic language in which the various methods are scripted appears to be the reason of the moderate processing speed. The script is gone through step by step and only one task can be performed at the same time. Although the Visual Basic language is reasonably simple to learn and very suitable for those who are not that familiar with programming, the methodology is a disadvantage of the language. To quicken the process of the various methods, the scripts should be translated to more advanced programming languages. Rhinoceros supports DotNET plug-ins and the C++ programming language as well, which are able to script the methods with more advanced techniques. Using these advanced languages could quicken the procedures considerably. However, the Rhino Visual Basic language uses special commands that are able to directly determine the intersection between lines, surfaces and solids. DotNET plug-ins and the C++ language do not support these commands and the programmer should construct these algorithms by himself then, which could be difficult.

Nevertheless, as a first study to the development of a tool that simplifies a building model, the developed methods seem to work properly for most cases. From the results of the tested models it can be concluded that the methods have potential. To optimize the process, further research is required to translate the scripts of the methods to more advanced programming languages. As Rhinoceros also supports DotNET plug-ins and the C++ language, it remains a very suitable environment in which the tools could be implemented.

7.3.11 Suggestions for additional methods

Only for building models that consist of several parts which are separated by air, the developed methods seem not capable of wrapping a fitting surface around the model. In this paragraph two suggestions for additional methods to simplify such models are discussed.

Shrinking sphere
Simplification of a certain building model by wrapping a hollow surface around the model could be realized by using a shrinking sphere. Suppose a user-defined sphere that encloses the building model completely. When the sphere is able to deform due to a certain elasticity and if it is prohibited to perforate the sphere, the model could be wrapped by shrinking the sphere. In fact some kind of vacuum will be generated then that creates a surface around the model of interest from the sphere. If one is able to script this procedure and to ascribe an elasticity to the sphere, it could be a very suitable method to enclose all kinds of models.

Laser scanning approach

Another option to wrap a surface around a building model is a method that works similar to laser scanning equipment. Figure 7.44 gives the procedure of scanning an object using laser devices. If a certain script is able to imitate this equipment, a certain building model could be scanned to find the outer points of the model. The source of the scanner should be able then to circle around the model in all possible directions. A single line could be generated then from the scanner to the model and for each position of the scanner, the intersection with the boundary of the model could be determined. A single point can be added to the model at that location then. Repeating this procedure for all possible positions of the scanner should result in a point cloud on the boundaries of the model. With a technique called reconstruction the point cloud could be converted into a 3D surface model then. Reconstruction involves finding and connecting adjacent points in order to create a continuous surface. Rhinoceros should be very suitable for the development of such a method.

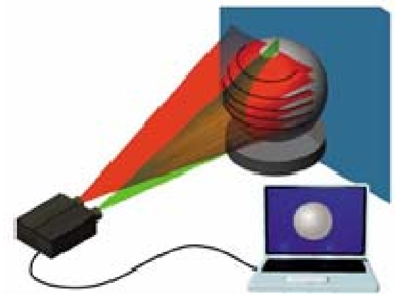


Figure 7.44: Laser scanning

If other methods to simplify a building model are developed in future research, the user of the Virtual Wind Tunnel will have a variety of design tools at his disposal. The user must decide for himself then which available tool is the most appropriate for a certain case.

7.3.12 Evaluation

In the previous paragraphs various methods of the third design tool that simplifies the building of interest are discussed. With these methods it is possible to wrap a surface around the model that replaces the external and internal geometry of the original building design. The total process consists of creating the enclosing curves for several height levels with the *Rotating Lines*, *Rectangle*, *Rotated Square* or *Integrated* method, eventually simplifying the enclosing curves with the *NURBS fitting* method and finally lofting a surface through the curves. The original geometry is deleted automatically then. The result of the various methods is a single, hollow surface model that can be used for CFD analysis. Calculations to determine the wind loads can be performed for only the surface model itself, or it can be placed in a 3D model of the environment that can be generated with the first and second design tools. In this way the influence of the surrounding buildings can be taken into account in the calculation of the wind loads. The surface model of the building of interest has to be placed manually in the 3D model of the environment then.

To find the outer points and generate the enclosing curve for a certain model, the user must choose the most appropriate method. In general it can be said that the *Rotating Lines* method is most suitable for building models with a more or less circular floor plan. When the center location of the model is taken as the location from where the rotating lines are constructed, it must be possible to accurately generate the enclosing curves at various height levels. For buildings with a rectangular or extended ground plan the *Rectangle* method is the appropriate method to find the outer points and generate the enclosing curves. The *Rotated Square* method, which is based on the *Rectangle* method, is the most suitable method for buildings with in-built façades. For building models where the previous methods fail, the *Integrated* method that combines all other methods should be used in order to create the enclosing curves.

For each method to find the outer points and generate the enclosing curves, two variants are developed. The first variant determines the intersection between the rotating or moving line element of the methods and the line elements, surfaces and solids of the model. However, when a model is converted to meshes when it is imported in Rhinoceros, it is not possible to directly find the outer points of the model with the first variant. The meshes can be converted to triangular surfaces then with the *Preprocessing* method, after which it is possible to use the first variant of the methods. However, this is an extra step in the process and takes some time. Another possibility to determine the intersection between the line elements and a mesh is using the second variant of the methods. For this second variant, Rhinoceros Beta 4 is required as the script uses some commands and methods that are only supported by the newest version. With these commands it is possible to directly determine the intersection with a mesh, without the need of converting it to its triangular surfaces. However, as Rhinoceros 4 is still in development, no final releases are available yet. It is therefore not allowed to use it on the faculty of Civil Engineering's computers, so only on outer computer systems these scripts can be used.

The total procedure of generating the enclosing curves, simplifying the curves with the *NURBS fitting* method and lofting a surface through the curves is performed several times for various building models. In the object-oriented models, the designer is free to present moveable objects like doors and windows opened or closed. If front doors and windows are still open when executing the methods to find the outer points, no point will be found at the locations of these doors and windows. In stead, the most outer point for that cross-section will be found on a wall or object inside the model. Therefore it is important to close all doors and windows of the model before the methods are applied.

Complex models contain lots of elements. Especially when the meshes are converted to surfaces, the amount of elements quickly increases. The amount of elements in the model has a significant influence on the time required to find all outer points. For each line of the method, all model elements have to be checked if there is an intersection with the line. Reducing the amount of elements will certainly decrease the amount of time required. Before the methods to find the outer points are executed, it is recommended to delete all inner geometry. In most models, the various building element types are constructed in several layers. If the elements of the layers that represent for example furniture and inner structures are removed from the model, significant time can be saved. Just turning off these layers is not sufficient, as the elements are still part of the model then. They are hidden, but the script still goes through all these elements. They really have to be removed from the model.

7.4 Tool 4: Generation of the computational domain

7.4.1 Introduction

To obtain reliable results from the CFD calculations, a computational domain with sufficient dimensions has to be generated around the research area. The domain must be large enough to avoid that the flow pattern at the boundaries of the domain is influenced by the buildings in the research area. At the location where the domain ends, a boundary condition has to be defined that determines the nature of the flow problem. Distances from the research area to the boundaries of the domain are usually related to the maximum height of the buildings that are exposed to the flow. As already mentioned in Chapter 3, the Cost Action C14 workgroup gives recommendations for the size of the computational domain. In Figure 7.45 this recommended domain size is reproduced from Figure 3.11.

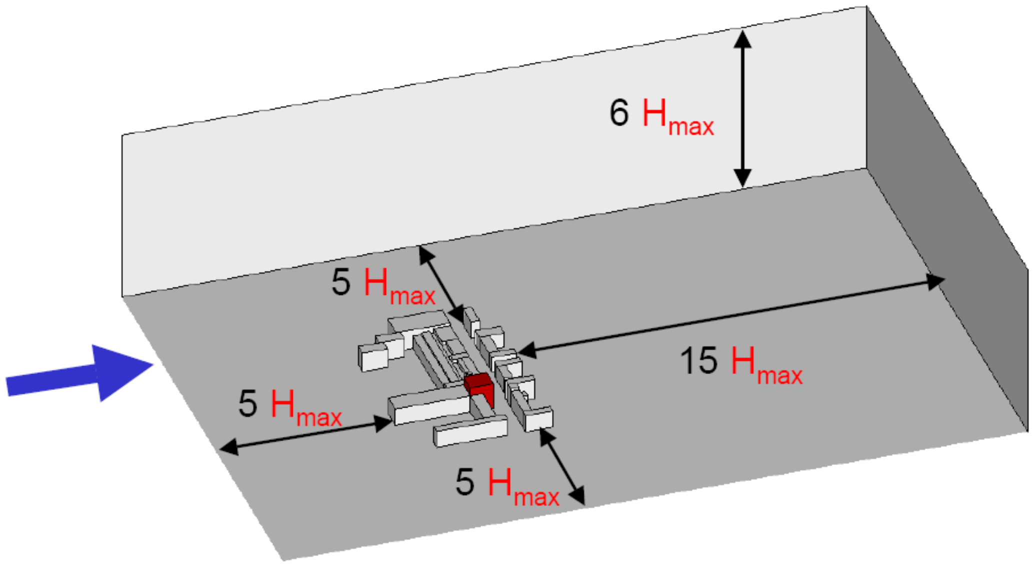


Figure 7.45: Recommended dimensions for a computational domain by the CAC14 workgroup (Franke, [8])

The inlet and side boundaries of the domain should be 5 H_{max} away from the area to be investigated. To allow a full development of the flow, the outlet boundary should be 15 H_{max} away from the research area. The top of the computational domain should be at least 6 H_{max} away from the tallest building to prevent an artificial acceleration of the flow over this building. However, according to Van Nalta [16], a survey of recent literature shows that a wide range of different domain sizes is used in practice. In his Master's thesis at the Structural Design Lab of Delft University of Technology, Van Nalta developed his own domain, the Van Nalta domain, which encloses all other domain sizes. Figure 7.46 shows the dimension of the Van Nalta domain.

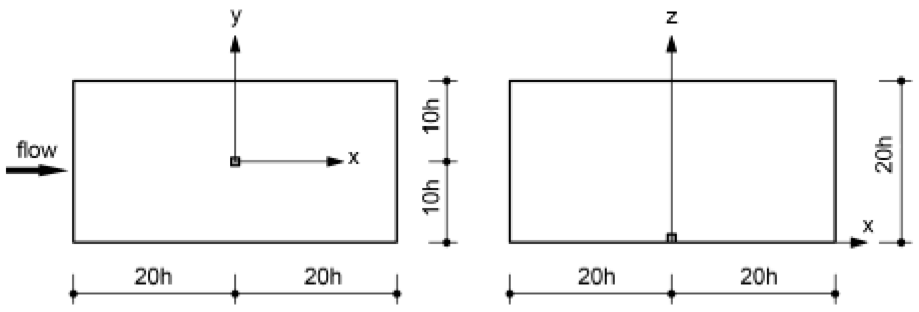


Figure 7.46: Top and side view with the dimensions of the Van Nalta domain (Nalta, [16])

The inlet and outlet boundaries are 20 H_{max} away from the research area, the side boundaries are 10 H_{max} away. The top of the computational domain is even 20 H_{max} away. This is very far in comparison with the recommendations of the Cost Action C14 workgroup. Calculations with the Van Nalta domain show that the dimensions of this domain are more than sufficient (Nalta, [16]). Velocity profiles at the upstream, side and top boundaries are unaffected by objects in the research area. The large size surely has an influence on the grid size. However, the influence is considered to be small, because cells are larger near the boundaries and the largest amount of cells is located near the center of the domain.

Because the Van Nalta domain is developed at the Structural Design Lab and quite some research is done with the domain recently, it will also be used in this thesis to perform the simulations. In the next paragraph, some more details about the domain are given.

7.4.2 The Van Nalta domain

Figure 7.47 shows the unmeshed structure of the Van Nalta domain. The size of the domain is related to the height h of the object placed inside the central cylinder. The coordinate system that is used is placed at ground level, in the center of the cylinder and has the x-axis in the flow direction, the y-axis in the cross-direction and the z-axis in the height direction. The total length of the domain is 40h (x-direction), the total width is 20h (y-direction) and the total height is 20h (z-direction). The domain contains three horizontal planes at z=0, z=5h and z=20h. The cylinder has a diameter of 5h.

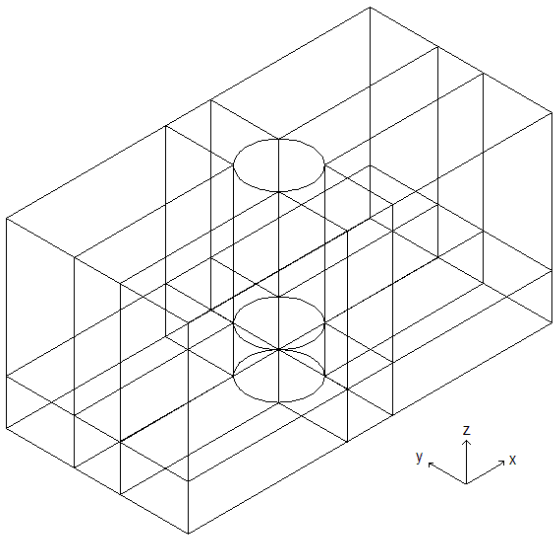


Figure 7.47: Structure of the Van Nalta domain (Nalta, [16])

The domain can be divided into control volumes using a mesh. Cells far away from the center can be larger, because the gradients of the flow parameters are generally small there. A uniform grid for these areas will reduce computing time. The mesh around the object itself should be unstructured to be able to input any shape. These demands require a hybrid grid.

In the domain of Figure 7.47, the object is placed in the lower part of the central cylinder. This part is called the *building environment* and the mesh should be unstructured here. To decrease the number of cells and the total computing time, a structured hexahedral grid is applied in the regions outside of the cylinder. These regions are called the *wind environment*. In order to automatically create the mesh of the wind environment, regardless of the object to be studied, the mesh outside this environment should not be affected by the object inside the cylinder. This demand is met in the Van Nalta domain. Figure 7.48 shows the mesh of the domain and some boundary conditions.

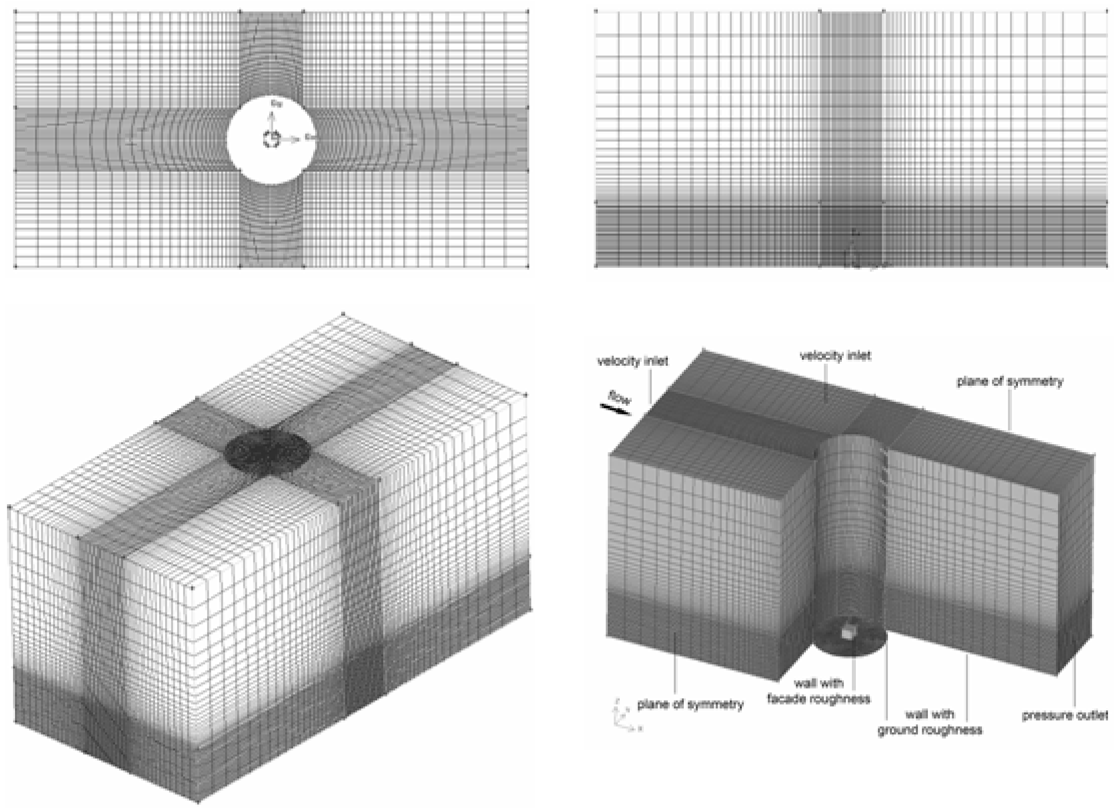


Figure 7.48: Mesh and some boundary conditions of the Van Nalta domain (Nalta, [16])

The topmost pictures of Figure 7.48 show the mesh on the ground floor of the wind environment and the side boundary. The bottom pictures show the mesh of the outer boundaries and the boundary conditions for the domain. As can be seen, the cell size increases as the distance to the central object increases. The mesh near the object is the most important part of the mesh, since the largest gradients in the flow will be located here. For 3D unstructured meshes, the tetrahedron cell types are most suitable as they are flexible and as they can be used for automated meshing of complex geometries (Snijders, [23]). That is why the cylinder of the domain can best be meshed using an unstructured grid with small tetrahedral cells.

To solve a flow problem, boundary conditions are required. The Van Nalta domain has a velocity inlet boundary at the front, where the velocity is defined as a logarithmic profile by using a user-defined function. The parameters of the logarithmic profile are based on the Dutch building code. The outlet of the domain is defined as a pressure outlet boundary, where the pressure is set to be the atmospheric pressure. At the outlet, the studied object is supposed to have no effect anymore on the flow. The Van Nalta domain has symmetry walls on the sides, which can be seen as very smooth walls. At a symmetry boundary, the components normal to the boundary are all set to zero. This means that no energy can leave the domain through the boundary. The top of the domain is also modeled as a velocity inlet. The bottom of the domain finally has a wall boundary condition. Here the no-slip condition is applied, meaning that the flow velocity in all directions at the wall is set to zero.

7.4.3 Purpose

The purpose of the fourth design tool is to create the Van Nalta domain, depending on the radius of the research area and the maximum height of a building in the research area. The radius and the height of the largest building in the environment follow from the tools to generate the 3D research area. Together with the height of the building of interest, the maximum height of the building in the research area can be determined. Until now, the Van Nalta domain was only used to simulate the wind effects on single objects. These objects were placed in the bottom cylinder of the domain. To simulate the wind effects on complex building models that are placed in a built environment, it is desired to also use this domain. The built environment, that has a circular ground surface, will be placed in the bottom cylinder then. The next pictures again show the dimensions of the original Van Nalta domain.

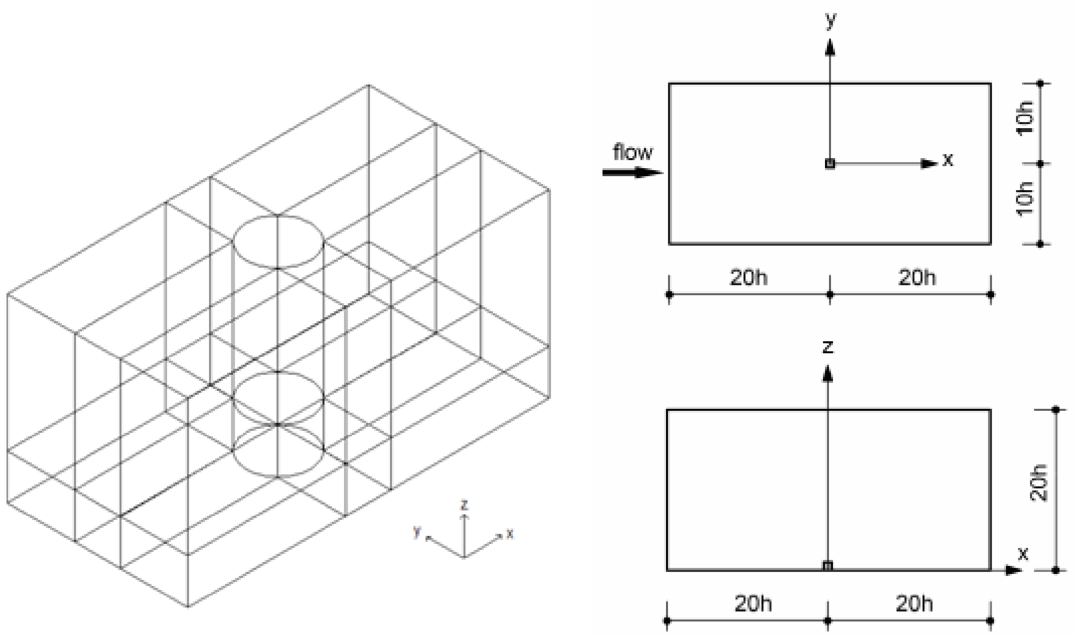


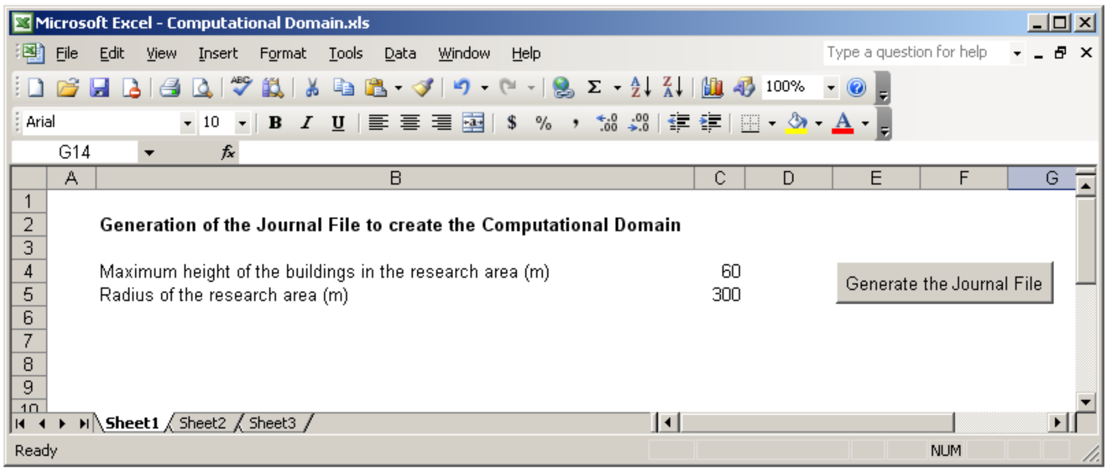
Figure 7.49: Dimensions of the Van Nalta domain (Nalta, [16])

Originally, the dimensions of the domain were only dependent on the height of the object of concern. From now on, the radius of the research area will also determine the dimensions of the domain. The total length and width of the domain remain 40h respectively 20h, but the radius of the cylinder will become the radius of the research area that is generated with the first design tools. This means that the *building environment* of the domain increases and the *wind environment* decreases. However, as the dimensions of the domain are still more than sufficient in comparison with several other domains that are used in practice, it is likely that this will not affect the results.

Gambit, the Fluent pre-processor, will be used to generate and mesh the computational domain. An advantage of Gambit is that it can run so-called journal files. In a journal file, text based commands, arranged as they would be typed into the program or entered through a graphical user interface, can be scripted. When running the journal file, Gambit will work over the commands given in the file, building the model that is scripted.

The purpose of the fourth design tool now is to create the journal file with which the computational domain can be generated in Gambit, depending on the maximum height and radius of the research area. The generation of the meshes for the *wind environment* will also be scripted in the journal, just as the definition of some boundaries. After the domain is generated, it is the intention that the user itself places the model of the research area into the lower cylinder of the domain. Only the objects in the research area and the cylinder itself have to be meshed manually then. Finally the boundaries of these objects have to be defined.

The journal file can be generated using a Visual Basic macro that is run in Microsoft Excel. Figure 7.50 shows the interface of this macro in Excel. It is important to set the regional settings in Windows to English (UK) to prevent errors with dots and commas.


Microsoft Excel - Computational Domain.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
10
G14
Sheet1
Sheet2
Sheet3
Ready
NUM

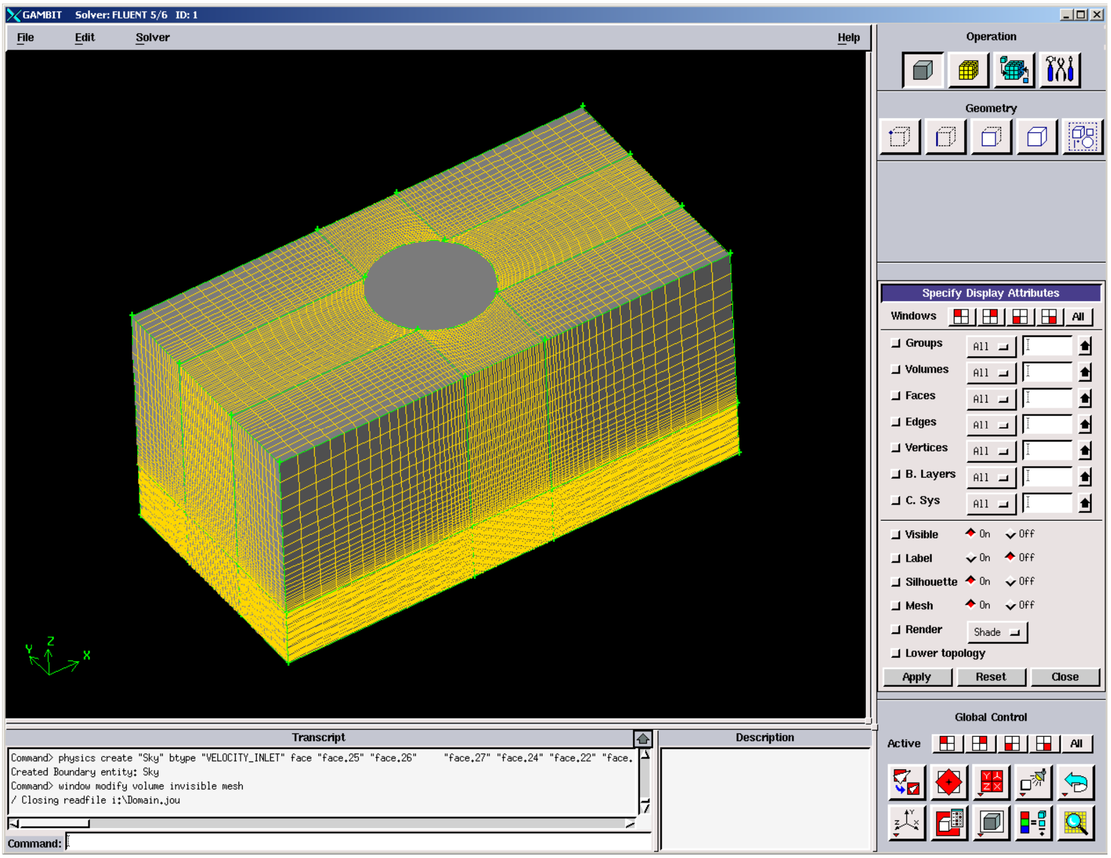
Generation of the Journal File to create the Computational Domain

Maximum height of the buildings in the research area (m)	60
Radius of the research area (m)	300

Generate the Journal File

Figure 7.50: Macro to generate the journal file with which the domain can be created

When the journal file is generated and run in Gambit, the computational domain including the mesh of the *wind environment* can be created. The result is given in Figure 7.51. The length and width in this example are 2400 m respectively 1200 m; the radius is 300 m. The next step is to manually import the research area with the building of interest into the lower cylinder of the domain. The user only has to mesh the buildings of the research area and the lower and upper cylinder then. The Visual Basic script to generate the journal file is given in Appendix L.



```
Command> physics create "Sky" btype "VELOCITY_INLET" face "face.25" "face.26"    "face.27" "face.24" "face.22" "face.
Created Boundary entity: Sky
Command> window modify volume invisible mesh
/ Closing readfile i:\Domain.jou
```

Figure 7.51: Result of the journal file to generate the computational domain

7.5 Conclusion

In this chapter the various design tools to setup the geometry for a CFD calculation are discussed extensively. The purpose of these tools is to assist the user of the Virtual Wind Tunnel to predict the wind loads on a building or structure and compare alternative geometries in a relatively short time without much interference of the user. Because the wind around a building is influenced by its surroundings, tools are developed to create a restricted 3D environment for a certain area. Information about the environment is obtained from GIS technology. Another tool is developed to simplify the building model of interest. Usually these building models contain lots of detail and inner geometry that are not relevant to determine the wind loads on a building. Several methods are developed that determine the outer points and generate the enclosing curves for several height levels of the model. With a special fitting technique these curves can be simplified if necessary to reduce the amount of detail. Finally a surface can be generated through the enclosing curves that wraps the model. It is just this hollow surface model that will be used by the CFD software to calculate the wind loads on the building. If the influence of the environment has to be taken into account, the surface model of the building can be placed in the 3D model of the environment. Such an integrated model of the research area is given in Figure 7.52. The result of the methods to simplify a building model is placed in the 3D model of a part of the Delft University of Technology district. The model of concern is the SketchUp model of an office building.

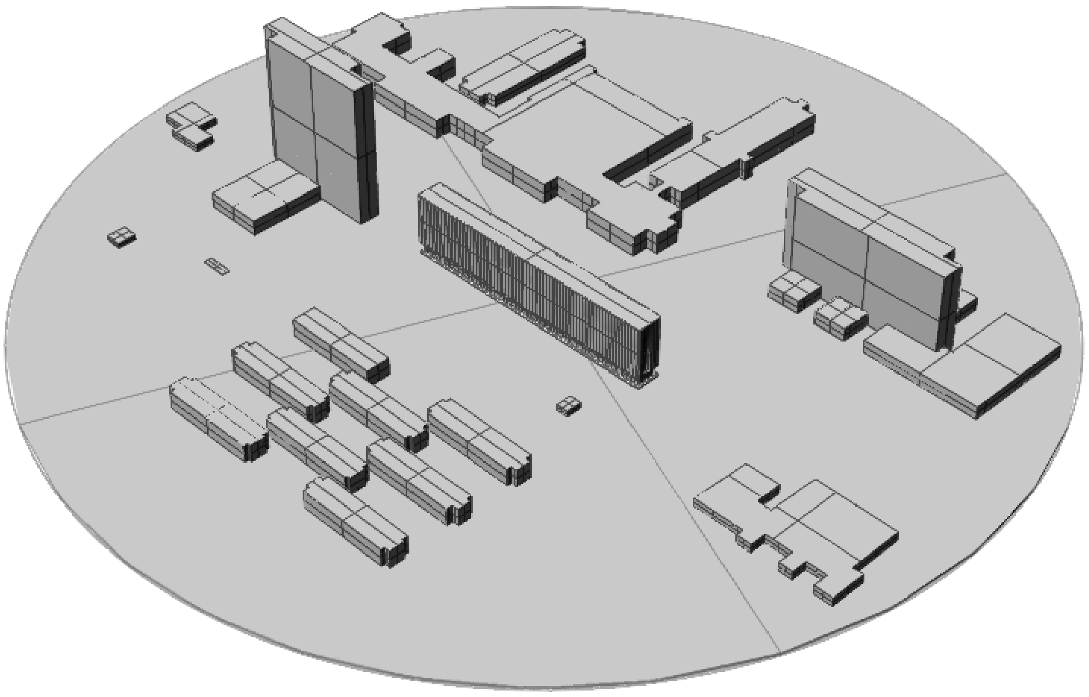


Figure 7.52: Integrated model of the research area that can be used for CFD calculations

The office building forms the central building of the research area. The simulations to predict the wind loads are performed in a computational domain where the model of the research area is imported. To obtain reliable results, the domain must be large enough to avoid influencing of the flow pattern. The dimensions of the domain depend on the maximum height of the buildings in the area and the radius of the research area. These values are returned when the design tools to generate the research area are executed. With the height and radius, the fourth design tool that is developed can finally be used to create the Van Nalta domain with the required dimensions. The 3D model of the research area has to be placed manually in the domain then. After meshing the area and setting up the calculation, the CFD software is able to predict the wind loads.

The design tools to simplify the building model of interest and generate the 3D model of the research area are developed in Rhinoceros. Because several methods are developed to find the outer points of the model of interest, quite some scripts are written for the various tools. A special toolbar for Rhinoceros is therefore developed that enables the user to quickly load the various scripts. The toolbar is called *Virtual Wind Tunnel* and the interface is given in Figure 7.53.

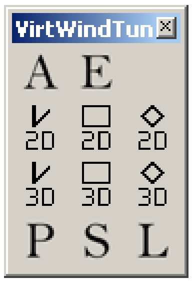


Figure 7.53: Toolbar for Rhinoceros that supports the written scripts of the design tools

With the toolbar it is possible to perform all processes that are needed to generate a 3D model of the research area with a simplified surface model of the building of interest. The toolbar is divided in four rows. In the top row, the *A* and *E* toolbar buttons refer to the *Area* script and the *Extrusion* script. With the *Area* script the restricted 2D research area can be generated from the Top10Vector dataset. With the *Extrusion* script the generated 2D research area can be extruded to a 3D model on the basis of the AHN dataset.

The second row of the toolbar contains three buttons that refer to the scripts to find the outer points of a 2D model. The first button refers to the *Rotating Lines* method, the second button refers to the *Rectangle* method and the third button refers to the *Rotated Square* method. Each method has two variants. The first variant determines the intersection between the line element of the method and the lines, surfaces and solids of the model. The second variant determines the intersection between the line element and a mesh element. This second variant can only be used in Rhinoceros 4. The two variants of each method are supported by the toolbar. Clicking with the left mouse button on the toolbar button will load the first variant of the method; the second variant will be loaded when clicking with the right mouse button on the toolbar button.

The third row of the toolbar contains the buttons that refer to the scripts to find the outer points of a 3D model. The methods are the same as in the second row, but the scripts are extended to determine the outer points for several height levels. When running these scripts, the user also has to enter the height of the model, the starting height of the method and the step size.

The fourth row finally contains another three buttons. The *P* toolbar button refers to the *Preprocessing* script to convert block instances or meshes into the triangular surfaces. The *S* button refers to the *Curve Simplification* method that simplifies the enclosing curves with the *NURBS fitting* method. The *L* button finally refers to the *Loft* script that lofts surfaces between the enclosing curves and creates the wrapped surface.

8. CFD Calculations

To finish the thesis project some CFD calculations are performed for several test cases that are generated with the various developed design tools. It is not the intention to obtain very accurate results from the calculations, as there is still quite some uncertainty about the use of CFD in wind engineering. The calculations are just performed to test if the various design tools work for CFD applications. Additional research is required to investigate the accuracy of the results, especially for the calculations with more complex building models.

Two conceptual grid generation methods will be discussed in this chapter, but they are still not optimal. Meshing complex geometry requires lots of cells, but the capacity of the current desktop computers is the limiting factor for the amount of cells that can be used. With the actual hardware it is very hard to develop a proper grid that simulates the flow pattern on all places of the research area accurately. The simulations that are discussed in this chapter and the recommendations that are given must therefore be seen as a first step to the development of a proper grid generation method for meshing complex geometry. Additional research is required to develop the methods further. However, for simple geometry it is already difficult to obtain reliable results with the actual computer resources, as the amount of cells that can be used is quickly consumed. It might take some years before the wind effects on entire built areas can be simulated accurately.

The simulations that will be discussed in this chapter are executed in the Van Nalta domain, where the models are placed in the lower part of the central cylinder. The domain itself is already meshed during the generation of the domain. After the model of interest is imported in the domain, only the cylinder containing the research area has to be meshed manually. In the graduation studies of Van Nalta [16] and Snijders [23], a grid generation method is developed to mesh the cylinder containing simple geometry, like a cube. This method is used as a starting-point for the development of a more general grid generation method that can be used to mesh more complicated geometry. However, verifying the results of the calculations is difficult as there is hardly any reference information available for the discussed test cases. Especially for complex building geometry it is very hard to judge the results. The only way of verifying them is to compare the results with real wind tunnel studies.

In the next paragraphs two grid generation methods will be discussed that are developed for meshing the several test cases. The first method uses hexahedral elements and can be used to mesh simple geometry. The second method uses tetrahedral elements and is able to mesh more complicated geometry. For the simple building models the results of the CFD calculation can be compared with a calculation according to the Dutch building code or the Eurocode. In Paragraph 8.6 such a comparison will be made for a building model of the faculty of Electrical Engineering of Delft University of Technology. In additional studies the results of the CFD calculations with more complex building geometry should be compared with real wind tunnel tests to verify the results of these calculations. As for this thesis no reference information is available for the performed CFD calculations with the more complex geometry, the calculations are just executed to demonstrate the potential of the grid generation methods.

8.1 Single cube: 60 x 60 x 60 m

8.1.1 Grid generation

The first object that is used to generate the geometry for the CFD calculations with the various design tools is the cube that is used in the previous graduations studies of Van Nalta [16] and Snijders [23]. The cube has dimensions of 60 x 60 x 60 m^3. With the *Rectangle* and *Loft* methods a surface is wrapped around the model that fits the original cube accurately.

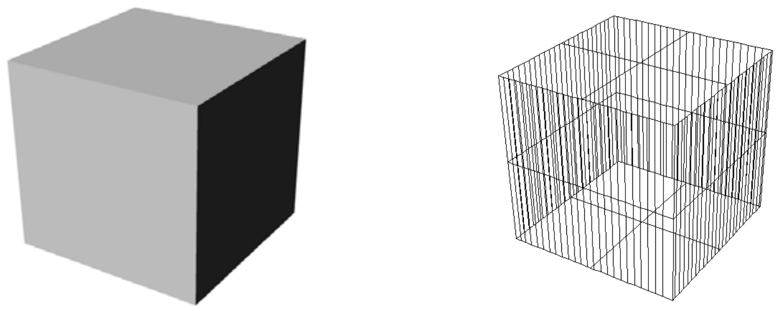


Figure 8.1: Wrapped surface model (right) of the original cube (left)

The surface model of the cube is placed in the lower cylinder of the Van Nalta domain. Only the lower and upper cylinders of the domain have to be meshed manually, as the rest of the domain is already meshed during the generation of the domain. The regions of the domain outside the cylinders are meshed with structured hexahedral elements as they give best results in computing time and reduce the numerical error. Cells far away from the central object are larger as the gradients of the flow parameters are generally small there. This also has a positive influence on the required amount of cells to mesh the domain. The computational domain is divided in two levels. In the lower part of the domain the mesh is denser and the cells have the same height everywhere. The height of the lower part of the domain is 5 times the height of the highest object in the research area. The height of the upper part of the domain is 15 times the height of the highest object in the area.

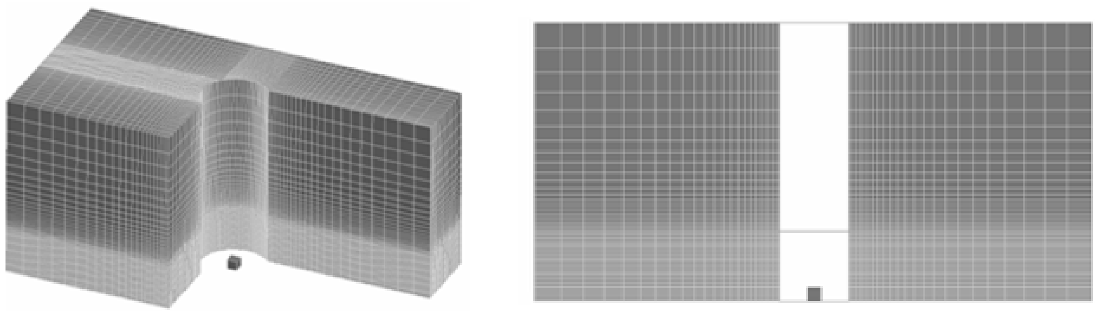
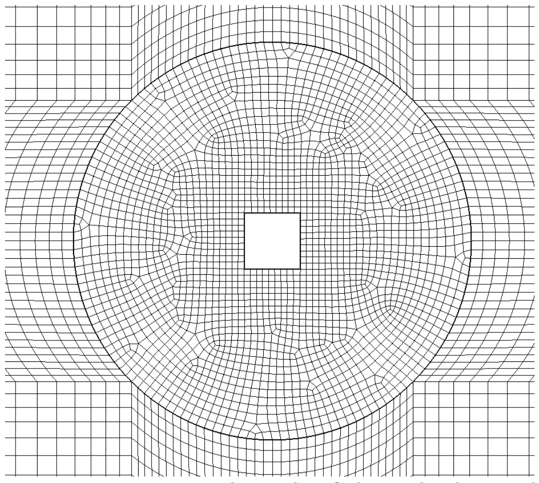


Figure 8.2: Surface model of the cube placed in the unmeshed lower cylinder of the domain

In his graduation study, Van Nalta developed a method to mesh the lower cylinder containing a cube with a high quality grid. To limit the amount of cells, hexahedral elements were used to mesh the cylinder. As an introduction to the grid generation process, this method is now used to mesh the lower cylinder containing the surface model of the cube. After the model is placed in the cylinder, a size function is attached to the ground edges of the cube and the bottom of the cylinder. With this function the growth in cell size from the cube's bottom to the edge of the circular ground face is controlled smoothly. As only a small growth factor is used, the grid quality remains high. To facilitate a smooth transition from the cylinder mesh to the mesh outside the cylinder, the growth in cell size by the size function is restricted by attaching the mesh distribution on the circular edge of the cylinder, defined by the outside mesh, to the function. After the size function is attached to the ground face, it is meshed with quadrilateral elements using the pave scheme. This creates an unstructured grid on the bottom of the cylinder with small cells at the edges of the cube that grow smoothly over the ground surface to the edge of the cylinder. The mesh distribution on the ground edges of the cube is then linked to the top edges of the cube. With this procedure, the mesh nodes on the ground edges are in fact copied to the top edges of the cube to retain the same cell size. Then the roof is meshed with quadrilateral elements using the map scheme. This creates a regular, structured grid on the roof of the cube. The vertical side edges of the cube are then meshed using an interval size. This divides the edges in intervals of the specified size. Finally the lower cylinder is meshed with hexahedral elements using the Cooper scheme. By specifying the bottom face of the cylinder and the roof of the cube as source faces, the Cooper scheme projects the mesh on these faces with a uniform grading to the top of the cylinder, forming 3D elements. In fact, the scheme just extrudes the 2D grid on the faces to form a 3D grid. The height of the elements is defined by the size of the intervals on the vertical edges of the cube. As the chosen interval size is equal to the start size of the size function, hexahedral cells of more ore less equal width, length and height are the result. The process is then repeated for the upper cylinder, where the mesh distribution on top of the bottom cylinder is used to project the mesh through the upper cylinder.



- Attach a size function to ground edges of the cube and the bottom of the cylinder
  - Start size = 0.1*h
  - Growth rate = 1.02
  - Size limit = h
- Mesh the ground surface with quadrilaterals using the pave scheme
- Link the mesh on the ground edges of the cube to the top edges of the cube
- Mesh the top surface of the cube with quadrilaterals using the map scheme
- Mesh the vertical side edges of the cube with interval size 0.1*h
- Mesh the cylinder with hexahedral elements using the Cooper scheme

Figure 8.3: Ground mesh of the cylinder and a summary of the grid generation method

In Figure 8.3 the mesh on the ground surface of the lower cylinder and a part of the ground mesh around the cylinder is given. Starting from the ground edges of the cube, the cell size increases smoothly to a maximum size at the edge of the cylinder. This is due to the size function. The cells outside the cylinder determine the maximum cell size of the cells inside the cylinder. Because the cells have to connect perpendicular to the cube and the cylinder, a transition in cell direction occurs at a certain distance from the cube. This transition should not be located to close to the central object. The ground mesh, together with the mesh on top of the cube, is then projected to the top of the cylinder with a uniform grading. The grading is determined by the interval size on the vertical edges of the cube. The grid generation method is summarized in the textbox next to Figure 8.3. The variables that are used to mesh the lower cylinder are specified in Table 8.1.

Size function attached to the ground edges of the cube and the circular ground face of the bottom cylinder	Start size	6 (= 0.1 x h)
	Growth rate	1.02
	Size limit	60 (= h)
Side edges of the cube	Interval size	6 (= 0.1 x h)

Table 8.1: Variables used to mesh the lower cylinder containing a cube

A see-through of the mesh of the lower cylinder is given in the left picture of Figure 8.4. The inner cylinder mesh is not given, but must be imagined as volume elements with dimensions corresponding to the cylinder's bottom and the cube's roof mesh elements, with a uniform height. The height of the elements is defined by the interval size on the vertical edges of the cube. The volume elements are then projected through the cylinder to the top of the cylinder. The right picture of Figure 8.4 gives a cross-section of the cylinder mesh. With this grid generation method applied to the cube volume, the amount of cells in the lower cylinder is 177.640. The upper cylinder contains 56.856 cells and the domain around the cylinder contains 316.720 cells. The total amount of cells in the computational domain is 491.226.

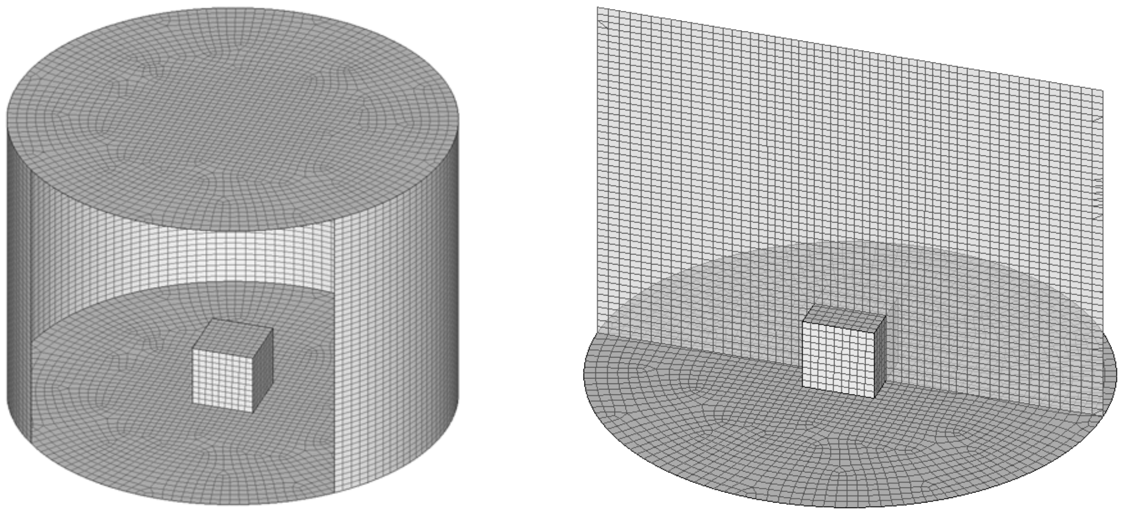


Figure 8.4: See-through of the cylinder mesh and cross-section of the mesh near the cube

The next step is to define the remaining boundaries of the domain. Most boundaries are already defined during the generation of the domain with the fourth design tool. For all objects that are placed in the lower cylinder and for the ground surface of the cylinder, the boundaries have to be defined manually. The ground surface and the objects will be defined as wall elements with the no-slip condition, meaning that the flow velocity in all directions at the wall is set to zero. The parameters of all boundaries of the domain are defined in Fluent during the setup of the calculation.

8.1.2 Setting up the calculation in Fluent

Once the entire computational domain is meshed and the boundary zones are defined, the calculation can be setup in Fluent. The calculation will be steady-state, meaning that the governing equations do not contain time-dependent terms. The guidelines that are used to setup the calculation are formulated by Van Nalta and are with some small adaptations given in Appendix M. In this paragraph some essential steps of the setup procedure are discussed more extensively.

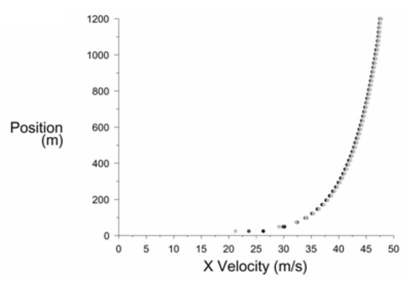
Boundary conditions

The boundary zones of the domain that were defined during the grid generation process are specified by assigning physical conditions to the zones. The quality of the solution largely depends on the quality of the boundary conditions. The following conditions need to be applied. They can be defined as constant values or by using a user-defined function.

- Flow inlet and outlet parameters
- Wall parameters
- Side parameters

Flow inlet and outlet parameters

The velocity inlet boundary condition is used to define the velocity profile at the inlet boundary. The logarithmic velocity profile that is prescribed in the Dutch building code NEN 6702 is used to define the inlet conditions. The profile is characteristic for the wind climate in Zuid-Holland for unbuilt areas and is given in Figure 8.5.



$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z-d}{z_0}\right) \qquad (8.1)$$

Figure 8.5: The logarithmic velocity profile is used as the inlet boundary condition

The figure gives the velocity profile at the inlet boundary of the computational domain in Fluent. In the equation that describes the profile is U(z) the mean wind speed, u^* the friction velocity, z_0 the roughness height, d the displacement length and κ the Von Karman constant. According to the Dutch building code NEN 6702, the roughness height for the unbuilt area of the characteristic wind climate is $z_0 = 0.2$ m. At the inlet boundary also a value for the turbulence quantities needs to be defined. As the realizable *κ-ε* turbulence model will be used for the calculations, the turbulent kinetic energy *κ* and the turbulent dissipation rate *ε* have to be defined. A user-defined function is used to add the turbulence quantities and the logarithmic velocity profile to the calculation. It is given in Appendix M.

The top of the computational domain is also defined as a velocity inlet with the same function as the real inlet. In this way a constant horizontal velocity u_{max} is defined at the top of the domain. At the outlet of the domain a pressure outlet condition is used where the pressure is equal to the atmospheric pressure.

Wall parameters

Wall boundary conditions need to be defined for the ground surfaces of the domain and for the surfaces of the building. With these boundary conditions a roughness is given to the ground and the cube to model the environment and the façades. The parameters that need to be defined to give the walls a certain roughness are the roughness length K_s and a roughness constant C_s. According to Van Nalta [16] the wall parameters can be determined with:

Roughness length:	$K_s = 13\ Z_0$ for non-uniform roughness
	$K_s = 20\ Z_0$ for uniform roughness
Roughness constant:	$C_s = 0.75$ for non-uniform roughness
	$C_s = 0.50$ for uniform roughness

As the ground roughness is usually considered as non-uniformly distributed, the roughness constant of 0.75 is attached to the bottom of the domain in Fluent. With a roughness height of $z_0 = 0.2$ m for the unbuilt area, the roughness length that will be attached to the ground surfaces becomes 2.6. The surfaces of the cube are also defined as wall boundaries. For the façades of the cube, the roughness is considered to be uniformly distributed and a roughness constant of 0.5 is attached to the sides of the cube. The Eurocode gives friction coefficients for three types of façades. These friction coefficients are 0.01 for smooth surfaces, 0.02 for rough surfaces and 0.04 for very rough surfaces. As the façades of the cube are modeled as very rough surfaces, the roughness length that will be attached to the cube becomes 0.8.

Sides
At the sides of the domain the symmetry condition is applied to the boundaries. This means that no energy can leave the domain through the boundary, as the components normal to the boundary are all set to zero.

Convergence

When running a CFD simulation, the set of equations are solved numerically in an iterative process. The calculation is repeated until a solution with a sufficient and user-specified accuracy is found. For any given conservation equation, an approximate solution is obtained at each iteration that results in a small imbalance in the conservation statement. When the flow problem is setup correctly, the imbalance in each cell decreases as the iteration process continues. This imbalance is called the residual. The total residual for each variable across the entire domain is the sum of the absolute values of the individual cell residuals. Convergence criteria for the residuals indicate that a certain level of convergence has been achieved. Before a simulation is run it is important to wonder what accuracy is needed. For aerodynamic applications for example, deep convergence is required if the lift and drag coefficients are predicted. If one is simply looking for some approximating flow features, rough convergence is probably acceptable. According to Fluent (Internet, [9]) the default convergence criterion of 10^{-3} is sufficient for most problems. However, for computational wind engineering this criterion seems to be insufficient. For a calculation that is initialized from the velocity inlet, leading to a constant wind speed in the entire domain, Van Nalta recommends a convergence criterion of 10^{-9}.

When a solution is fully converged, the values of the residuals have leveled out to some value and then stopped changing as the iteration process continues. If the convergence criteria are set too strictly, no solution is found and the iteration process continues until a specified maximum number of iterations. An example is given in the left picture of Figure 8.6. The solution is fully converged to a value of about 10^{-18} at minimum, but the convergence criteria are not met as the iteration process continues. Another possibility is that the residuals for all variables fall below the convergence criteria, but are still decreasing. The solution is still changing, but the desired accuracy is reached. An example is given in the right picture of Figure 8.6, where the iteration process stops after 2750 iterations. Although the solution is not fully converged, the convergence criteria for all variables are met.

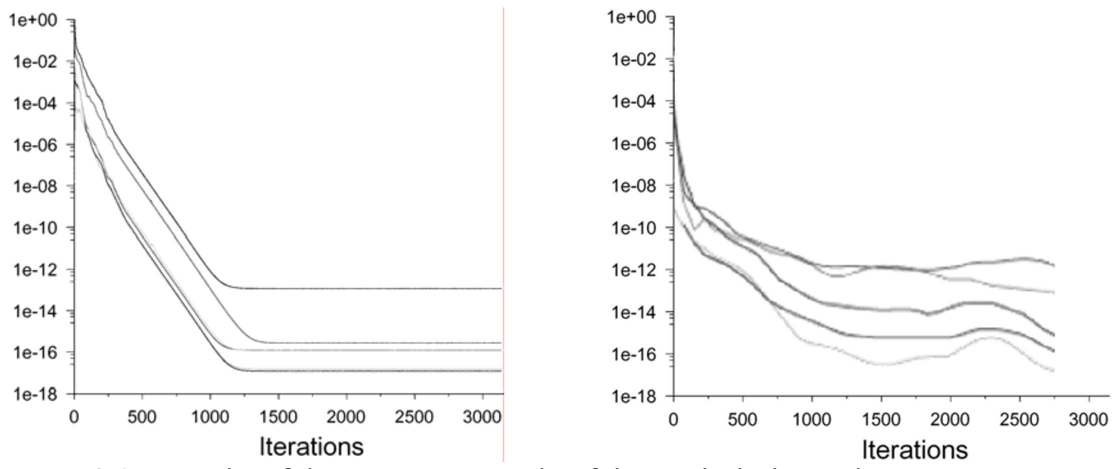
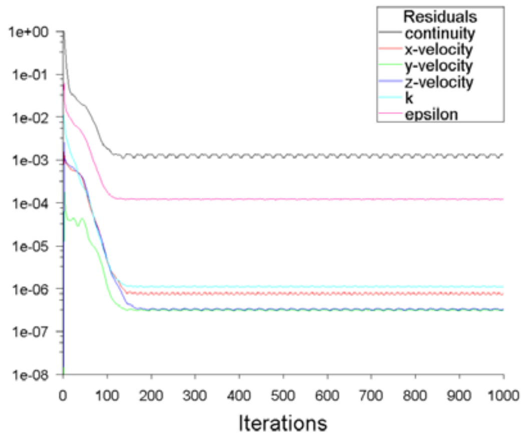


Figure 8.6: Examples of the convergence paths of the residuals during the iteration process

8.1.3 Results

The flow problem discussed in the previous paragraphs is solved iteratively with a maximum of 1000 iterations and convergence criteria of 10e^{-9}. With a mesh consisting of 491.226 cells, the simulation took about 3½ hours. In the left picture of Figure 8.7 the residuals of the variables are plot. During the simulation the continuity, x-, y- and z-velocity and the turbulence variables κ and ε are monitored. The solution is fully converged, but the convergent criteria are not met. The criteria of 10^{-9} are probably too high.



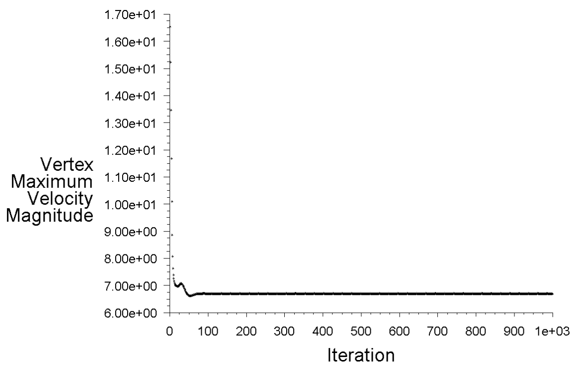
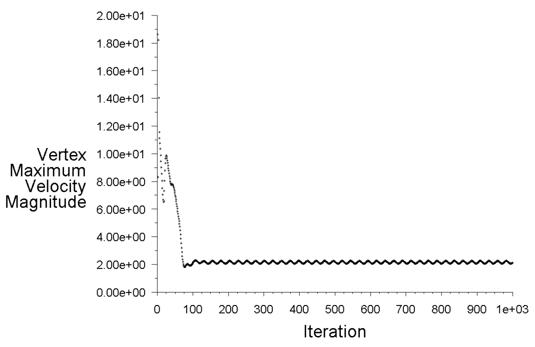


Figure 8.7: Convergence of the residuals (left) and surface monitor above the roof (right)

Monitoring the residuals is not the only option to judge the convergence of a solution. Also surface and force monitors can be applied to examine the convergence. A surface monitor can be a plane or a point and can for example be placed above the roof of the object and in the wake behind the object. Monitoring the velocity magnitude in those points for example during the iteration process helps to judge the solution. When they level out to a certain value, the solution is considered to be converged. The right picture of Figure 8.7 gives the convergence history of the velocity magnitude just above the roof of the cube. The left picture of Figure 8.8 gives the convergence history of the velocity magnitude in the wake behind the cube. Force monitors can finally be used to monitor the forces on a certain surface. The right picture of Figure 8.8 gives the monitored drag at the back of the cube.



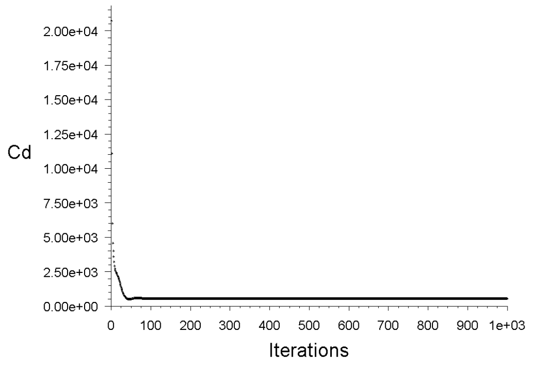


Figure 8.8: Surface monitor in the wake (left) and force monitor of drag at the back (right)

The convergence of the residuals in combination with surface or force monitors provides a much better idea of the convergence of a solution. From the figures on this page it can be concluded that the solution of the calculation with the cube is fully converged. As was to be expected, the results are the same as Van Nalta got from his calculations. The results can be presented further with colorful plots or derived in absolute quantities, such as maximum force, pressure or momentum. On the following page some results are presented in a visual way.

When the simulation process is finished and the flow field variables are found, information about the flow parameters can be presented in a visual form to check if the solution is physically reasonable. With isometric views insight into the pressure distribution on the façades of a model can be provided. Important or critical places are easy to identify. An example is given in the left picture of Figure 8.9. Also the turbulent kinetic energy can be displayed on the model to show the critical turbulent zones for example.

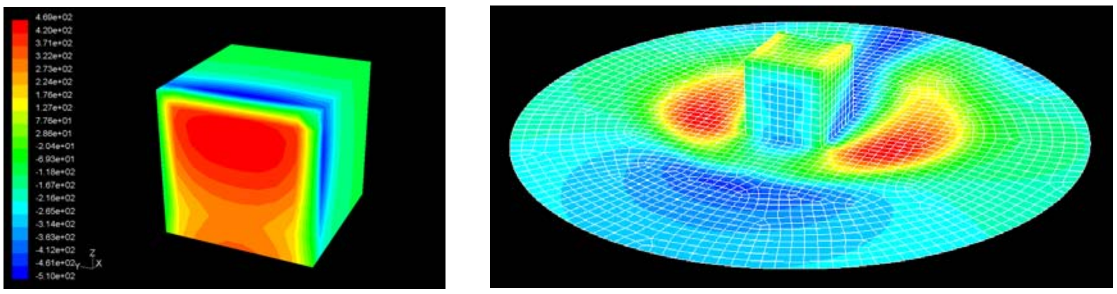


Figure 8.9: Isometric views of pressure distribution and turbulent kinetic energy distribution

With contours of velocity magnitude and pathlines a visualization of the flow around the model can be given. In the left picture of Figure 8.10 the velocity contours around the cube are displayed. Near the ground and at the back of the cube the air velocity is small. Due to the logarithmic wind profile that is used, the air velocity increases with the height. Also the wake structure and size of the wake at the back of the cube are shown. The results seem to be physically reasonable. In the right picture of Figure 8.10 pathlines colored by the velocity magnitude are displayed around the cube. Pathlines represent the track that particles will travel when they are released in the fluid motion. Especially for the visualization of complex three-dimensional flows they are an excellent tool.

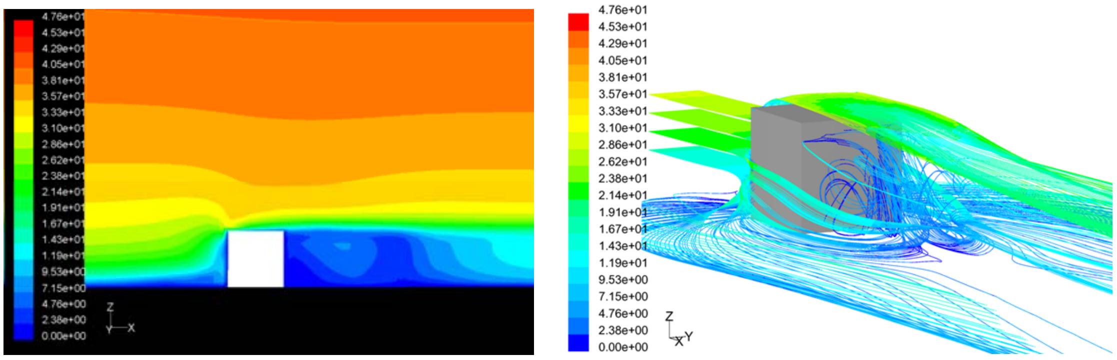


Figure 8.10: Contours of velocity magnitude and pathlines to visualize the flow

There are many ways of representing and analyzing the solution of a simulation. When CFD results are compared with experiments a problem arises, as for experiments only a limited amount of data can be obtained or visualized. The best way to compare results with experiments is the use of pressure coefficients. The pressure coefficient is related to the wind speed at building height. For the front façade of the cube an example is given in Figure 8.11. At 45 m, or at ¾ of the height, a maximum pressure coefficient of 0.8 is found. This corresponds to the Dutch building code, where the wind load is also calculated with a coefficient of 0.8 for pressure and a coefficient of 0.4 for suction.

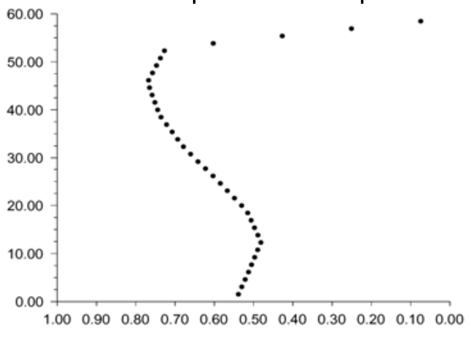
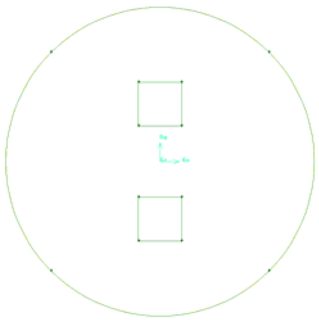


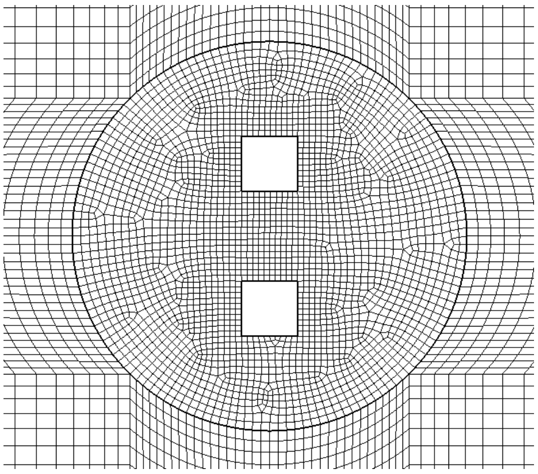
Figure 8.11: Pressure coefficients at the middle of the front façade

8.2 Two cubes



As a first step to the development of a grid generation method for models containing several buildings, a second cube is added to the research area. The cubes are placed at the x=0 axis with a distance between them corresponding to the distance between a cube and the edge of the cylinder. Nevertheless, problems aroused immediately when the model was meshed. The grid generation method discussed in the previous paragraph seemed not suitable for meshing a cylinder containing multiple objects. The method also did not work for a single cube that was moved from the center of the cylinder. The structured mesh that is applied on the roof of the cube using the map scheme appeared to be the cause of the method's failure. After the ground surface of the cylinder is meshed using the size function, the mesh distribution on the ground edges of the cube is copied to the top edges. This distribution is then used to mesh the roof of the cube. However, when the cube is not placed in the middle of the cylinder, the distribution of the mesh on the ground edges of the cube is not equal for all edges. The mesh distribution on the top edges of the cube is therefore also not equal. When the roof of the cube was meshed with the map scheme, the cylinder could not be meshed with the Cooper scheme. Using the pave scheme to mesh the roof of the cube with an unstructured mesh appeared to be the solution.

After adapting the grid generation method it was possible to mesh the cylinder containing the two cubes. First a size function is attached to the ground edges of the cubes and the bottom of the cylinder to smoothly control the growth in cell size. Then the ground surface and the roof of the cubes are all meshed with quadrilateral elements using the pave scheme. This results in an unstructured grid on both the bottom of the cylinder and the roof of the cubes. Then the vertical edges of the cubes are meshed using an interval size between the mesh nodes. Finally both cylinders are meshed with hexahedral elements using the Cooper scheme. After the resulting boundary zones are defined, the calculation is setup in Fluent. The same procedure is used as discussed in Paragraph 8.1.2. In Figure 8.12 the ground mesh of the cylinder and the grid generation method is given. In Table 8.2 the variables are summarized that were used to mesh the cylinder.



- Attach a size function to ground edges of the cubes and the bottom of the cylinder
    - Start size = 0.1*h
    - Growth rate = 1.02
    - Size limit = h
- Mesh the ground surface with quadrilaterals using the pave scheme
- Link the mesh on the ground edges of the cubes to the top edges of the cubes
- Mesh the top surfaces with quadrilaterals using the pave scheme
- Mesh the vertical side edges of the cubes with interval size 0.1*h
- Mesh the cylinder with hexahedral elements using the Cooper scheme

Figure 8.12: Ground mesh of the cylinder and a summary of the grid generation method

Size function attached to the ground edges of the cube and the circular ground face of the bottom cylinder	Start size	6 (= 0.1 x h)
	Growth rate	1.02
	Size limit	60 (= h)
Side edges of the cube	Interval size	6 (= 0.1 x h)

Table 8.2: Variables used to mesh the lower cylinder containing two cubes

In the next figure the convergence of the residuals is given. The solution appears to be fully converged as the residuals have leveled out to some value and stopped changing as the iteration process continued.

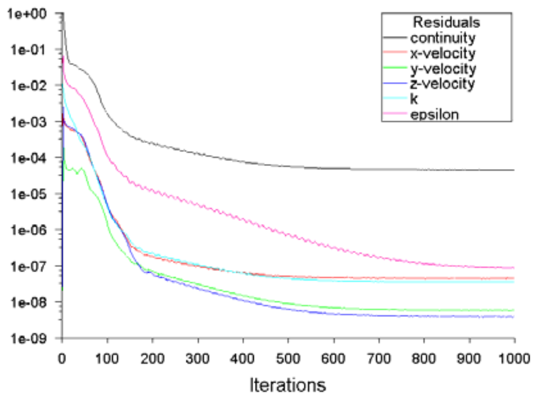


Figure 8.13: Convergence of the residuals

In Figure 8.14 some other results of the simulation are given. In the left picture streamtraces are displayed together with the turbulent kinetic energy on the ground and the surfaces of the cubes. The picture is generated in Tecplot (Internet, [24]), another CFD post-processor. The streamtraces show the flow pattern around the obstacles and are similar to the pathline rakes of Fluent. The right picture shows the flow pattern in top-view using pathlines in Fluent.

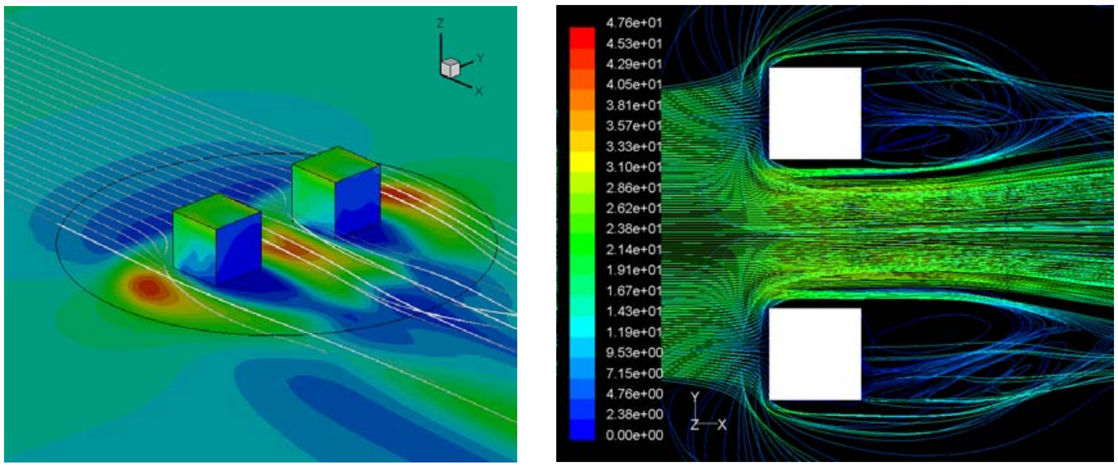


Figure 8.14: Streamtraces and pathline rakes show the flow pattern around the two cubes

To test the adapted grid generation method further, some other models are used to investigate the possibilities and shortcomings of the method. In the next paragraph a calculation with the building of the faculty of Electrical Engineer is discussed. The building model is derived from the 3D model of the Delft University of Technology district that was generated with the first design tools.

8.3 Faculty of Electrical Engineering

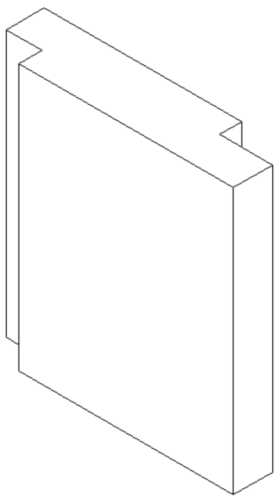


Figure 8.15: Faculty of Electrical Engineering

Another model that is used to test the adapted grid generation method is the building of the faculty of Electrical Engineering of Delft University of Technology. Unlike the cube model that is used in the previous graduation studies, the various edges of the Electrical Engineering building are not equal in length. The longest horizontal edge is about 71 m, the shortest edge 7 m. All vertical edges have a length equal to the height of the building, which is 88 m. For the cube, the length of all edges was 60 m. The size function that was used to control the growth in cell size from the cube's bottom to the edge of the circular ground face uses a start size for the cells of 0.1 times the height of the cube. Adapting this relation for the Electrical Engineering building gives a start size of 8.8 m for the cells. However, as the shortest horizontal edge is only 7 m, it is recommended to decrease the start size to the length of the shortest edge at maximum. This results in at least one cell at that edge. As the width of the building is 21 m, a startsize of 7 m results in three cells over the width of the building. This is probably too less to obtain reliable results. Therefore a start size of 3 m is applied, what results in 7 cells over the width of the building. In corners Gambit automatically adapts the shape and size of the cells if necessary to connect all cells on the model correctly. The growth in cell size is again restricted by attaching the mesh distribution on the circular edge of the cylinder, defined by the outside mesh, to the size function. The height of the cells is determined by the mesh distribution on the vertical edges. It is recommended to mesh all vertical edges using an interval size equal to the start size of the size function. This results in mesh elements of more or less the same length, width and height. It is also possible to mesh the vertical edges with an interval count. This divides the edge in a user-specified amount of intervals. If all vertical edges have the same length it is possible to create intervals of the desired size with this method, as long as the interval count is chosen correctly. However, if there are vertical edges of different length, it is not recommended to use an interval count to mesh the edges. This would result in various interval sizes on the different vertical edges, which is not preferred. For the Electrical Engineering building, all vertical side edges are meshed using an interval size equal to the start size of the size function. In Table 8.3 the variables that are used to mesh the cylinder are summarized.

Size function attached to the ground edges of the cube and the circular ground face of the bottom cylinder	Start size	3
	Growth rate	1.02
	Size limit	88
Side edges of the cube	Interval size	3

Table 8.3: Variables to mesh the lower cylinder containing the Electrical Engineering building

Figure 8.16 on the next page shows some results of the simulation. The left picture gives the pressure distribution on the building; the right picture gives the contours of velocity around the building.

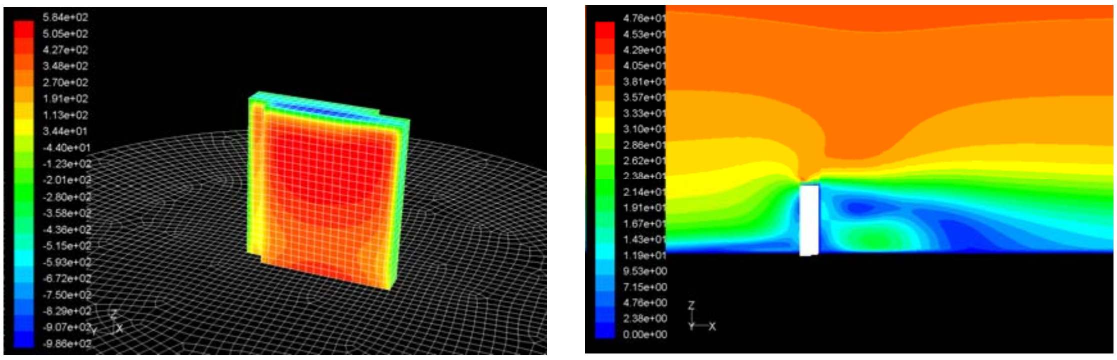


Figure 8.16: Pressure distribution and contours of velocity around the faculty's building

Some more tests with the adapted grid generation method showed that the method works for straight buildings where the ground and the roof have the same layout. The method starts with the attachment of a size function to the ground edges of the building model and the ground surface of the lower cylinder. It is recommended to use the shortest edge of the building model at maximum as the start size for the size function. The shortest edge of the model and its length are easy to find using a special tool in Gambit. After the size function is attached to the bottom of the cylinder, it is meshed using the pave scheme, resulting in an unstructured grid on the ground surface. The mesh distribution on the ground edges of the building model is then copied to the top edges of the model. The top surface is meshed using the pave scheme, creating an unstructured grid on the roof of the model. The next step is to mesh the vertical side edges. It is recommended to use an interval size in stead of an interval count, in order to create the same mesh distribution on vertical edges of different length. For meshing the vertical edges, it is recommended to use an interval size equal to the start size of the start function. Finally the lower and upper cylinders are meshed with hexahedral elements using the Cooper scheme. In Appendix N a detailed step by step guide is given of this first grid generation method.

Although the method works fine for the discussed geometry, the applicability is very limited. If the layout of the ground and the top of the building is not the same, the method does not work anymore. Linking the mesh distribution on the ground edges of the model to the top edges is not possible for such models. When the model is not straight because it contains balconies for example, the method also does not work. For the apartment building for example that was used to test the various design tools, it is not possible to use the developed grid generation method to mesh the model as the mesh on the ground edges can not be copied to the top edges. Another disadvantage of the method appears to be the use of hexahedral elements to mesh the cylinder. The cells have to connect perpendicular to the building model and the round edge of the cylinder. For complex building models or models containing several buildings, this is very difficult or even impossible to achieve with rectangular cells.

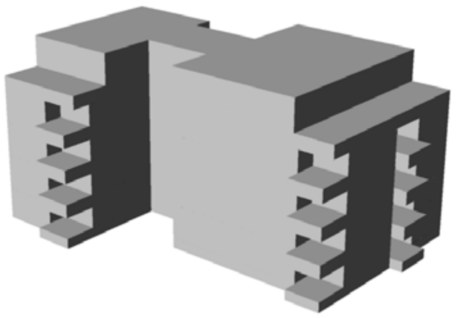


Figure 8.17: The first grid generation method does not work for complex building geometry

In the next paragraph another grid generation method is discussed that uses tetrahedral cells in stead of hexahedral cells. This method seems suitable to mesh the cylinder containing one or more complex building models.

8.4 Tetrahedral cells

In general, the advantage of using hexahedral meshes is that it requires fewer cells and usually gives a more accurate and earlier converged solution. However, the number of shapes that can be described with hexahedral cells is limited, what makes it unfit for meshing complex building models. Tetrahedral cells offer the most flexibility and theoretically it is even possible to mesh any shape with tetrahedral cells. A drawback of tetrahedrons is that it may require two or three times as many cells to get the same solution as with hexahedral cells. The large amount of tetrahedral cells is quite often too many for the available computer resources. The computer for example that is used for the calculations of this thesis has an Intel Pentium IV processor with 1.0 gB of RAM memory. If the total amount of cells remains under approximately 1.2 million, a reasonably calculation speed is achieved. When more cells are used, the amount of RAM becomes insufficient and data has to be written and read to and from the hard drive. This slows down the calculation speed considerably. As the Van Nalta domain without the inner cylinders already contains about 317.000 cells, only 883.000 cells remain to mesh the cylinder containing the research area. This is certainly too less to obtain very accurate results for complex models. However, with the current development of computer resources this problem will probably overcome in the nearby future.

Anticipating on these developments, the second grid generation method that is developed for this Master's thesis does use tetrahedral cells as they are still the most suitable type for meshing complex geometries and as they can be used for automated meshing. With automated meshing it is possible to mesh an entire area by only defining a mesh distribution on the edges of the models within the area. Gambit is capable of creating the total mesh for that area automatically then. However, this method does not always result in the best grid by definition and meshing the domain manually sometimes gives more accurate results. Nevertheless, meshing complex geometry by hand is very time-consuming when the model contains lots of edges and surfaces and errors are easy to make. Therefore automatic meshing with tetrahedral cells seems to be the most appropriate method that is universally applicable. At the moment it is not expected to obtain very accurate results from the method for complex building models, due to the limited computer resources. Nevertheless, as will be demonstrated in the rest of this chapter, it appears to be a potential method to generate a grid for complex building geometry.

The first step of this second grid generation method is to mesh all edges of the model inside the cylinder with a certain interval size. It is recommended to use an interval size that is equal to the length of the shortest edge of the model at maximum. The result is an equal mesh distribution on all edges of the model with at least one cell at the shortest edge. If the model contains only large edges or if there are enough resources available, it is of course recommended to use a smaller interval size. When all edges of the models are meshed, the cylinder can be meshed with tetrahedral elements using the TGrid scheme. The cylinder is automatically filled with a grid then. In Appendix N a detailed step by step guide is given of this second grid generation method.

In the next paragraphs some results of the grid generation method are discussed. The method is applied to the models of an apartment building and an office building, which were used to test the design tools, and to a model of the faculty of Architecture of Delft University of Technology. Due to the limited amount of cells, the results are probably not very accurate. However, the results must be seen as a proof of concept that the models can be meshed, simulations can be performed and output is obtained.

8.4.1 Apartment building

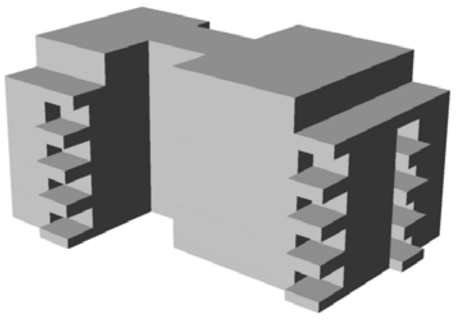
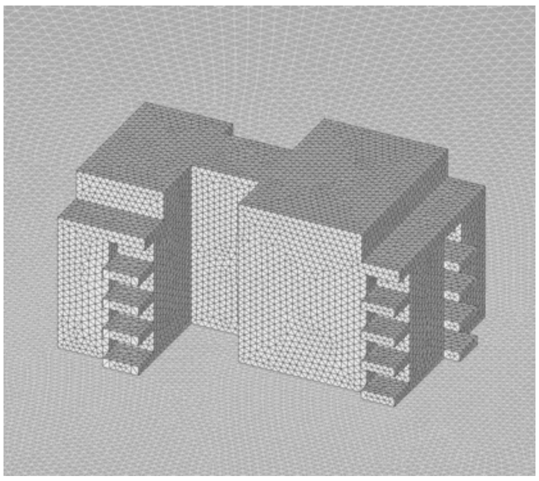


Figure 8.18: Apartment building model

The first model that is used to generate a grid with the second grid generation method is the relatively complex model of an apartment building. The model was first used to test the various developed design tools to create a wrapped surface around the model. This hollow surface model of the building is very suitable for CFD calculations as all original internal and external geometry is replaced. Only the surfaces that come in direct contact with the flow are modeled. Due to the balconies and divergent roof layout in comparison with the ground, it was not possible to use the first grid generation method as the ground and top edges can not be linked. With hexahedral elements it is also very difficult to mesh the complex geometry and to connect the cells perpendicular to the building model and the round edge of the cylinder. Using tetrahedral elements is the only way of meshing such complex building models.

The shortest edge of the building is 1.4 m. This length is used as interval size for meshing all edges of the building. Meshing the cylinder with tetrahedral elements results in 646.252 cells in the lower cylinder and 1.033.375 cells in the total computational domain. The mesh is far too coarse as the balconies for example just have two cells over the height. An accurate simulation of the flow pattern around the balconies is therefore not possible. However, as the maximum amount of cells of 1.2 million is almost reached, refinement of the mesh is not an option at the moment. In Figure 8.19 the mesh on the apartment building and the convergence of the residuals during the simulation are given.



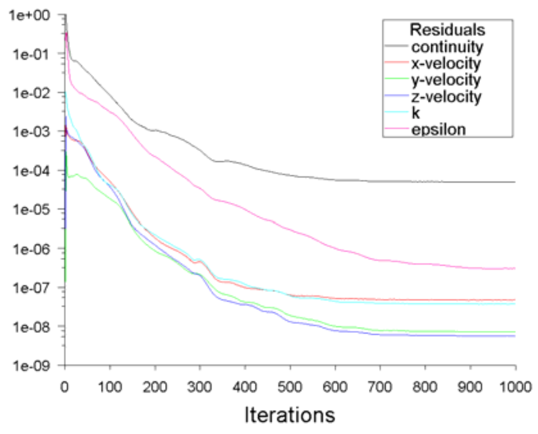


Figure 8.19: Mesh of the apartment building and convergence of the residuals

Looking at the path of the residuals it can be concluded that the residuals are converged. The variables have leveled out to some value and then stopped changing. As discussed in one of the previous paragraphs, monitoring the residuals is not the only way to judge the convergence of a solution. Also surface and force monitors can be applied to examine the convergence. Another option to evaluate the solution is to perform a grid convergence study. With this method the quality of the grid is judged. The method is discussed in Paragraph 8.7.2. As the maximum amount of cells that can be used is already reached for this model, such a study is not an option at the moment.

In Figure 8.20 an impression is given of the pressure distribution on the façades of the apartment building. Verifying the results is difficult as there is no reference information available for this model. For the moment, real wind tunnel studies will be the only option to verify the results of CFD calculations with such models.

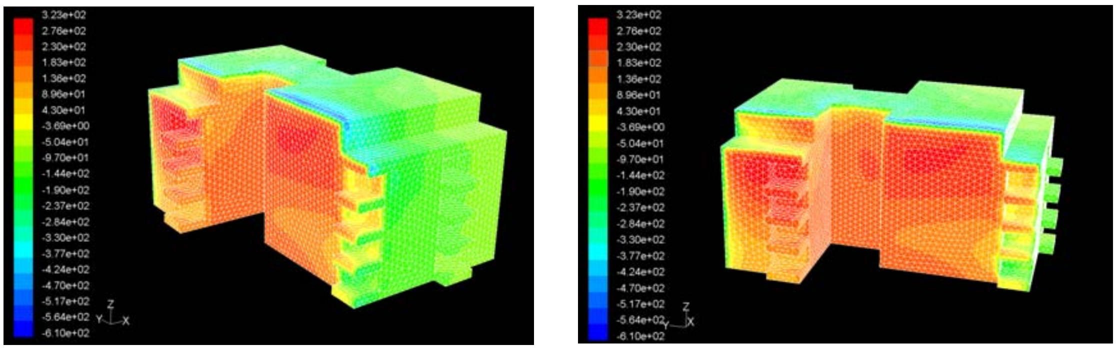


Figure 8.20: Static pressure distribution on the façades of the apartment building

In Figure 8.21 a part of the flow pattern around the building is given. The global pattern is perhaps reasonably approximated. However, the pattern around the balconies for example is probably not very accurate as they are meshed with only two cells over the height. For an accurate description of the flow pattern at those locations a lot more cells are required.

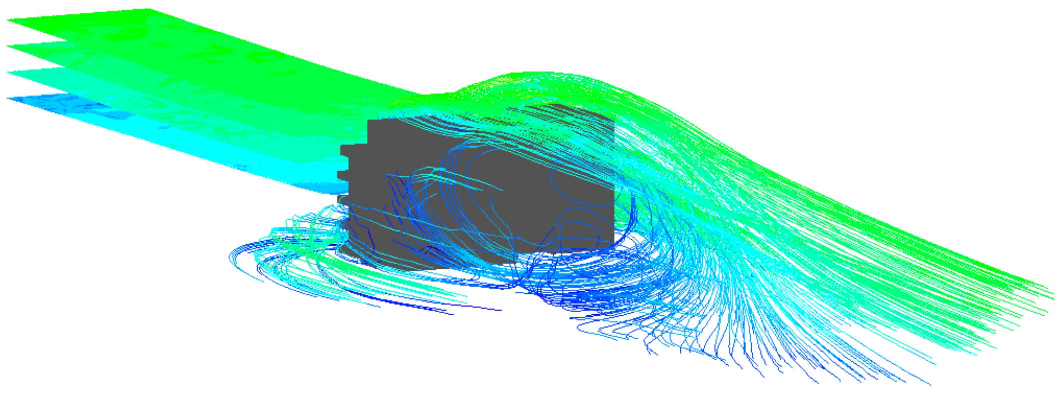


Figure 8.21: Flow pattern around the apartment building

8.4.2 Faculty of Architecture

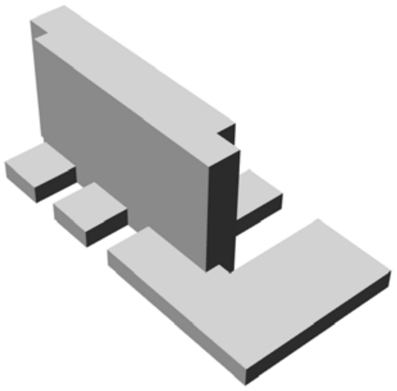


Figure 8.22: Faculty of Architecture

A second model that is used to generate a grid with the second grid generation method is the model of the faculty of Architecture of Delft University of Technology. The model is derived from the 3D model of the university district that was generated with the first design tools. The shortest edge of the model is 1.2 m. The edges are all meshed using an interval size of 1 m, resulting in 641.501 cells in the cylinder and 1.128.951 cells in the total domain. As the amount of 1.2 million cells is almost reached, further refinement is not useful. In Figure 8.23 the mesh on the faculty's building and the convergence of the residuals are given. As the residuals have leveled out to some value, it can be concluded that the residuals have converged. For this calculation, also surface monitors have been used to monitor the velocity magnitude above the roof of the main building and in the wake behind the building. In Figure 8.24 the convergence history of the monitored variables is given. The convergence of the residuals in combination with the surface monitors provides a much better idea of the convergence of a solution. As all variables have leveled out to a certain value, the solution is considered to be fully converged.

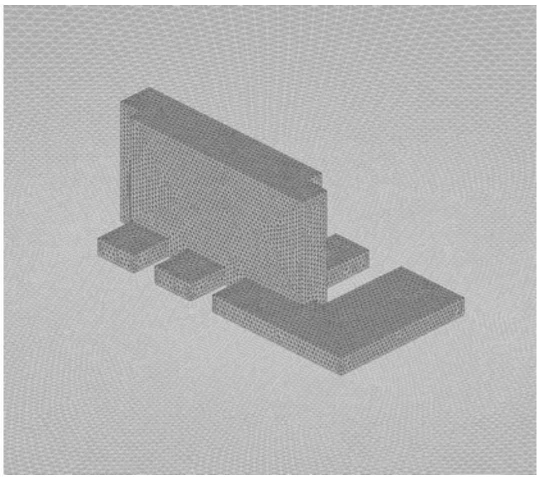
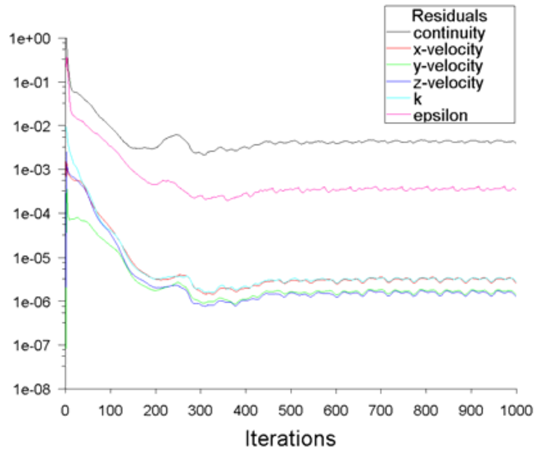
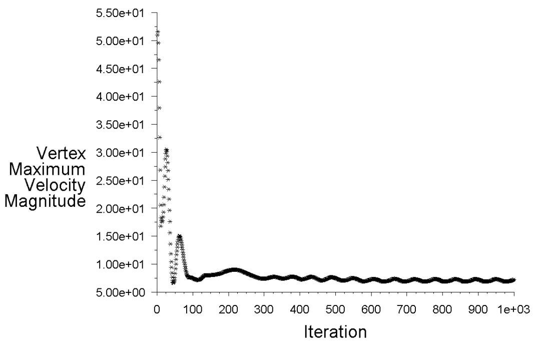



Figure 8.23: Mesh on the faculty of Architecture and the convergence path of the residuals



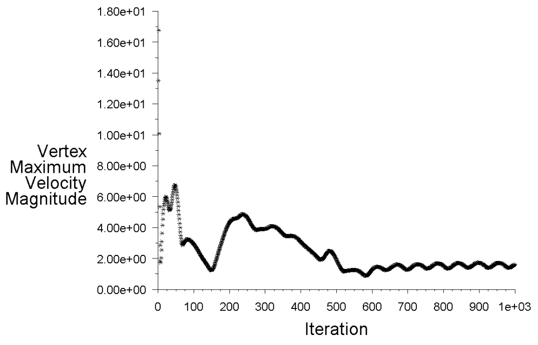


Figure 8.24: Surface monitor above the roof (left) and in the wake (right): Velocity Magnitude

In Figure 8.25 some results of the simulation are presented in a visual form. In the left picture the static pressure distribution on the façades of the building are given. The right picture gives the velocity vectors around the centre of the building. With velocity vectors a clear visualization of the flow pattern around the building can be provided. The vectors are colored by their velocity magnitude.

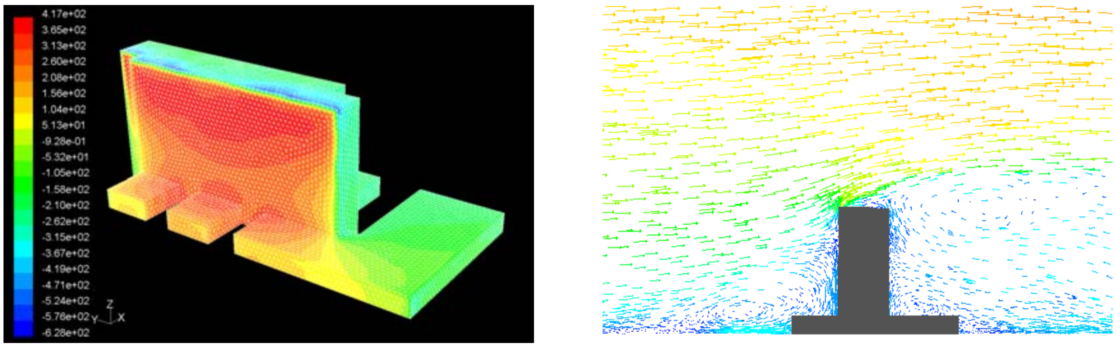


Figure 8.25: Static pressure distribution and velocity vectors around the centre of the building

In Figure 8.26 the flow pattern around the building is given. The pathlines are released at two different height levels; at 10 m and at 20 m above ground level. In this way the flow pattern around and above the lower parts of the faculty's building can be clearly visualized.

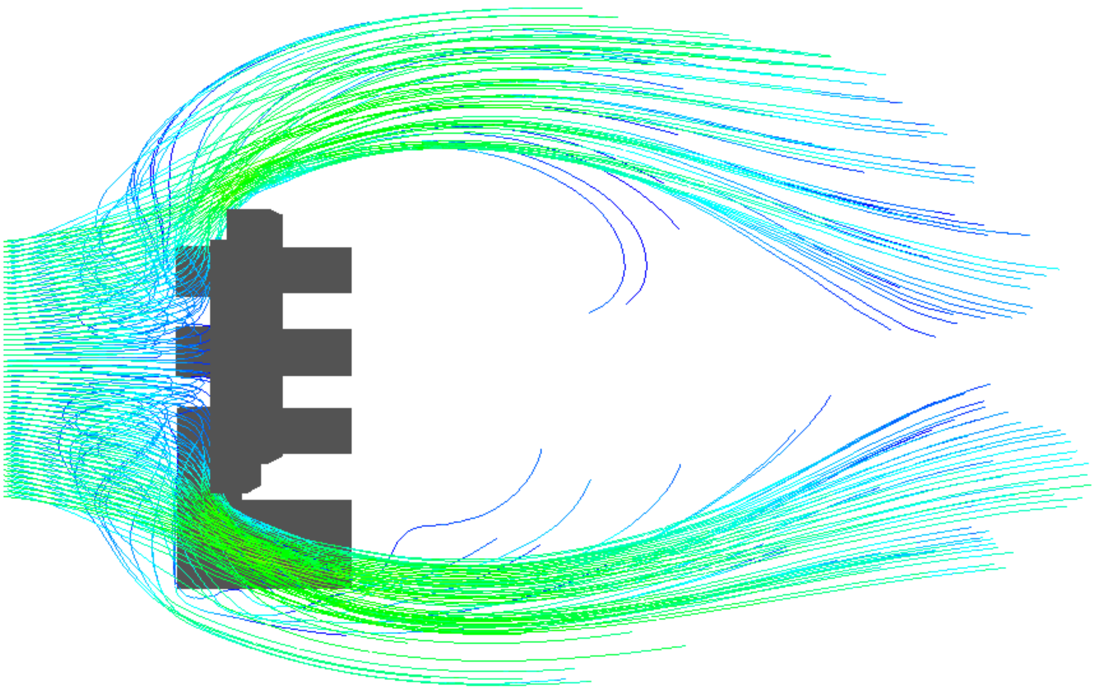


Figure 8.26: Pathlines show the flow pattern around the faculty of Architecture

8.4.3 Office building

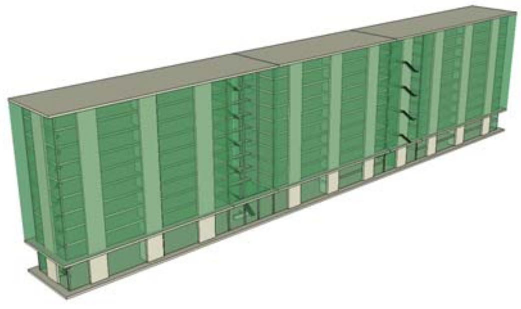
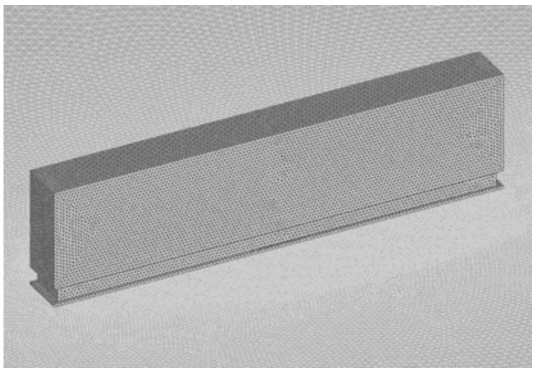


Figure 8.27: Model of the office building

Another building model that is used to test the second grid generation method is the model of an office building. This model is also used to create a wrapped surface around the model with the various developed design tools. The shortest edge of the hollow surface model is 1.5 m. Using an interval size of 1 m to mesh the edges appeared to be the minimum, as this size resulted in 709.391 cells in the cylinder and 1.116.989 cells in the total domain. Further refinement is not useful as the amount of 1.2 million cells is almost reached. In Figure 8.28 the mesh on the office building and the convergence of the residuals are given. As the variables have more or less leveled out to some value, it can be concluded that the residuals have converged. However, the paths are not very smooth and better results are possible. Refining the grid further will certainly give a more converged solution.



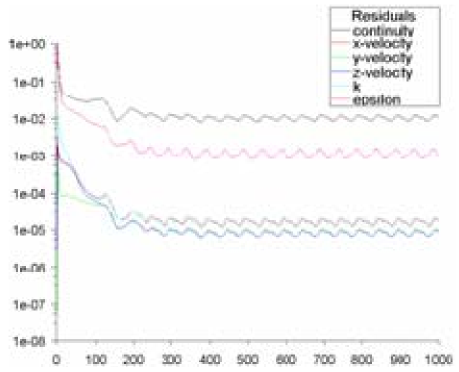
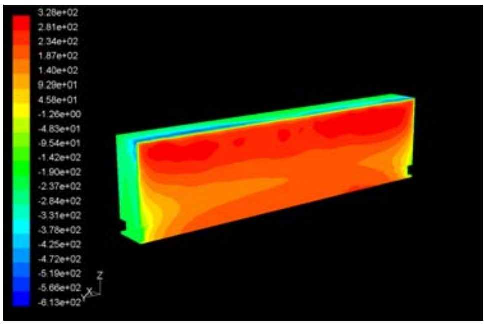


Figure 8.28: Mesh of the office building and the convergence path of the residuals

In Figure 8.29 the static pressure distribution on the façades of the office building and the flow pattern around the building are given. Due to the limited amount of available cells, the results are probably not very accurate. However, the simulations with the apartment building, the faculty of Architecture and the office building show that the grid generation method works and results can be obtained. This proves that the various developed design tools to setup the geometry work for CFD applications. With the discussed grid generation methods it is possible to mesh simple as well as complex building geometry. Especially for complex geometry the results are probably not accurate enough yet to determine the global wind loads on a building. However, taking the constant developments in computer resources into account, it will be possible in the nearby future to determine the wind loads on complex building geometry accurately.



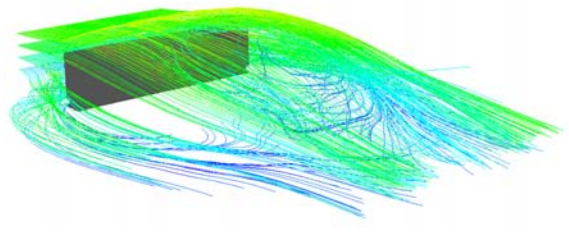


Figure 8.29: Static pressure distribution on the façades and flow pattern around the building

8.5 Part of the Delft University of Technology district

A final test for the developed grid generation method is the model of a part of the university district. For the single building models discussed in the previous paragraphs, the limited amount of cells that can be used with the available hardware resulted in a coarse grid on the models. As the model of the university district contains several buildings, the grid will be even coarser. To give a first impression of the flow pattern around the several buildings, the grid is perhaps suitable. However, to determine the flow pattern at specific locations accurately the grid is far too coarse. The goal of this test is just to show that the grid generation method also works for a model containing several buildings.

The model that is discussed in this paragraph is derived from the larger 3D model of the university district that was generated with the first design tools. For this case, only the buildings of the faculty of Electrical Engineering, Civil Engineering and Architecture and the sport centre are modeled. The model is manually adapted a little to remove some very short edges and to improve the ability to mesh the model. The building model of the faculty of Civil Engineering is also adapted. In the GIS datasets, the total building was presented with only one polygon. Extruding the polygon gave a single model with the same height everywhere. However, in reality the building consists of several parts with their own height. The original polygon is therefore manually split in several polygons that represent the various building parts. Extruding these polygons over various heights gives a building model that represents reality much better. The model of the research area is given in Figure 8.30. The radius of the total area is 500 m, which is quite large in comparison with the smallest area that would fit around the building models. The radius of this smallest area is 250 m and the design tools to generate a 3D model of the environment also give this radius as output. However, to provide a smooth transition between the mesh on the building models and the edge of the cylinder, a certain distance between the buildings and the cylinder edge is required. The radius of the actual research area is for this example two times the radius of the original area. The computational domain is therefore also generated with a radius of 500 m for the central cylinder.

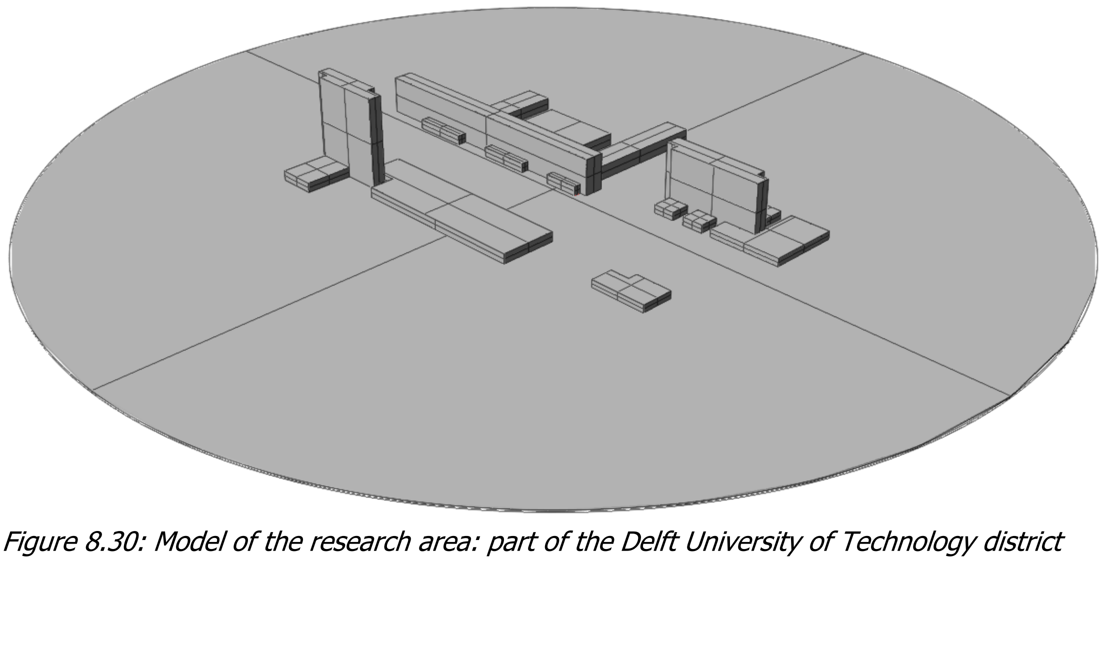


Figure 8.30: Model of the research area: part of the Delft University of Technology district

The shortest edge of the model is 7.4 m. Using an interval size of 3.7 m to mesh the edges of all buildings models results in at least 2 cells at the shortest edge. After meshing the cylinder with tetrahedral elements, the amount of cells in the lower cylinder is 620.112. The total amount of cells in the computational domain is 1.009.571. Further refinement is therefore not very valuable. In Figure 8.31 the mesh on the several building models and a part of the mesh on the ground surface is given. Although the grid is considerably coarse, it can be concluded that the grid generation method also works for models with several buildings.

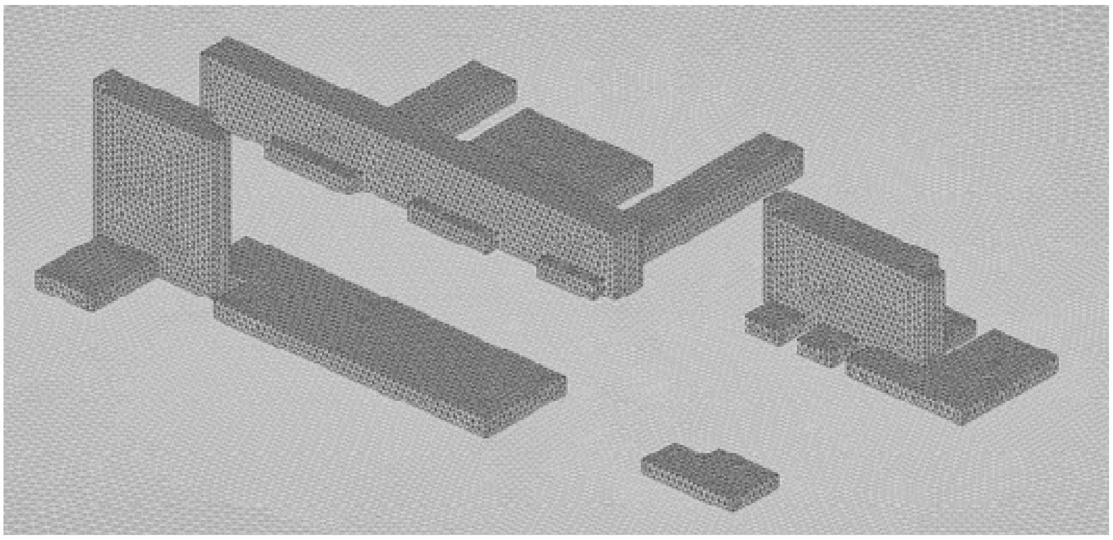


Figure 8.31: Mesh on the several building models and the ground surface

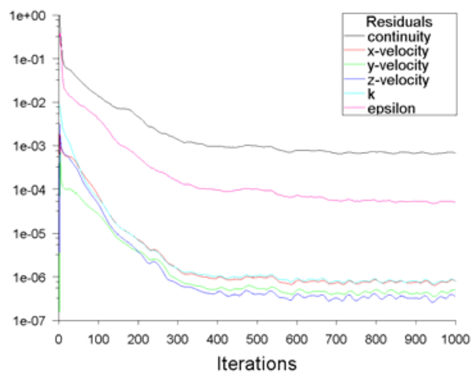


Figure 8.32: Convergence of the residuals

In Figure 8.32 the convergence of the residuals is given. As the residuals have more or less leveled out to some value, it can be concluded they have converged. Force and surface monitors are not applied. In Figure 8.33 the static pressure distribution on the façades of the building model are given. It is difficult to judge the accuracy of the solution, as the grid is reasonably coarse. However, the results look physically reasonable. At the façade of the Civil Engineering building right behind the high-rise of the faculty of Electrical Engineering for example, less pressure is measured.

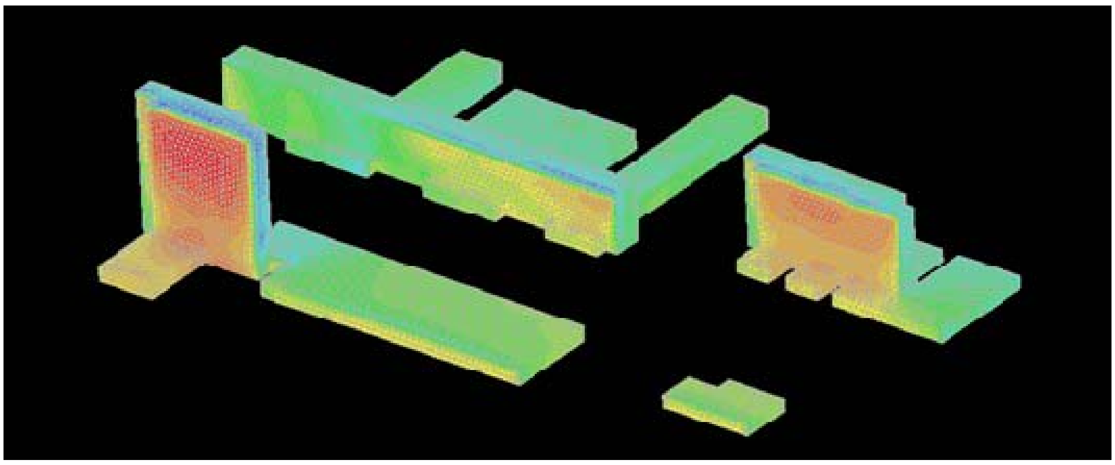


Figure 8.33: Static pressure distribution on the façades of the various building models

In Figure 8.34 the flow pattern around the several buildings is represented with pathlines. The particles are released at 150 m in front of the research area and 5 m above ground level. The pathlines are colored by the velocity magnitude of the particles. The blue color indicates a low velocity; green and yellow indicate a higher velocity. In the figure it is nicely visualized how the air is forced between the faculty of Civil Engineering and the faculty of Architecture; these are the top buildings in the model. The color of the lines show how the velocity increases as the air is squeezed between the buildings. This pattern differs somewhat from reality as no vegetation is modeled. However, the results look reasonable.

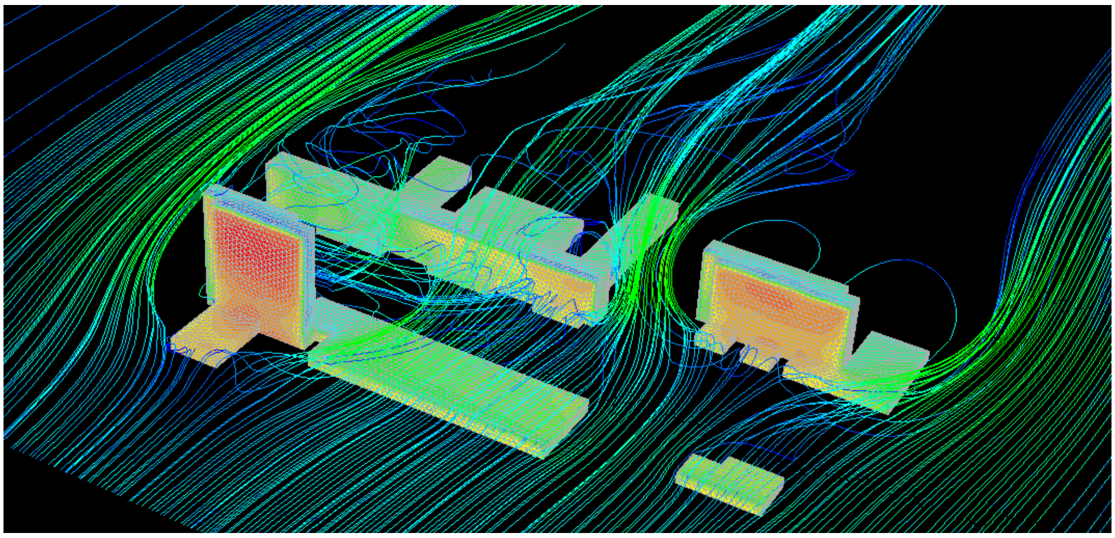


Figure 8.34: Pathlines colored by velocity magnitude; front view

In Figure 8.35 the pathlines are given from another point of view. The large wake behind the faculty of Architecture is nicely visible. Behind the high-rise building of Electrical Engineering, a large whirl arises as the air collides with the faculty of Civil Engineering. Pathlines are a very powerful tool to visualize such phenomena. However, one must keep in mind that only the pathlines that are released at 5 m above ground level are visualized. For different heights the flow pattern will differ and divergent phenomena might occur at different heights.

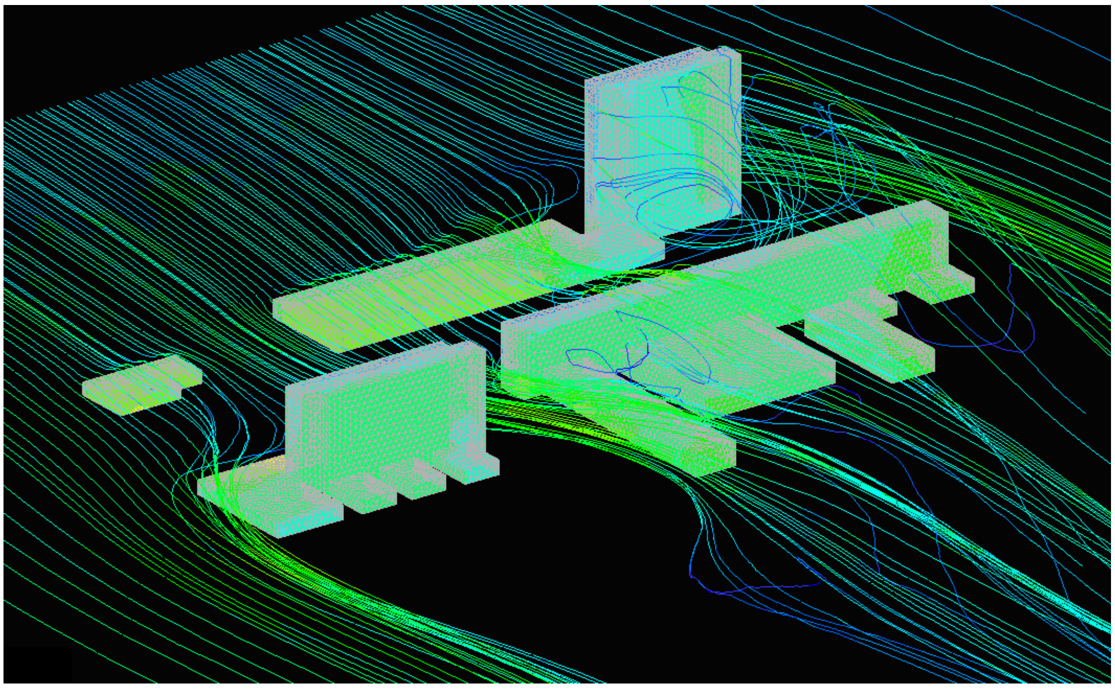


Figure 8.35: Pathlines colored by velocity magnitude; back view

The flow pattern in the research area can also be visualized using Tecplot, another CFD post-processor. The streamtraces in Figure 8.36 are all released at 150 m in front of the research area, but at different heights. Using Tecplot it is also possible to make animations of the flow pattern, what improves the insight in the pattern further. Because the research area was meshed with a considerably coarse grid, the quantitative pressure distribution for example is not very accurate. However, the streamtraces and pathlines probably give a reasonable indication of the flow pattern in the research area. These results can for example be used to determine the most suitable location for a certain wind turbine or to find a sheltered place for a terrace or tram stop.

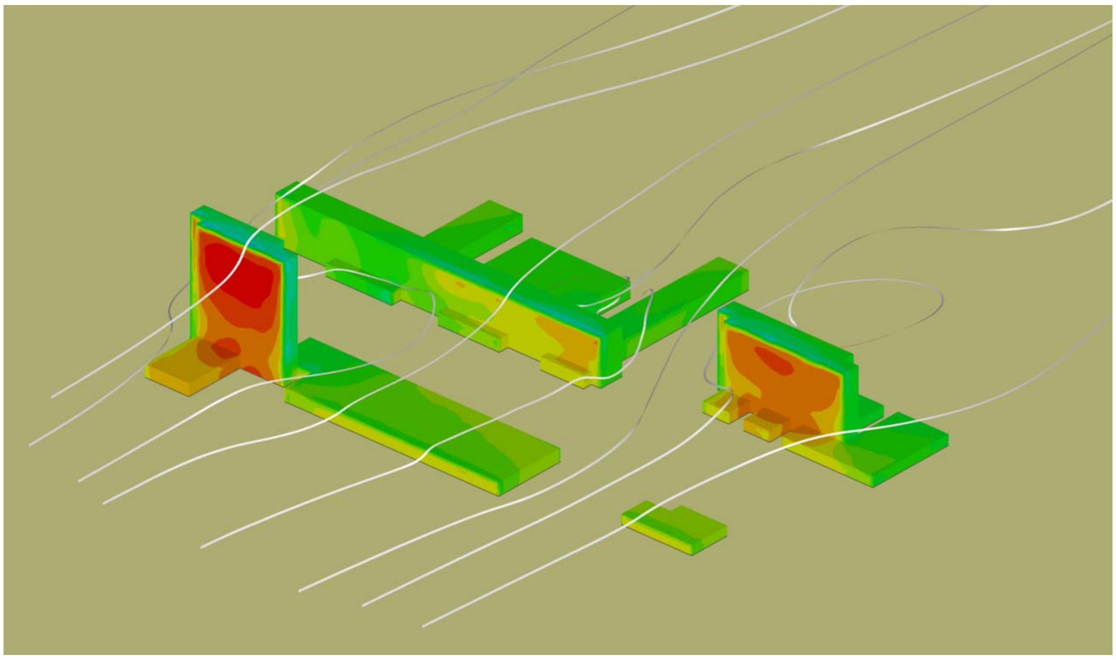
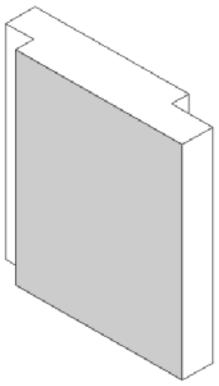


Figure 8.36: Streamtraces visualize the flow pattern in the research area

8.6 Comparison with the building codes

The results of the CFD calculations that are discussed in this chapter have demonstrated that the design tools work for CFD applications. With the developed grid generation methods the models can be meshed and results can be obtained. As the amount of cells that can be used to mesh the domain is limited, the grid is too coarse to simulate the flow pattern through the research area accurately. However, for a first indication of the global wind loads on a building, the coarse grid is perhaps sufficient. Comparing the results of a CFD calculation for a simple case with the building codes would therefore be very valuable.

The building model of the faculty of Electrical Engineering of Delft University of Technology is selected for the comparison, as the building has a simple, rectangular shape. With both the Dutch building code NEN 6702 and the Eurocode it should be possible to determine the wind load on this shape. Figure 8.37 gives the façade of the Electrical Engineering building on which the wind load will be calculated. The dimensions of the façade are b x h = 71 x 88 m^2.



Façade dimensions: b x h = 71 x 88 m^2

Figure 8.37: Façade of concern for the calculation of the wind load

8.6.1 Calculation of the wind load according to the Dutch building code

According to NEN 6702 [18] the wind loads on a structure or building can be calculated from:

$$p_{rep} = p_w \cdot C_{dim} \cdot C_{index} \cdot \phi_1 \tag{8.2}$$

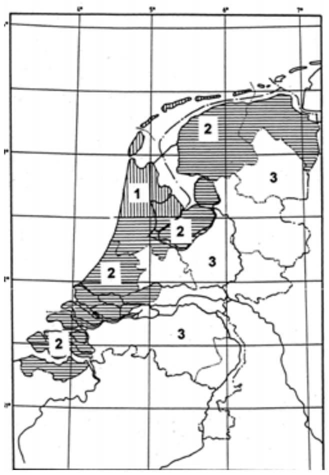
$$F_{index} = A \cdot p_{rep} \tag{8.3}$$

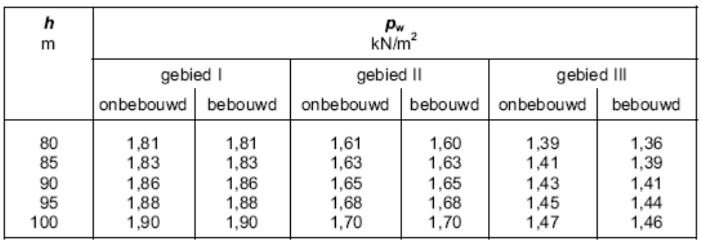
Where:	p_{rep}:	representative wind pressure that follows from the most unfavourable combination of forces that occur at the same time
	p_w:	extreme wind pressure, depending on the global and local wind climate
	C_{dim}:	relation between wind gusts and building dimensions
	C_{index}:	form factor for pressure, suction and friction
	Φ_1:	dynamic factor
	F_{index}:	representative wind force on a surface
	A:	surface exposed to the representative wind pressure

The determination of the various factors is given on the next pages.

Extreme wind pressure

The Netherlands are divided in three wind areas based on geographical differences, with a representative, constant wind velocity. For each area the code gives an extreme wind pressure p_w for built and unbuilt regions as a function of the height above the terrain. Figure 8.38 gives the three wind areas in which the Netherlands are divided with the extreme wind pressure p_w near the height of the Electrical Engineering building. As the building has a height of 88 m and is located in the built region of area II, the extreme wind pressure is 1,64 kN/m^2.





h m	p_w kN/m^2					
	gebied I		gebied II		gebied III	
	onbebouwd	bebouwd	onbebouwd	bebouwd	onbebouwd	bebouwd
80	1,81	1,81	1,61	1,60	1,39	1,36
85	1,83	1,83	1,63	1,63	1,41	1,39
90	1,86	1,86	1,65	1,65	1,43	1,41
95	1,88	1,88	1,68	1,68	1,45	1,44
100	1,90	1,90	1,70	1,70	1,47	1,46

Figure 8.38: The three wind areas with their representative wind pressures (NNI, [18])

According to NEN 6702 [18], the extreme wind pressure p_w can also be calculated with:

$$p_w = \left[1 + 7 \cdot \frac{k}{\ln\left(\frac{z - d_w}{z_0}\right)} \right] \cdot \frac{1}{2} \cdot \rho \cdot \left[2,5 \cdot u^* \cdot \ln\left(\frac{z - d_w}{z_0}\right) \right]^2 \qquad (8.4)$$

Where:	u^*:	friction velocity
	z_0:	roughness height of the terrain
	z:	reference height
	d_w:	displacement length
	k:	specific factor
	ρ:	air density

In Table 8.4 the characteristics to determine the extreme wind pressure are summarized for the three wind areas. For a building height of 88 m the wind pressure p_w is:

$$p_w = \left[1 + 7 \cdot \frac{0,9}{4,79} \right] \cdot \frac{1}{2} \cdot 1,225 \cdot \left[2,5 \cdot 2,82 \cdot \ln\left(\frac{88 - 3,5}{0,7}\right) \right]^2 = 1,64 \text{ kN/m}^2 \qquad (8.5)$$

	unbuilt			built		
	I	II	III	I	II	III
u^*	2.25	2.3	2.25	3.08	2.82	2.6
z_0	0.1	0.2	0.3	0.7	0.7	0.7
d	0	0	0	3.5	3.5	3.5
k	1.0	1.0	1.0	0,9	0,9	0,9

Table 8.4: Characteristics for the wind climate according to NEN 6702 [18]

Dimension factor

The factor C_{dim} is a factor that takes the relation between wind gusts and the dimensions of a building into account. The factor is introduced as large surfaces do not experience the effects of wind gusts at the same time everywhere. For buildings with a rectangular projection to the wind direction, the dimension factor can be calculated from:

$$C_{dim} = \frac{1+7\cdot I(h)\cdot\sqrt{B}}{1+7\cdot I(h)} \leq 1 \tag{8.6}$$

$$I(h) = \frac{1}{\ln\left(\frac{h}{0,2}\right)} \tag{8.7}$$

$$B = \frac{1}{0,94+0,021\cdot h^{2/3}+0,029\cdot b^{2/3}} \tag{8.8}$$

Where:	I(h):	turbulence intensity at height h
	B:	effective width
	h:	height of the building part where the wind acts upon
	b:	width of the building part where the wind acts upon

For the turbulence intensity and the effective width of the building it follows:

$$I(h) = \frac{1}{\ln(440)} = 0,16 \quad ; \quad B = \frac{1}{0,94+0,021\cdot 88^{2/3}+0,029\cdot 71^{2/3}} = 0,54 \tag{8.9}$$

For the dimension factor it follows:

$$C_{dim} = \frac{1+7\cdot I(h)\cdot\sqrt{B}}{1+7\cdot I(h)} = \frac{1+7\cdot 0,16\cdot\sqrt{0,54}}{1+7\cdot 0,16} = 0,86 \tag{8.10}$$

Form factor for pressure, friction and suction

For façades of buildings with a rectangular floor plan, the form factor C_{index} follows from Figure 8.39. For pressure the form factor is 0,8.

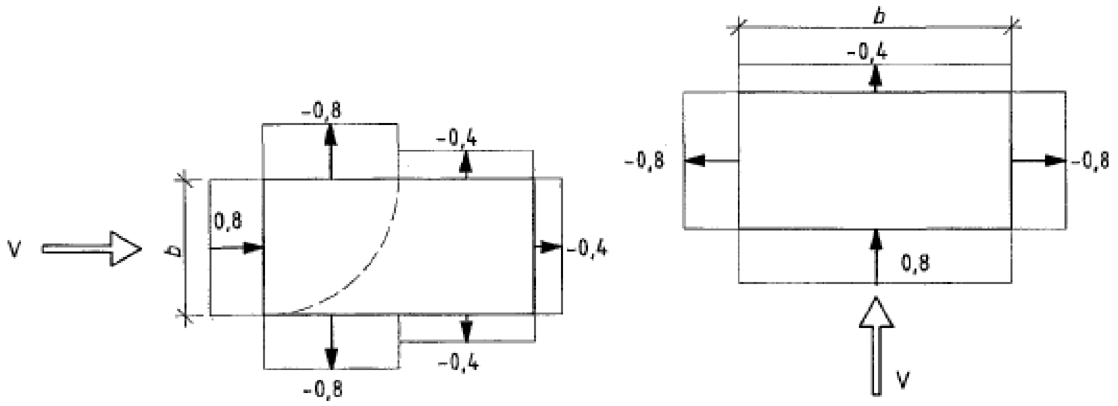


Figure 8.39: Form factors for façades of buildings with a rectangular floor plan (NNI, [18])

Dynamic factor
The effects of vibrations of the structure are taken into account with the dynamic factor Φ_1. As dynamic behaviour is not considered in this thesis, a factor of 1 can be used.

Representative wind pressure
With the derived variables it follows for the representative wind pressure:

$$p_{rep} = p_w \cdot C_{dim} \cdot C_{index} \cdot \phi_1 = 1,64 \cdot 0,86 \cdot 0,8 \cdot 1 = 1,13 \text{ kN/m}^2 \tag{8.11}$$

Reference area
The reference area of the façade of the Electrical Engineering building that is subjected to wind is:

$$A = b \cdot h = 71 \cdot 88 = 6.284 \; m^2 \tag{8.12}$$

Representative wind force
The representative wind force on the concerning façade can finally be calculated from:

$$F_{index} = A \cdot p_{rep} = 6.284 \cdot 1,13 = 7.101 \text{ kN} \tag{8.13}$$

8.6.2 Calculation of the wind load according to the Eurocode

In Paragraph 3.2.2 the process to calculate the wind load according to the Eurocode is summarized. This procedure will now be used to determine the forces on the building of the faculty of Electrical Engineering. The Eurocode prescribes the wind load with the following equation:

$$F_w = q_{ref} \cdot c_e(z_e) \cdot c_d \cdot c_f \cdot A_{ref} \tag{8.14}$$

Where:	q_{ref}:	The reference mean velocity pressure
	c_e:	The exposure coefficient. This coefficient takes the wind profile into account
	z_e:	The reference height. Usually this is the height of the building or structure
	c_d:	This coefficient takes the dynamic effects of the response of the building or structure into account
	c_f:	The force coefficient. This coefficient takes the geometry of the building or structure into account
	A_{ref}:	The reference area of the building or structure that is subjected to the wind

Reference mean velocity pressure
In the CFD calculations the logarithmic velocity profile that is prescribed in the Dutch building code is used, which is characteristic for the wind climate in a built area in Zuid-Holland. For a proper comparison, the reference wind velocity in the Eurocode calculation is therefore derived from the characteristics defined in the Dutch building code. The logarithmic velocity profile is given by:

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z-d}{z_0}\right) \tag{8.15}$$

Where:	u^*:	friction velocity
	κ:	Von Karman constant, which is 0.42
	z:	reference height
	z_0:	roughness height of the terrain
	d:	displacement length

In Table 8.4 the characteristics of the wind climate for three wind areas were summarized. As the building of the faculty of Electrical Engineering is located in a built region of area II, the reference wind velocity can be calculated from:

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z-d}{z_0}\right) = \frac{2,82}{0,42} \ln\left(\frac{88-3,5}{0,7}\right) = 32,18 \text{ m/s} \tag{8.16}$$

With an air density of 1,225 kg/m^3, the reference mean velocity pressure follows from:

$$q_{ref} = \frac{\rho}{2} \cdot \left(U(z)\right)^2 = \frac{1,225}{2} \cdot 32,18^2 = 634,45 \text{ N/m}^2 \tag{8.17}$$

Exposure coefficient
The exposure coefficient takes the wind profile into account and is defined by the turbulence intensity, a roughness coefficient and a topography coefficient. The exposure coefficient can be calculated from:

$$c_e(z_e) = c_r^2(z_e)c_t^2(z_e)\left[1+2gI_v(z_e)\right] \tag{8.18}$$

Where:		
	g:	the peak factor with value 3,5
	$I_v(z_e)$:	the turbulence intensity
	$c_r(z_e)$:	a roughness coefficient
	$c_t(z_e)$:	a topography coefficient

The topography coefficient c_t takes the increase of the mean wind velocity over isolated hills and cliffs into account. As there are no cliffs or hills in the research area, the coefficient is 1. The roughness coefficient takes the roughness of the terrain into account and is given by:

$$c_r(z_e) = k_T \cdot \ln\frac{z_{min}}{z_0} \tag{8.19}$$

Where:		
	k_T:	terrain factor
	z_{min}:	minimum reference height
	z_0:	roughness length

The terrain factor can be determined with:

$$k_T = 0{,}19\cdot\left(\frac{z_0}{z_{0,\mathrm{II}}}\right)^{0{,}07} = 0{,}19\cdot\left(\frac{0{,}7}{0{,}05}\right)^{0{,}07} = 0{,}23 \tag{8.20}$$

Where:		
	z_0:	roughness length
	$z_{0,\mathrm{II}}$:	the roughness length for terrain category II, which is 0,05

The minimum reference height z_{min} depends on the terrain category and is 10 for built regions of area II. The roughness coefficient can now be calculated from:

$$c_r(z_e) = k_T \cdot \ln\frac{z_{min}}{z_0} = 0{,}23\cdot\ln\frac{10}{0{,}7} = 0{,}61 \tag{8.21}$$

The final variable that defines the exposure coefficient is the turbulence intensity. The turbulence intensity is defined as the ratio between the maximum or minimum wind speed and the mean wind speed at a certain height. It can be calculated from:

$$I_v(z) = \frac{k_T}{c_r(z)\cdot c_t(z)} = \frac{0{,}23}{0{,}61\cdot 1} = 0{,}38 \tag{8.22}$$

From the calculated variables the exposure coefficient follows from:

$$c_e(z_e) = c_r^2(z_e)c_t^2(z_e)\left[1+2gI_v(z_e)\right] = 0{,}61^2\cdot 1^2\cdot\left[1+2\cdot 3{,}5\cdot 0{,}38\right] = 1{,}35 \tag{8.23}$$

Dynamic factor
The effects of vibrations of the structure are taken into account with the dynamic factor c_d. As dynamic behaviour is not considered in this thesis, the factor is 1.

Force coefficient
The force coefficient takes the geometry of a building into account. It is defined by:

$$c_f = c_{f,0} \cdot \psi_r \cdot \psi_\lambda \tag{8.24}$$

Where:	$c_{f,0}$:	the force coefficient of rectangular sections with sharp corners
	ψ_r:	the reduction factor for square sections with rounded corners
	ψ_λ:	the end-effect factor for elements with free-end flow

The various parameters are based on results of wind tunnel tests. The force coefficient $c_{f,0}$ depends on the width and depth of the building. For rectangular buildings the coefficient can be derived from Figure 8.40. The width b of the façade of the Electrical Engineering building is 71 m; the depth d of the building is 21 m. The ratio between the depth and width is 0.296. This gives a force coefficient $c_{f,0}$ of 2,1.

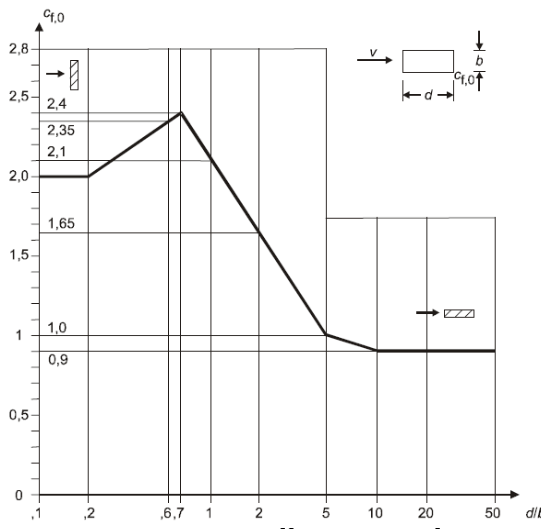


Figure 8.40: Force coefficients $c_{f,0}$ of a rectangular section (NNI, [17])

The reduction factor ψ_r is a coefficient that takes the effect of rounded corners into account. As the building has no rounded corners, the coefficient is 1. The end-effect factor ψ_λ takes the reduced resistance of the structure due to the wind flow around the end of the building into account. It depends on the slenderness of the structure and a solidity parameter and can be derived from Figure 8.41. The solidity takes the effect of openings in the structure into account, but is 1 for the closed building of the faculty of Electrical Engineering. According to the Eurocode, the effective slenderness for rectangular buildings higher than 50 m is:

$$\text{Minimum of } \left(\lambda = 1,4 \cdot \frac{l}{b} \ ; \lambda = 70 \right) \tag{8.25}$$

Figure 8.41 gives the variables l and b to determine the effective slenderness for wind normal to the plane of the picture. For the faculty of Electrical Engineering the effective slenderness is 1,4 x 88/71 = 1,735. The end-effect factor becomes 0,62.

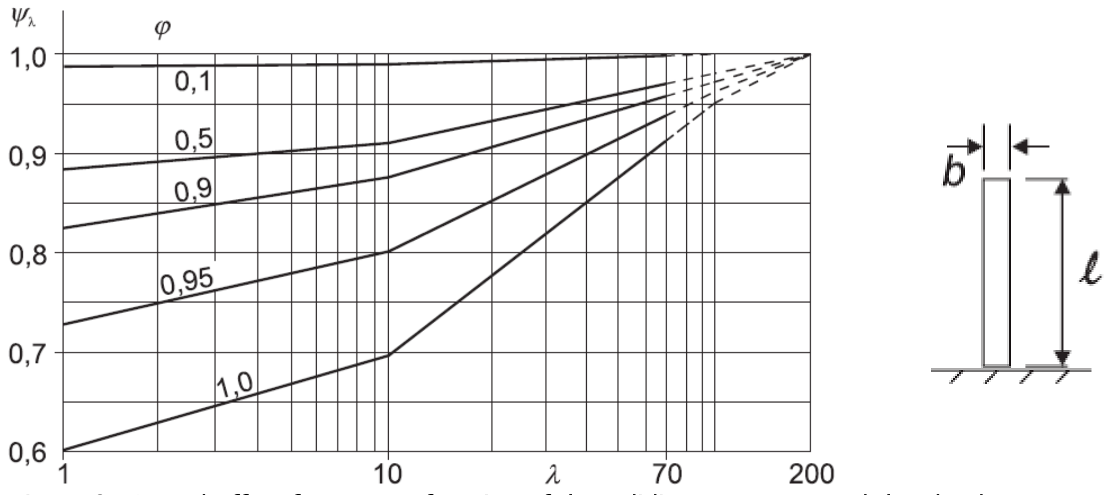


Figure 8.41: End-effect factor as a function of the solidity parameter and the slenderness (NNI, [17])

For the force coefficient it follows:

$$c_f = c_{f,0} \cdot \psi_r \cdot \psi_\lambda = 2,1 \cdot 1 \cdot 0,62 = 1,302 \qquad (8.26)$$

Reference area

The final variable that is required to calculate the wind load is the reference area of the building that is subjected to wind. For the faculty of Electrical Engineering, the reference area of the concerning façade can be determined with the following expression:

$$A_{ref} = b \cdot h = 71 \cdot 88 = 6.284\ m^2 \qquad (8.27)$$

Total wind load

From the calculated variables the total wind load according to the Eurocode on the façade of concern can finally be determined:

$$F_w = q_{ref} \cdot c_e(z_e) \cdot c_d \cdot c_f \cdot A_{ref} = 634,45 \cdot 1,35 \cdot 1 \cdot 1,302 \cdot 6.284 = 7.015.963\ N \qquad (8.28)$$

8.6.3 Wind load according to CFD calculations

To compare the wind load according to the building codes an additional CFD calculation is performed with the building of the faculty of Electrical Engineering. In stead of the calculation that is discussed in Paragraph 8.3, the building is now meshed with tetrahedral cells as this element type will be used most often in practice. The edges of the model are meshed with an interval size of 1, what resulted in approximately 1.1 million cells. Further refinement is not possible, but as the shortest edge of the building model was 7 m, the mesh is perhaps fine enough already for the determination of global wind loads. In the previous CFD calculations that are performed in this thesis, the boundary conditions were all taken from the conditions for the single cube. However, to simulate reality and to be able to make a fair comparison with the building codes, some boundary conditions have to be adapted.

Ground surfaces
For the CFD calculation with the single cube an unbuilt area was considered with a roughness height of z_0 = 0.2 m. However, the faculty of Electrical Engineering is located in an urban area and in the calculations according to the Eurocode a roughness height of z_0 = 1 m was used. Fluent uses two parameters to define the roughness of a boundary: the roughness length K_s and the roughness constant C_s. The parameters can be determined with:

Roughness length:	$K_s = 13 \cdot Z_0$ for non-uniform roughness
	$K_s = 20 \cdot Z_0$ for uniform roughness
Roughness constant:	$C_s = 0.75$ for non-uniform roughness
	$C_s = 0.50$ for uniform roughness

As the roughness of the ground surface is considered to be non-uniform, a roughness constant of 0.75 is attached to the ground surface in Fluent. With a roughness height of z_0 = 1 m, the roughness length K_s becomes 13. This differs considerably with the roughness length of 2.6 that was used for the calculations with the cube.

Building façades
For the calculations with the cube, the roughness of the walls was considered to be uniformly distributed and a roughness constant of C_s = 0.50 was attached to the sides of the cube. As the roughness of the façades of the faculty of Electrical Engineering can also be considered as uniformly distributed, the same value is attached to these façades. The Eurocode gives in addition friction coefficients for three façade types. These coefficients are 0.01 for smooth surfaces, 0.02 for rough surfaces and 0.04 for very rough surfaces. The sides of the cube were modeled as very rough surfaces, what resulted in a roughness length of 0.8 However, the façades of the faculty of Electrical Engineering can be considered as smooth. This gives a roughness length of 20 x 0.01 = 0.2.

Velocity profile
In the previous CFD calculations of this thesis a User-Defined Function (UDF) was used to add the logarithmic velocity profile to the calculations. In this function the characteristics of the profile were all defined for the unbuilt regions of area II. However, as the building of the faculty of Electrical Engineering is located in the built region of area II, some characteristics have to be changed in the function. Table 8.5 gives the original and adapted values of the characteristics that define the logarithmic velocity profile that is used in the calculations. The values originate from the Dutch building code NEN 6702 [18].

	Original values (unbuilt region)	Adapted values (built region)
Friction velocity u^*	2,3	2,82
Von Karman constant κ	0,42	0,42
Displacement length d	0	3,5
Roughness height z_0	0,2	0,7

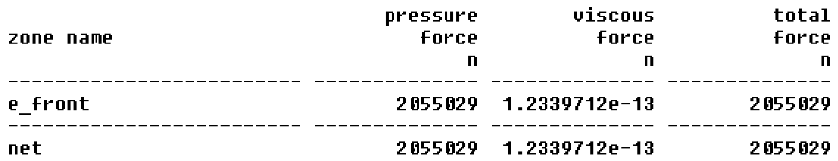
Table 8.5: Characteristics of the logarithmic velocity profile for the unbuilt and built area II

Results

After 1000 iterations the solution appeared to be fully converged, as the residuals had leveled out to some value and then stopped changing. To determine the forces on the concerning façade of the Electrical Engineering building, the *Force Reports* option in Fluent is used. This option allows the user to report the total forces or moments on a certain boundary of the model. Figure 8.42 gives the report of the forces on the concerning façade. The total force consists of a pressure force and a viscous force. The viscous force is the force due to the shear stress on the surface and is defined as:

$$\tau = viscosity \cdot \frac{du}{dy} \tag{8.29}$$

Herein is y the direction perpendicular to the surface and u the velocity parallel to the surface. As there is hardly any gradient du/dy on the surface perpendicular to the flow direction, the viscous force at the front façade of the building is nearly zero.



```
                             pressure        viscous          total
zone name                       force          force          force
                                    n              n              n
------------------------- -------------- -------------- --------------
e_front                       2055029   1.2339712e-13        2055029
------------------------- -------------- -------------- --------------
net                           2055029   1.2339712e-13        2055029
```

Figure 8.42: Force Report of the concerning façade of the faculty of Electrical Engineering

8.6.4 Comparison

The results of the various calculations are summarized in Table 8.6. According to the Dutch building code, the total pressure force on the concerning façade is 7.101 kN. Forces caused by suction or friction are not considered in the comparison. With the Eurocode, a total pressure force of 7.016 kN is calculated. The results of both codes are nearly the same. However, according to the CFD calculation the pressure force on the façade is only 2.055 kN, which differs a factor 3,5 with the building codes. Assuming that the calculations according to the building codes and the CFD calculation are performed correctly according to the various guidelines, the difference is quite large.

	Dutch building code	Eurocode	CFD results
Total force on the concerning façade	7.101 kN	7.016 kN	2.055 kN

Table 8.6: Total force according to the Dutch building code, Eurocode and CFD results

8.6.5 Conclusion

The results of the CFD calculation and the calculations according to the building codes differ considerably and it is very hard to give an explanation for the large difference. The results of the calculations according to the building codes are nearly the same and the values for the extreme wind pressure p_w, which are used in the Dutch code NEN 6702, are derived from real wind tunnel studies. Although the correction factors that are used in the building codes are on the safe side, a difference between the CFD calculation and the codes of factor 3,5 is too large to have full confidence in the CFD results. All CFD calculations that are performed in this thesis are setup according to the guidelines given by Van Nalta [16]. The parameters that describe the logarithmic velocity profile are all derived from the Dutch building code. It is therefore difficult to explain the large difference between both methods. When the results of a calculation with the 60 x 60 x 60 m^3 cube, performed by Snijders [23] in his Master's thesis, are analyzed, the Force Report gives a total force of 1.205 kN on the surface that is directed to the flow. According to the Dutch building code, a total force of 3.758 kN is to be expected on that surface. So, in the previous graduation studies the resulting forces already differed a factor 3 as well.

A first explanation of the large difference could be an incorrect setup of the CFD calculation. The parameters of the logarithmic velocity profile or other boundary conditions are perhaps not defined correctly. However, as two graduation studies are already performed on this topic, errors are not expected here. On the other hand, Van Nalta has validated the current approach to perform a CFD calculation by comparing the pressure coefficients on a cube as a result of real wind tunnel studies, with pressure coefficients on a cube that follow from CFD calculations. These coefficients were the only wind tunnel results available; forces for example are not compared. As pressure coefficients do not give information about the real pressure that acts on a surface, incorrect conclusions can be drawn from such a comparison. The pressure coefficients for the concerning surface of the cube can for example be the same for both the CFD calculation and the wind tunnel test. It is however possible that the real pressure that acts on the surface differs considerably. In further studies the real pressure distribution on simple models have to be compared between CFD calculations and real wind tunnel studies to exclude eventual errors in the current approach to perform a calculation.

It is also possible that the large difference between the CFD results and the building codes originate from the quality of the grid. Although the residuals were converged considerably after the calculation, the grid quality still can cause some errors. Although a grid convergence study can not be performed at the moment due to the actual computer resources, a grid independent solution is desired since it eliminates any errors that originate from the coarseness of a grid. However, refining the grid further is not an option as the maximum amount of cells was almost reached in the calculation.

Another explanation of the poor results could be the modeling of turbulence. At the moment the realizable *k-ε* model is the only possible way of calculating turbulence, due to the limited computer resources. Modeling the separation of air from the building with the *k-ε* model is difficult and an incorrect modeling of turbulence can have a significant influence on the accuracy of a solution. However, as the faculty of Electrical Engineering has straight corners and as there are hardly any viscous forces at the façade of concern, the poor turbulence model can not explain the large difference. Using other turbulence models is therefore not very valuable in this case. Only for rounded corners the *k-ε* model becomes inferior to model the air separation accurately and large differences in the results can occur then. The Reynolds stress model for example can deliver more accurate results. However, this model requires 50-60% more calculation time per iteration compared to the *k-ε* model (Nalta, [16]). In addition, 20% extra memory is required and more iterations are needed to obtain a converged solution. Nevertheless, really accurate results can only be obtained using Large Eddy Simulation. As the computational demand is very high for this modeling type, LES will just come into reach in the far future.

A final explanation of the large difference could be an imperfection of the building codes. In the CFD calculation the pressure distribution on the façade is determined by the logarithmic velocity profile. As the wind velocity is increasing with the height, the forces on the building near the ground will be smaller than at a larger height. In the building codes an extreme wind pressure is calculated for a certain height. As this peak pressure is just multiplied by the area of the surface of concern, it is considered to act everywhere on the surface. In Figure 8.43 the contours of static pressure on the façade of concern according to the CFD calculation are given. The pressure distribution differs considerably with the distribution according to the building codes. There are areas of low pressure and at the boundaries of the façade even suction occurs.

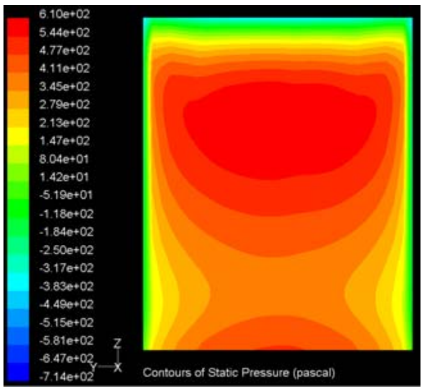


Figure 8.43: Static pressure contours on the concerning façade

In comparison with the building codes, the total force according to the CFD calculation that acts on the façade will be smaller due to the more realistic pressure distribution. A difference in the results of factor 3,5 is still quite large, but the current approach of the building codes with a poor pressure distribution and correction factors that are on the safe side will result in an overestimated total force on a structure. However, from the pressure distribution given in Figure 8.43, it can be derived that the maximum pressure on the façade is 610 N/m^2. Comparing this value with the extreme wind pressure of 1640 N/m^2 that is used in the calculation according to the Dutch building code NEN 6702, it can be concluded that a large difference occurs here as well. The most plausible explanation for the large difference between CFD and the building codes is therefore an incorrect approach to setup a CFD calculation.

In additional research lots of comparisons should be made between real wind tunnel studies and CFD calculations to judge the accuracy of the CFD results. When an explanation for the large difference between CFD and the building codes is found, the current approach of the Virtual Wind Tunnel can be validated. First comparisons should be made for simple models with both straight and rounded corners. In addition some more complex models could be used. Then comparisons should be made for a research area containing several simple shapes. Finally a more complex building that is placed in a built environment should be used to compare the results between wind tunnel studies and CFD calculations. If the accuracy of the CFD results is determined, the different methods to predict the wind loads can be judged. When in the end the CFD results are profitable in comparison with the codes, smaller forces can be obtained from the wind calculations. More efficient structures can be designed that will reduce the costs considerably. Especially for complex building models the Virtual Wind Tunnel could become a very valuable design application then.

8.7 Conclusions

8.7.1 General

In this chapter the results of some CFD calculations are discussed that were performed on several models which were generated with the various developed design tools. The goal of the calculations was to proof that the tools work for CFD applications and to demonstrate that calculations can be performed with the generated models. Two grid generation methods are developed that are able to mesh both simple and complex building geometry. Especially for meshing complex geometry, a lot of cells are required to obtain accurate results. However, the computer resources are still the limiting factor for the amount of cells that can be used. With the hardware used for this thesis it was therefore not possible to obtain very accurate results for complex building geometry. However, from the simulations discussed in this chapter it can be concluded that the developed design tools work for CFD applications, that the grid generation methods are able to mesh the various geometry and that results can be obtained.

The Van Nalta domain was used to perform the calculations, with the building models placed in the lower part of the central cylinder. As the domain itself was automatically meshed during the generation of the domain, only the cylinder containing the model of interest had to be meshed manually. Two grid generation methods are discussed to mesh the central cylinder. The first method is based on a grid generation method developed by Van Nalta and uses hexahedral cells to mesh the cylinder. This method can only be used to mesh simple geometry with the same layout from the bottom to the top of the building model. The second grid generation method uses tetrahedral elements and can be used to mesh more complex geometry. However, meshing a model with tetrahedrons requires a lot more cells, quite often too many for the available resources. At the moment it is therefore impossible to obtain very accurate results and it will be very hard to simulate the flow pattern around specific small building parts precisely. The comparison between a CFD calculation and the building codes for the faculty of Electrical Engineering showed a large difference between both methods. At the moment there is no sure explanation for the different results, but it is possibly caused by an incorrect approach to setup a CFD calculation. An analysis of the results of a calculation performed by Snijders [23] with the 60 x 60 x 60 m^3 cube showed a difference of factor 3 as well in comparison with the building codes. As the same approach is used for the calculations, it is plausible that the current approach to setup a CFD calculation is inferior. Further research is therefore required to validate the current approach by comparing the results from CFD calculations with results from real wind tunnel studies. From these comparisons a certain trend can perhaps be derived that can be used to outweigh the large difference between CFD and the building codes. If the CFD results are considered to be accurate finally, a major advantage of using CFD to determine the wind loads in stead of the building codes is that the CFD results give a lot more insight in the flow pattern. As the results can be presented in various visual forms, lots of phenomena can be investigated and critical places are easy to find.

8.7.2 Convergence

The convergence of a solution can be judged in several ways. One way is to monitor the convergence of the residuals during the iteration process. Other ways are using surface and force monitors to examine the convergence of specific quantities in the domain. When the monitored variables level out to a certain value, the solution is considered to be fully converged. The quality of the grid has a large influence on the convergence of a solution. One of the most reliable methods to investigate if a grid is appropriate or not is performing a grid convergence study. The method involves performing a simulation on two or more successively finer grids.

When a grid is refined, the cells become smaller and the number of cells in the flow domain increases. As the numerical error becomes smaller, the solution approaches the 'exact' solution. Suppose a model that is meshed with 50.000 cells for example. By refining the grid to 100.000 cells and 200.000 cells, the numerical error decreases for each refinement. The decrease in error after each refinement will be smaller as the grid becomes smaller. If the solution does not change considerably anymore, the solution is called the *grid independent solution*. If the computer resources allow such a study, a grid independent solution is desired in the end since it eliminates any errors that originate from the coarseness of a grid. The grid independent solution should be used for the comparison with real wind tunnel results to show what the error of the CFD calculation is. However, with the actual capacity of the desktop computers it is usually not possible to perform a grid convergence study as refinement of a grid is generally not an option.

8.7.3 Grid quality

The quality of the grid influences the results of a CFD calculation considerably. A poor quality can cause inaccurate solutions or a poor convergence. One option to judge the grid quality is to perform a grid convergence study. Another characteristic that defines the quality of a grid can be derived in Gambit as well.

Skewness
Skewness is defined as the difference between the shape of a cell and the shape of an equilateral cell of equivalent volume (Internet, [9]). Highly skewed cells should be avoided, since it negatively affects the accuracy and convergence of a solution. For triangular cells the angles between the cell edges should be close to 60 degrees and never larger than 90 degrees. Figure 8.44 gives an example of a triangular cell with a low and high skewness.

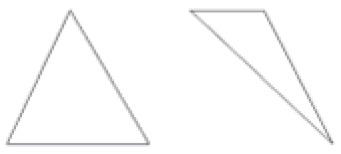


Figure 8.44: Cells with low skewness (left) and high skewness (right); (Internet, [9])

The skewness of a cell can be calculated from:

$$\text{Skewness} = \max\left[\frac{\theta_{\max}-\theta_e}{180-\theta_e}, \frac{\theta_e-\theta_{\min}}{\theta_e}\right] \quad (8.30)$$

Where:	θ_{max}:	largest angle in cell
	θ_{min}:	smallest angle in cell
	θ_e:	angle for equiangular cell (triangle: 60 degrees; square: 90 degrees)

A general guideline for judging a cell's quality based on the skewness is given in Table 8.7.

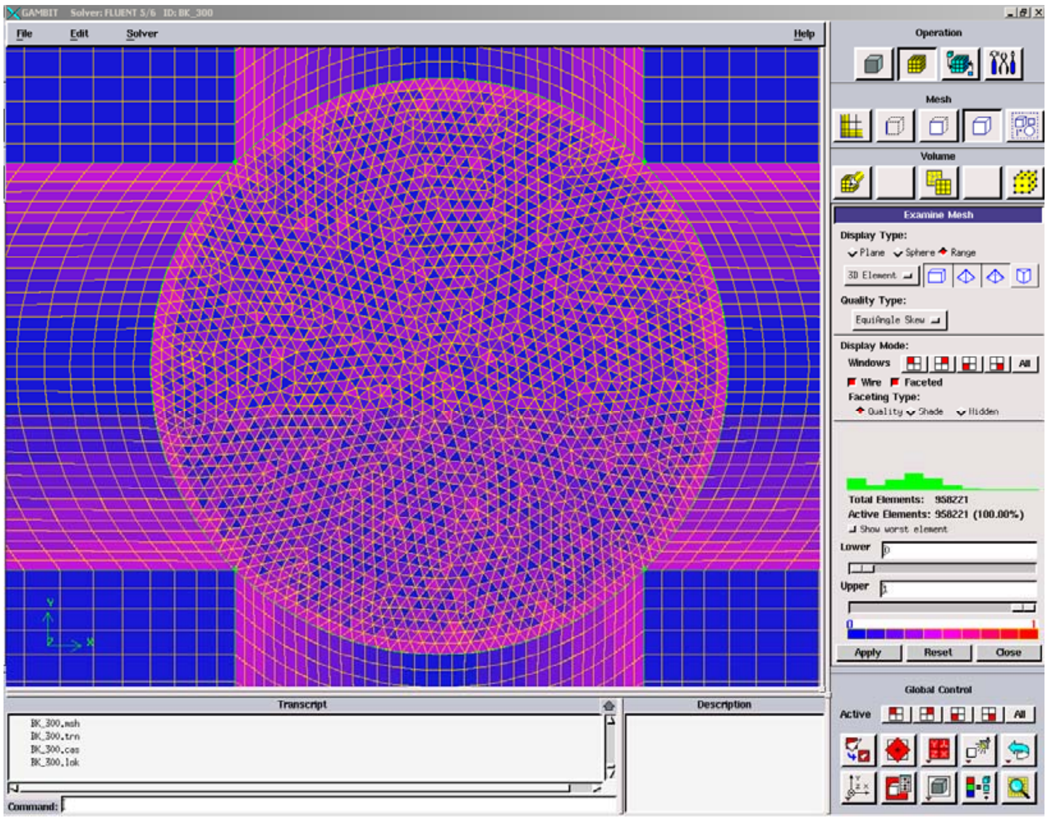
Cell quality	Excellent	Good	Acceptable	Poor	Sliver	Degenerate
Skewness	0.00-0.25	0.25-0.50	0.50-0.80	0.80-0.95	0.95-0.99	0.99-1.00

Table 8.7: Guidelines to judge the cell quality based on skewness

In any case the skewness of hexahedral, triangular and tetrahedral cells must be lower than:

- Hexahedrals: < 0.85
- Triangles: < 0.85
- Tetrahedrals: < 0.90

For the CFD calculation for the faculty of Architecture of Delft University of Technology, the grid quality is examined in Gambit using the skewness of the cells. In Figure 8.45 the skewness of all cells of the central cylinder and a part of the domain around the cylinder is indicated with colors. Blue cells have a very low skewness and are therefore of high quality. Light purple and especially red cells have a higher skewness and their quality considerably decreases. In the central cylinder the quality of the tetrahedral cells is smallest near the edge of the cylinder. The quality of the hexahedral cells is smallest near the corner points of the parts of the domain outside the cylinder. The worst element of the domain is located in the central cylinder and has a skewness of 0.74. It can be concluded that the quality of the cells of the computational domain is acceptable.



Total Elements: 958221
Active Elements: 958221 (100.00%)

```
BK_300.msh
BK_300.trn
BK_300.cas
BK_300.lok
```

Figure 8.45: Examination of the mesh using the skewness of the cells

The green histogram at the right of the mesh window represents the quality distribution of the mesh elements. Each vertical bar on the histogram corresponds to a unique set of upper and lower quality limits. Ideally the first bar of the histogram should contain most cells, as this would give a grid of the highest quality. However, for the calculations that are performed in this thesis it is impossible to receive such a high quality. For the amount of cells that can be used to mesh the complex geometry, the presented quality is reasonably. Without refining the grid considerably, it is not possible to improve the quality of the grid at the moment.

8.7.4 Local grid adaption

Due to the limited computer resources, the building models discussed in this chapter are meshed with a considerably coarse grid. However, in some cases there are regions where smaller cells are required. To investigate smoke dispersion or wind hindrance problems for example, or to determine local forces for the connection of façade elements, a lot more cells are required. To provide a more accurate solution, the resolution of the grid at such interesting areas can be refined by local grid adaption. As the maximum amount of available cells was already consumed for most cases of this thesis, grid adaption is not considered in the calculations. However, to determine the exact flow around the balconies of the apartment building for example, the grid at those locations certainly has to be refined.

8.7.5 Boundary conditions

The boundary conditions that are applied on the building models and the ground surface are derived from the conditions formulated by Van Nalta. With wall parameters a certain roughness is given to the ground surface and the façades of the buildings. The surfaces of the original cube were modeled with a uniform roughness, but for building models with balconies or partial glass façades for example, the roughness is not uniform. However, as the accuracy of the simulations was of minor importance, all building façades are modeled with the same uniform roughness as the cube. To describe the reality accurately, all façades have to be modeled with a corresponding roughness off course. The Eurocode offers friction coefficients for three types of façades. However, not all existing façade variants can be divided in those three types, so further research is required to translate a certain façade structure in a specific roughness value. For all calculations, the boundary conditions for the ground surface were defined for an unbuilt area. However, when the environment is also modeled, the boundary conditions must be adapted and another roughness factor must be ascribed to the ground. The Dutch building code provides various roughness heights for several area types. Research is required to choose the right roughness height for a certain ground surface.

8.7.6 Wind direction

For all calculations that are discussed in this chapter, only one wind direction is considered. The building models are all placed with their longest side perpendicular to the flow, which results in the maximum possible pressure on those sides. However, in reality wind comes from different directions and just as for ordinary wind tunnels, at least the governing directions that follow from the wind rose should be investigated. As the Van Nalta domain is developed for wind coming from one direction, the only way to investigate other wind directions is to rotate the model of interest in the domain. One option is to rotate the model itself to the desired direction before it is meshed. This can be done in Rhinoceros for example, but many other CAD packages will be able to read and adapt the models as well. Also Gambit is able to rotate the model. A disadvantage of this method is that for each rotation the model has to be meshed again and an entire new calculation has to be performed, which takes quite some time. Another method to investigate different wind directions is proposed by Snijders [23] and implies a rotation of the total inner cylinder, including the mesh. When the inner cylinder is rotated over a small angle, the flow gradients will generally not change very much. Results from a previous calculation can then be used as a starting point for new calculations. This will certainly quicken the convergence process and shortens the time required to obtain a solution. However, the method is not applied yet and additional research is necessary to develop the method further.

8.7.7 Van Nalta domain

All calculations that are discussed in this chapter are performed in the Van Nalta domain. Although there is no confidence about the accuracy of the results, the domain seemed suitable of performing the calculations. However, the domain is very large in comparison with other domain sizes that are used in practice. According to Van Nalta [16] the dimensions are even more than sufficient. Although the cells are larger near the boundaries of the domain, still quite a lot of cells are required to mesh the domain outside the cylinder. To be able to refine the grid in the cylinder, it is perhaps possible to reduce the size of the computational domain. The velocity profiles at the boundaries will possibly be unaffected as well for a smaller domain.

Another way to save cells is to remove the smallest elements from the research area. Especially for simulations with several buildings this can be valuable as such models require lots of cells. When the buildings with the smallest influence on the flow pattern are removed, cells can be saved to refine the grid on the remaining other buildings.

A final point of attention is the radius of the central cylinder. A certain distance between the buildings and the edge of the cylinder is required to provide a smooth transition between the mesh on the building models and the cylinder edge. For the calculations with the single building models a radius of 212 m is used, which resulted in a sufficient distance between the building model and the edge of the cylinder. The radius of 212 m originates from the calculations with the cube in the previous graduation studies. The ratio between the distances from the origin to the most far away corner of the cube and the ground edge of the cylinder is 1:5. This ratio is given in the left picture of Figure 8.46. For the calculation with a part of the university district, the ratio between the radius of the smallest circle around the model (250 m) and the radius of the cylinder (500 m) is only 1:2. Although the results of the simulation seemed reasonable, it is unknown if this ratio is sufficient. However, a ratio of 1:5 appears to be useless for this case as it results in a radius for the research area of 1250 m. Further research is required to investigate the effects of the radius of the central cylinder on the results of a calculation.

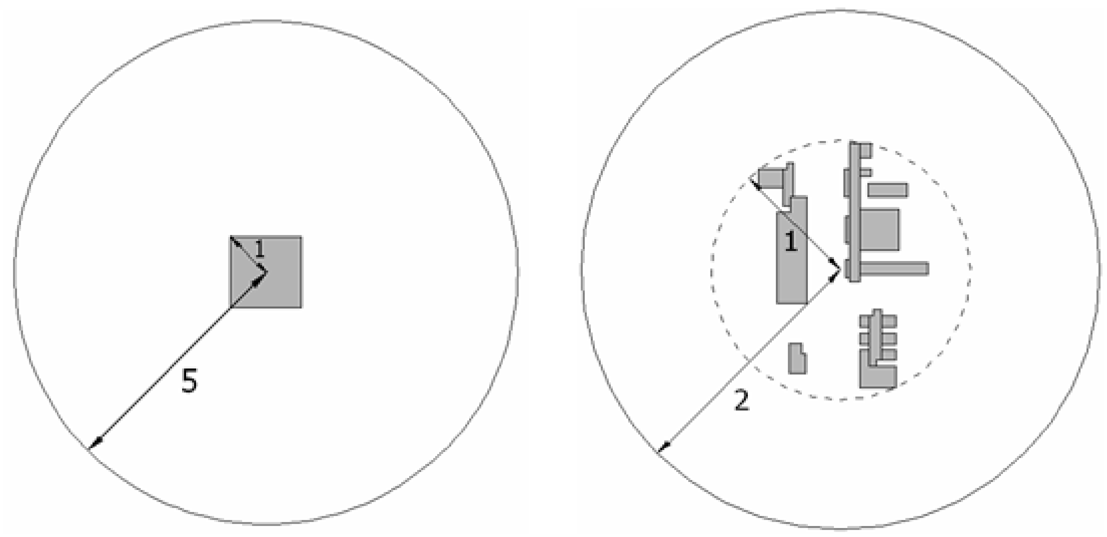


Figure 8.46: Ratio between the distances to the edges of the research area and the cylinder

9. Thesis Evaluation

In this chapter an evaluation is given of the Master's thesis project. The purpose of the thesis and its contribution to the Virtual Wind Tunnel is summarized. A review of the development process of the various design tools and a vision for the future of the Virtual Wind Tunnel is given in this chapter as well.

9.1 The Virtual Wind Tunnel

Until the second half of the 20th century, wind loads were hardly taken into account in the structural design of buildings. With the introduction of new and stronger materials in building engineering around the 1950's, structural elements became lighter and more slender. As the permanent vertical load decreased, wind loads became more important for the design of high-rise buildings. Research into wind loads resulted in various design codes to take the effects of wind on common building shapes into account. For centuries these codes were the only way to estimate the pressure on a building. However, for complex building shapes the codes seemed not sufficient and wind tunnel tests have been introduced to determine the wind loads on these buildings by experiment. Although wind tunnel studies give reliable results for situations where the codes do not foresee, they are very expensive and time-consuming. This discourages designers to perform such a test in the early stage of the design process. However, as it is just this stage where very important design decisions are made, more insight in the wind loads and flow pattern is desired. As a solution, computational methods are proposed to calculate the wind effects numerically using a technique referred to as Computational Fluid Dynamics (CFD). In theory CFD is able to solve the mathematical models up to a desired accuracy without the limitation of geometry. As changes in the physical model are relatively easy to perform, CFD seems very suitable for shape optimization with respect to wind loads.

However, for use in wind engineering CFD is not generally accepted yet. As there is still contradiction on the use of CFD to predict the wind effects on buildings and structures, results of CFD calculations are not widely accepted. To develop a general approach for CFD in wind engineering, a computational wind tunnel has been proposed at the Structural Design Lab of Delft University of Technology. The goal of this *Virtual Wind Tunnel* is an application that can be used in the early stage of the design process to indicate the wind loads on a building or structure using CFD. Ultimately, this structural design tool should enable structural engineers without any specific knowledge on wind engineering to perform reliable wind load calculations. The possibility of comparing several alternative geometries in a relatively short period should make shape optimization with respect to wind loads possible.

By now a computational domain is developed at the Structural Design Lab by Van Nalta [16] and Snijders [23], which showed promising results for the calculation of the pressure distribution on a 60x60x60 m^3 cube. However, for CFD simulations on complex building geometry the computational demand is very high and the limited power of the current desktop computers influences the ability to make accurate calculations. Only with very powerful machines or with a cluster of computers it will be possible to obtain accurate results for complex geometry. Nevertheless, as most structural engineers will only have a normal desktop computer at their disposal, the Virtual Wind Tunnel is not suited yet for final wind load determination. However, as the computer resources are developed continuously, it could be possible in the nearby future to simulate the wind effects on complex buildings accurately. Research into smoke dispersion, wind hindrance, heating or ventilation problems will also come into reach then.

To predict the wind loads on a building or structure and compare alternative geometries in a relatively short period, the process of setting up a CFD calculation has to be automated. When the building geometry is generated as much as possible automatically, less interference of the structural engineer is required and valuable time can be saved. Anticipating on the future developments in computer resources, several design tools are developed in this thesis to generate the building geometry for CFD calculations. With this toolbox it is possible to setup a CFD calculation and compare several alternative geometries in a relatively short time.

9.2 Design Tools

During this Master's thesis several design tools are developed to generate the geometry for CFD calculations. The purpose of the tools is to quickly setup a CFD calculation without much interference of the user. As alternative geometries can be generated in a relatively short time, the tools are very suitable for shape optimization with respect to wind loads. With the design tools three different types of geometry can be generated. The first and second design tools can be used to generate a 3D model of the environment for a certain area. The third design tool can be used to simplify the geometry of the building of interest and get rid of little details. The fourth design tool can finally be used to generate the computational domain for the CFD calculations. In this paragraph the possibilities and restrictions of the various design tools are discussed extensively.

9.2.1 Modeling the environment

As the flow pattern around a building is influenced by its surroundings, it can be very useful to take the environment into account in the calculations. Besides a model of the building of interest, also a 3D model of the environment is required then. The difficulty here is that there are hardly any useable 3D models of a certain area available. However, GIS technology offers two information models that can be used to generate a 3D model of a certain area in the Netherlands. The first dataset is the *Top10Vector*, which provides a digital topographic map of the whole country in 2D. The dataset is composed of closed polygons that represent various elements on the earth's surface, like buildings, roads and vegetation. The second dataset that originates from GIS technology is the *Actueel Hoogtebestand Nederland* (AHN). The AHN dataset provides an elevation model of the Netherlands where the height is stored for each 16 m^2 of the country. The height points are placed on a regular grid where each point represents the height for that particular location. The dataset contains height information of the ground surface, buildings, roads and water for example.

The first design tools that are developed during this thesis are able to link the two discussed datasets in order to generate a realistic 3D model of the environment for a certain location. The amount of built environment that has to be taken into account for the CFD calculations can be limited. For real wind tunnel studies, a research area with a radius of 300 m around the building of interest seems sufficient. Buildings further away do not have a significant influence on the flow pattern around the central building anymore. For the CFD applications the extent of built area that is represented in the calculations will initially be similar as for real wind tunnel studies. The process to generate a 3D model of the environment is divided in two parts. The first design tool can be used to create a restricted 2D research area from the Top10Vector dataset. By assigning the center location of the building of interest and the radius of the research area, a circular region with limited dimensions is extracted from the digital topographic map. With the second design tool all polygons in this area can be extruded to create a 3D model of the environment. The extrusion heights are derived from the AHN dataset for that particular area. As one single polygon will contain several height points, the design tool calculates the average of all height values and extrudes the polygon over the concerning height.

The Top10Vector dataset that was used for this thesis only contained information about the southern part of Delft. The AHN dataset provided even less information as it only contained height values for a part of the Delft University of Technology district. However, the design tools showed promising results and if the Top10Vector and AHN datasets become available for other areas, the tools will also be capable of generating 3D models for these locations.

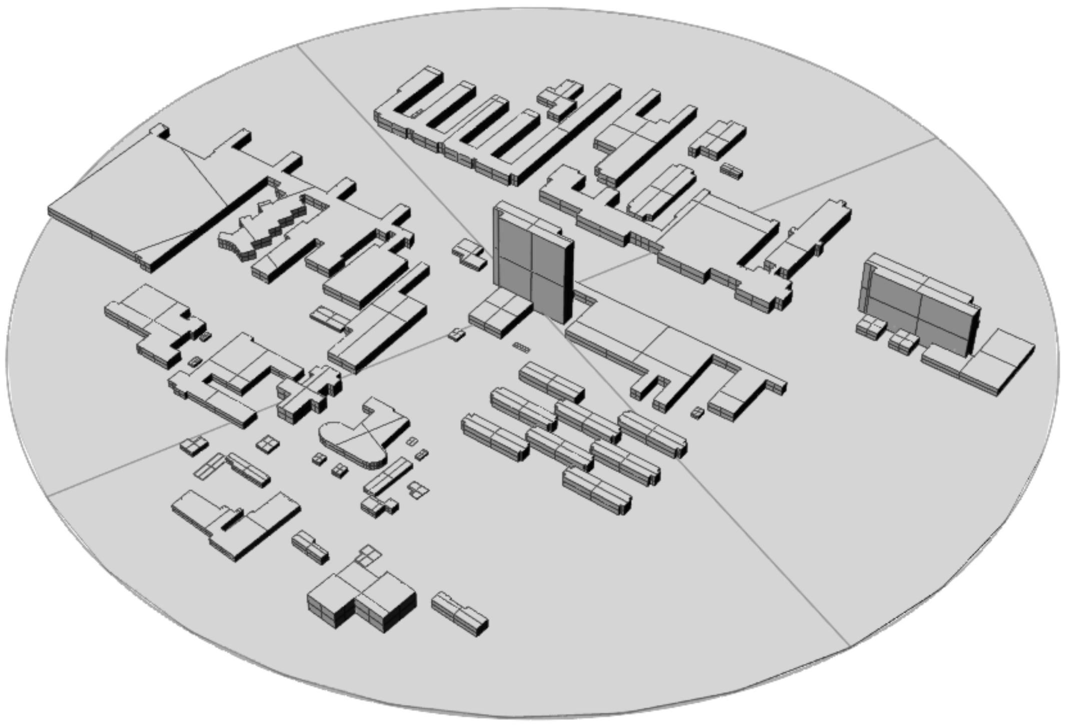


Figure 9.1: Result of the first design tools: a 3D model of the environment

Restrictions

Although the design tools work properly, there are some restrictions of the methods. First of all, the buildings in the Top10Vector dataset are represented by only one polygon. When the polygon is extruded over the average of all height values inside the polygon, the building has the same height everywhere. However, in reality buildings often consist of several parts with their own heights. For an accurate representation of the environment it is therefore desired to split the original polygon into several polygons that represent the various building parts. Extruding these parts over the different heights gives a building model that represents reality much better. Nevertheless, as a total building is represented as only one polygon at the moment, it is extruded over the average of all building parts. Complex methods are available that can recognize a large variation between neighbouring height points. With these methods it could be possible to generate a new polygon around the divergent height points. The original polygon that represents the total building can be divided in smaller polygons then that represent the various building parts. However, the point density of the AHN dataset is only one point per 16 m^2. This is very coarse and a polygon that is generated around some divergent points will certainly not represent the shape of the real building part. The various polygons will also not connect properly then. Only when the density of the AHN dataset is increased considerably, the method can work. Another option is to manually split the polygons in several parts and extrude the various building elements over the concerning height then. However, a thoroughly knowledge of the buildings in the research area is necessary for this method. It is therefore not widely applicable.

Another aspect is that no slope surfaces can be modeled with the developed methods. All polygons are extruded straight ahead and have a flat top side. If a building has a slope roof in reality, it is modeled as a flat building with the average of all height values as extrusion height. However, a very accurate description of the surrounding buildings is not necessary. Most detail is required for the building of concern. Only for a very accurate prediction of the flow pattern at specific locations in the environment, more detailed models of the surrounding buildings are required. At the moment GIS technologists investigate the possibilities to develop more accurate 3D models of a certain environment. If these methods become widely available in the future, more detailed models of the surrounding buildings can be generated for use in the Virtual Wind Tunnel. However, for global wind load determination on buildings that are placed in a built environment, the accuracy of the surroundings is sufficient.

A final point of concern is that only the buildings in the environment are modeled. All other elements, like roads, vegetation and water are not represented in the 3D model. A complex ground surface with differences in height would cause a very complex grid on the bottom of the research area, which results in less cells being available for meshing the building models. In stead of those various elements, the environment is modeled with a flat ground surface. However, for mountainous areas or areas with dikes or elevated roads for example that will influence the flow pattern considerably, it is important to model these elements. Some small adaptations of the developed scripts will make this possible already. Each element type has a separate layer in the Top10Vector dataset. The scripts remove all polygons from the model that do not represent a building by turning off their layers. Adapting the scripts so that a certain element is not removed will give a more detailed model of the environment.

9.2.2 Simplification of the building of interest

To determine the wind loads with computational methods, a CAD model of the building of interest is required. When such models are obtained from architects or structural engineers for example, they usually contain lots of information that are not relevant to determine the global wind loads. Little details have hardly any influence on the flow pattern, but they will cause a very complex grid that increases the calculation time tremendously. Another aspect is that only the surfaces that come in direct contact with the flow are necessary for CFD analysis. However, most building models contain internal geometry as well, like interior structures and furniture objects. Removing all unnecessary geometry by hand to remain only the exterior surfaces of the building model is very hard as there is not always a strict distinction between internal and external geometry. Façades for example can be constructed from various separate elements, from which some are part of internal structures as well. Deriving only the exterior surfaces of the building model is very complicated then. A final aspect is that for CFD analysis the building models have to be fully airtight. Small drawing inaccuracies can introduce gaps through which air can flow into the building, which is off course not permitted. To cover these problems, a third design tool is developed that excludes little details and replaces all internal and external geometry by wrapping an airtight surface around the building model of interest. It is only this wrapped surface that is used for CFD analysis.

Several methods are developed to create the wrapped surface around a certain building model. The methods are based on creating enclosing curves around the model at several height levels. By generating a surface through these curves then, the model can be wrapped. The various methods are briefly discussed in the rest of this paragraph.

Rotating Lines method

The first method to create the enclosing curves around the model of interest is the *Rotating Lines* method. The method is based on finding the outer points of the model at a certain height level. When all outer points are found, a curve can be generated through the points that encloses the model at that level. Repeating this procedure for the full height of the building, where the level of concern is elevated over a certain distance each time, gives various enclosing curves over the height of the building. A surface can be generated through the curves then that wraps the model and replaces the internal and external geometry.

With the *Rotating Lines* method the user is first asked to pick the center location of the building model. From this location a line is drawn in north direction with a user-specified length. For that line it is checked if there is an intersection with the various elements of the model. If so, the script determines the intersection point for that line with the largest distance from the center location of the model. At this location a point is added to the model then. The procedure is repeated for a second line that is rotated over a user-specified angle around the center location of the model. The script continues until the line is rotated over 360 degrees, giving the outer points for all directions. By adding a polyline through these points, the model is surrounded by an enclosing curve. The method is explained in Figure 9.2 for a simple building model.

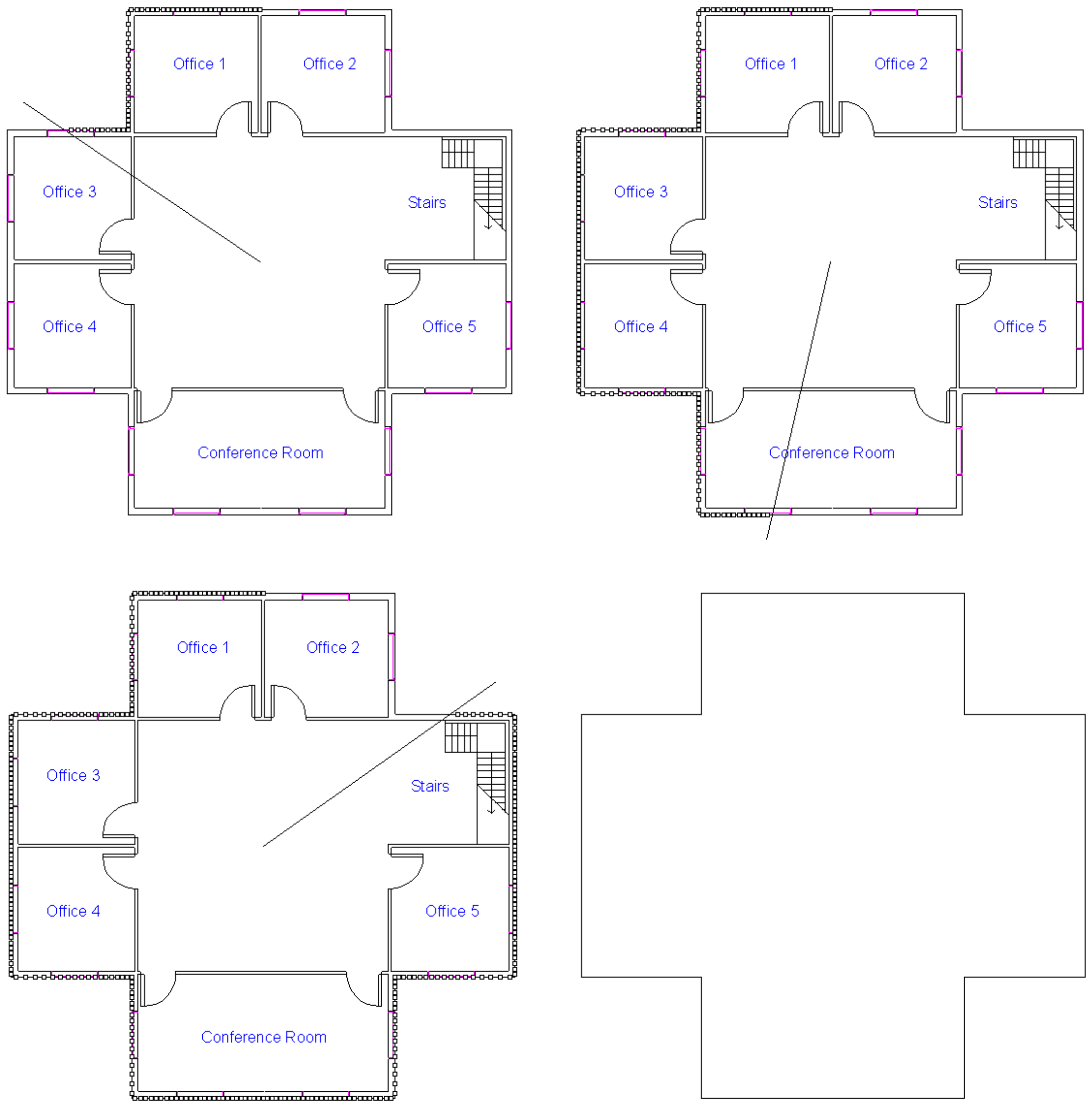


Figure 9.2: Rotating Lines method for a simple building model

Rectangle method

The second method that is developed to create the enclosing curves around the model of interest is the *Rectangle* method. The method is especially developed for large models with a rectangular shape, as the *Rotating Lines* method seemed not very suitable for such models. As a consequence of the rotation of the line, the distance between the outer points increases as the distance between the boundaries of the model and the center location of the model increases. Especially at corners the enclosing curve will differ from the original model then. A method that deals with this problem is the *Rectangle* method. The method is also based on finding the outer points of a model by the intersection of a line with the various elements of the model. A rectangle that has to be drawn around the model is now used to define the movement and length of the lines. Starting from the lower left corner of the rectangle, a vertical line will be created to the upper left corner of the rectangle. The line is then moved to the right over a user-specified distance. For each line it is checked if there is an intersection between the line and the various model elements. The distances between the various intersection locations and the start point of the line are then calculated. At the closest and most far away intersections a point is added to the model. The process is repeated until the line element is finally arrived at the right side of the rectangle. The procedure now continues with a horizontal line moving from the bottom of the rectangle to the top of the rectangle. When all outer points are found, a polyline is added through the points to generate the enclosing curve. The method is explained in Figure 9.3 for the simple building model.

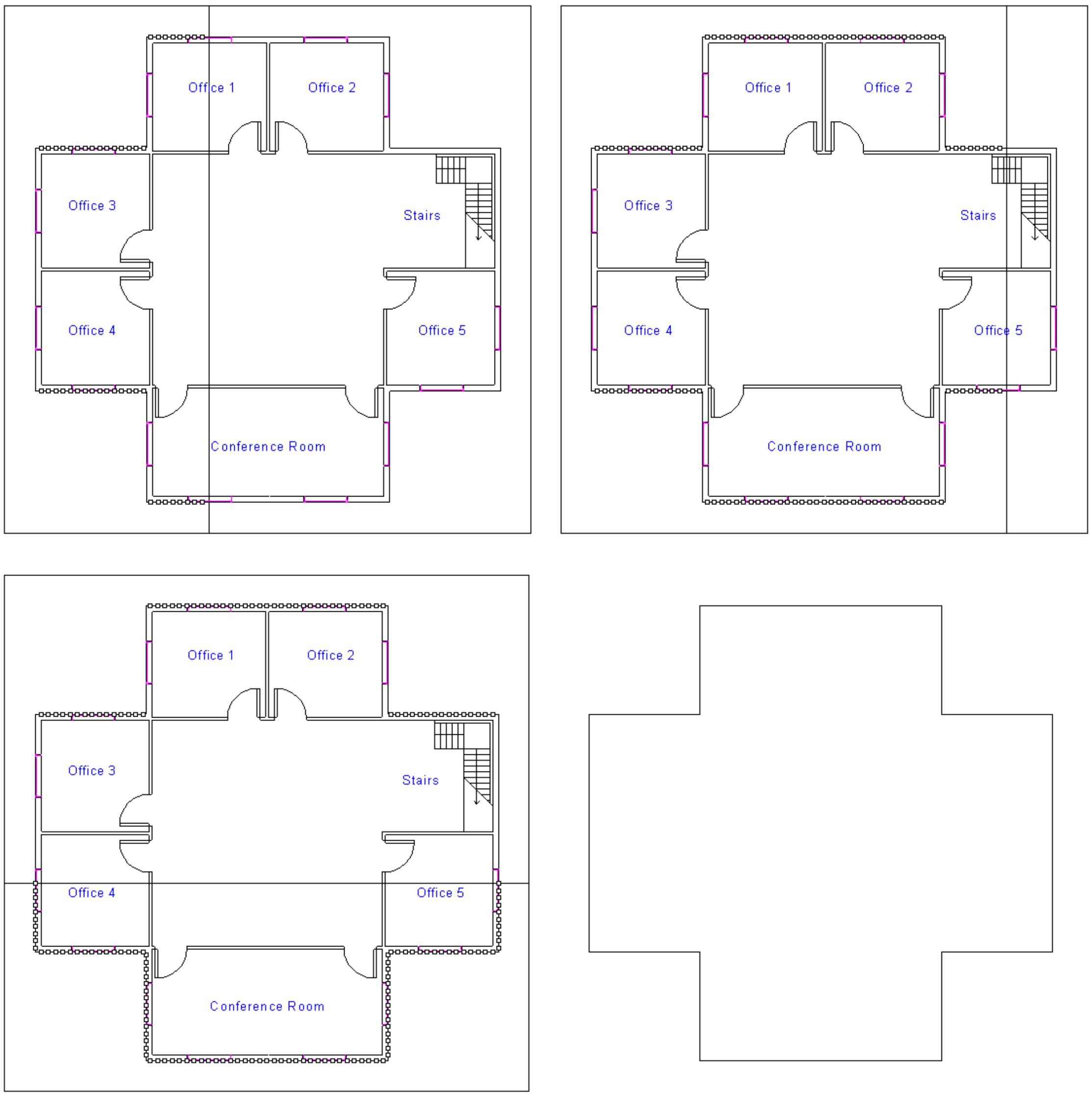


Figure 9.3: Rectangle method for a simple building model

Rotated Square method

The third method that is developed to create the enclosing curves around the model of interest is the *Rotated Square* method. The method is developed to find the outer points in case of in-built façades. The *Rotating Lines* method and the *Rectangle* method do not give proper results for such building models, as most outer points on the in-built façades are not found. The *Rotated Square* method, which is based on the *Rectangle* method, deals with this problem. The user still has to draw a rectangle around the model of interest. The corner points of this rectangle are now used to generate a square that is rotated over 45 degrees. The rotated square fits the original rectangle exactly and forms the basis for the lines that will find the intersections with the various model elements. Starting from the lower left edge of the square, lines will move to the upper right edge of the square over user-specified intervals. For each interval it will be checked if there is an intersection between the line and the various model elements. If the line has finally arrived at the upper right edge of the square, the procedure continues with lines moving from the upper left edge of the square to the lower right edge of the square. In Figure 9.4 a simple building model with in-built façades is given. A rectangle that has to be drawn around the model forms the base for the rotated square. Lines will then move from the lower left edge of the square to the upper right edge to find the outer points of the model.

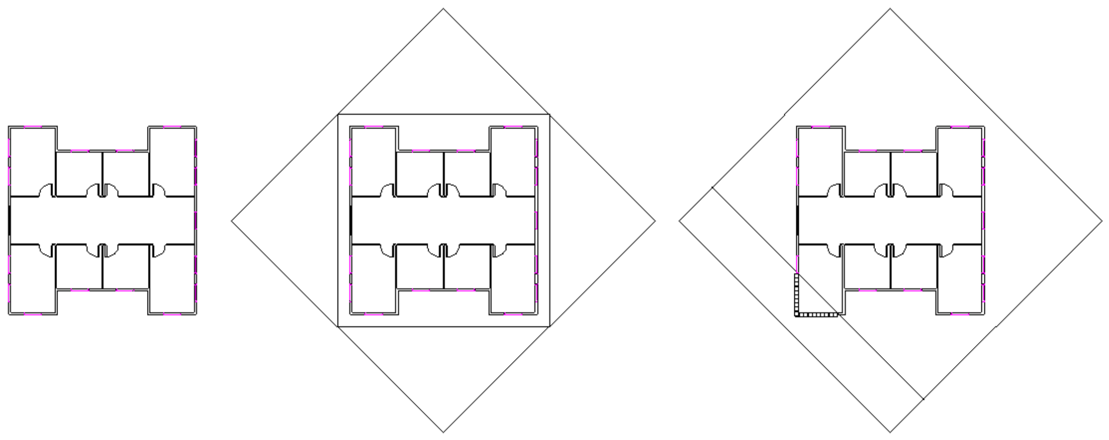


Figure 9.4: Rotated Square method for a building model with in-built façades

In Figure 9.5 the process continues. When the line has arrived at the upper right edge of the square, the procedure is repeated, but now with lines moving from the upper left edge to the lower right edge of the square. When all outer points are found, the enclosing polyline is generated through the points.

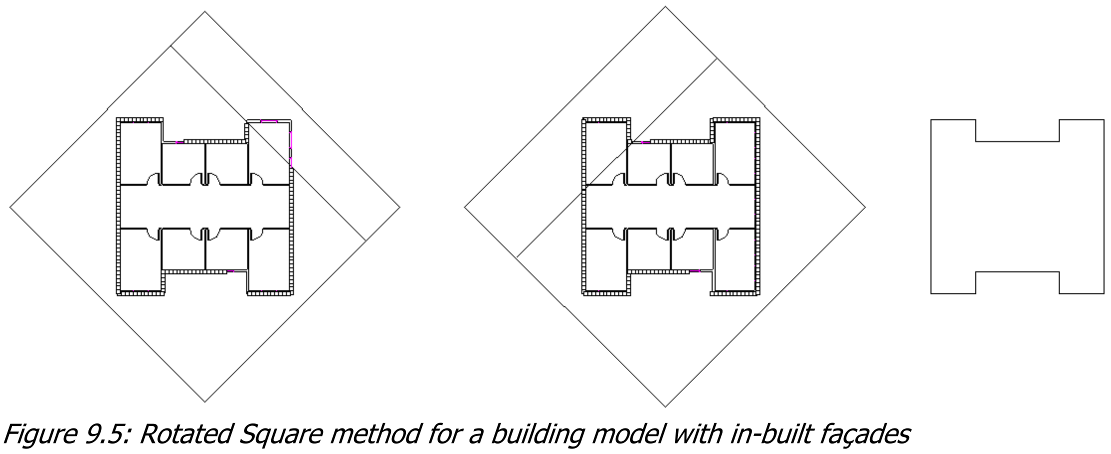


Figure 9.5: Rotated Square method for a building model with in-built façades

Integrated method

The discussed methods to generate the enclosing curves seem to work properly for circular and rectangular building models. However, especially for multiple curved models the methods sometimes fail in finding all outer points. Several tests showed that the three methods are all capable of finding certain outer points of such models, but not all points. Additionally, each method found some points that were not found with the other methods. To find all outer points on a complex, multiple curved surface, the three separate methods are joined in the *Integrated* method. Various tests with multiple curved models showed that the *Integrated* method succeeded each time to find all outer points of the models. A rectangle that has to be drawn around the model forms the base again of the method. The procedure to find all outer points starts with the *Rectangle* method and is directly followed by the *Rotating Lines* method. The middle of the rectangle forms the centre point of the rotating lines method. The lines with a length equal to diagonal between the lower left and upper right corner of the rectangle will rotate around the centre point to find the outer points of the model. Finally the *Rotated Square* method is executed, where the rotated square is constructed from the drawn rectangle. As a result of the execution of the three methods in one continuous process, the outer points do not lie in the right sequence in the array and it is not possible to directly generate the enclosing curve through the points. The array containing the outer points will be ordered first by a procedure that is based on finding the closest point for a certain outer point. Then the enclosing curve is generated through the points.

Extension to the third dimension

The methods that are discussed in this paragraph can be extended to the third dimension by elevating the level at which the outer points are determined. Starting from the bottom of the model, the level of concern is raised each time over a user-defined distance to generate the enclosing curves at several height levels. The procedure is repeated until over the full height of the building the enclosing curves are generated.

Restrictions of the methods to generate the enclosing curves

With the discussed methods it is for many cases possible to find the outer points on a two dimensional plane and to generate an enclosing curve that fits the original shape accurately. However, when the methods are applied in three dimensions, problems arise when a building model consists of various parts with a different height that are separated by air. As only one enclosing curve is generated through the outer points at a certain height level, the curve can also cross areas where no points are found, for example the air between separate building parts. When a surface is lofted through the curves, the air would be modeled as a building part then. It is suggested to adapt the methods to find the outer points on both a horizontal and a vertical plane that moves through the model. On the boundaries of the model a point cloud will arise then, through which a surface can be constructed.

Another restriction of the various methods is the time required to find the outer points of large building models. The limited processing speed is caused by the huge amount of elements of such models and the Rhinoceros Visual Basic language in which the methods are scripted. Although the process of the methods is very clear as all procedures are visualized on the screen, it takes quite some time before all enclosing curves of an entire model are generated. It is suggested to translate the various scripts to more advanced programming languages. As these languages are more complicated, some experience in programming is certainly required then. However, it could speed up the process considerably.

For additional research some suggestions are given for other methods to simplify the model of the building of interest. By using a shrinking sphere with a certain elasticity, the model could be wrapped when it is prohibited to perforate the sphere. A vacuum could be generated then that creates a fitting surface around the model of interest. Another method could be developed that works similar to laser scanning equipment. When a script is able to imitate the laser scanner, a point cloud on the boundaries of the model could be generated. This point cloud can be converted to a 3D surface model then that wraps the original model accurately.

Curve simplification

Applying the discussed methods to find the outer points of a model give the enclosing curves for several height steps. When the model is highly detailed or when it does not have smooth façades, the curves will fluctuate considerably. To simplify the enclosing curves and get rid of little details, a NURBS curve can be fitted through the enclosing curves. This technique uses the outer points through which the enclosing curves are constructed as control points to fit a NURBS curve through it. By assigning parameter values to the control points, a curve can be created that fits the points exactly or more approximately, dependent on the given parameter values. In Figure 9.6 an example is given of the NURBS fitting method. The dots in the figure represent the outer points that could be found for a certain façade. If the enclosing curve is generated through the points, a reasonably rough and fluctuating curve originates, which is represented by the dotted line in the figure. To simplify the curve and get rid of those fluctuations, a NURBS curve can be fitted through the outer points. Weight factors that are assigned to the outer points determine the influence of the points. They are the parameters that indicate the accuracy of the fitting process. If the weight value of the control points is larger than one, the points are closer approximated by the NURBS curve. If the weight values are smaller than one, the points are less approximated. In this way it is possible to create a straight NURBS curve that replaces the fluctuating enclosing curve and neglects small details.

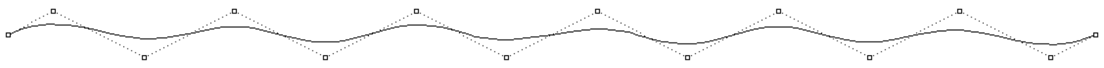


Figure 9.6: Enclosing curve simplification by fitting a NURBS curve through the outer points

However, it is not always necessary to simplify the enclosing curves. When a building has smooth façades for example without much detail, the enclosing curves will not fluctuate very much and perhaps they do not have to be simplified. It is up to the user of the design tools if the enclosing curves have to be simplified or not. In addition to the NURBS fitting method, another method to simplify the enclosing curves was proposed in this thesis. In theory, the least squares method should also be capable of simplifying the enclosing curves. The least squares method is in general a method to determine the best-fit curve through a certain dataset. It is based on minimizing the squared error in vertical direction between a data point and a parametric curve. The smaller the difference between the original dataset and the proposed curve, the better the fit. In Appendix O the method is discussed extensively and examples are given to fit a straight line and a parabola through a certain dataset. In order to fit a straight line through the dataset, a first degree polynomial is required that minimizes the total error. To fit a parabola through the dataset, a second degree polynomial is required. The first and second degree polynomials are a function of x, with only one solution for a certain x value. However, as the enclosing curves of the building models are closed curves, at least two solutions have to be found for a certain x value. Third degree curves at minimal are therefore required to formulate the fitting polynomials. It was suggested to use an ellipsoidal or circular equation to fit a curve through the outer points, but some tests showed that the result is always an elliptical curve. As this does not represent the reality, curves of a higher degree are necessary to generate a best-fit curve through the outer points. Nevertheless, the use of only the NURBS fitting method to simplify the enclosing curves is probably sufficient. If it is desired to use another method as well, the least squares method seems to be an appropriate method. Additional research is required then to develop the method further.

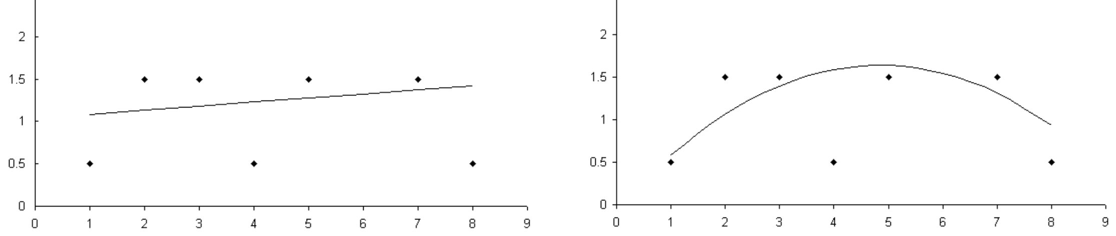


Figure 9.7: Least squares method: fitting a straight line and a parabola through a dataset

Lofting method
In this chapter several methods are discussed to generate the enclosing curves around a building model. The methods can be extended to the third dimension to create the curves at different height levels. Starting from the bottom of the building model, the level at which the outer points are determined is raised each time to generate the enclosing curves over the full height of the building. After the curves are created, the original building geometry is removed from the model. Then a surface can be lofted between the enclosing curves to form a closed surface model. It is just this hollow model that is used for the CFD calculations. The surface model replaces all inner and outer geometry of the original building model and reduces the amount of data for the CFD software considerably. When the surroundings also have to be taken into account for the calculations, the simplified building model can manually be placed in the 3D model of the environment. In Figure 9.8 the lofting method is given for the model of an office building. A hollow surface model is generated from the enclosing curves.

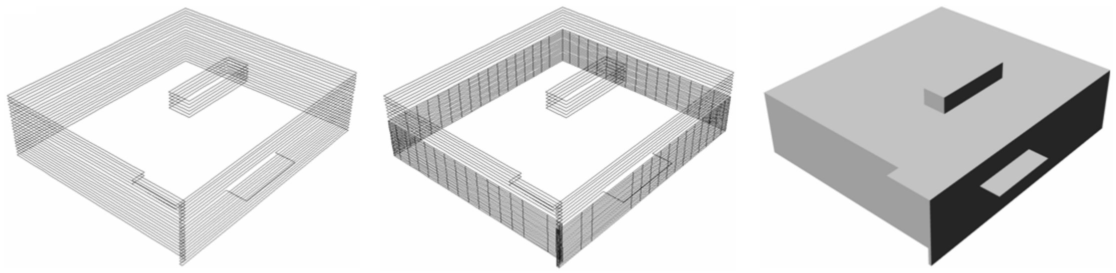


Figure 9.8: Lofting method to create a hollow surface model from the enclosing curves

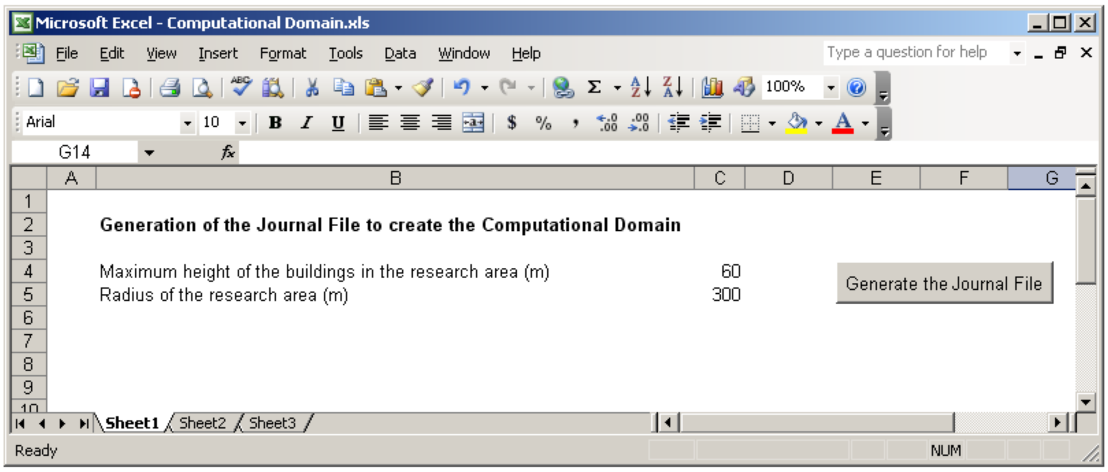
Preprocessing
A final method that is developed for the simplification of the building of interest is a method called Preprocessing. As architects and structural engineers work with various CAD systems nowadays, the various tools to simplify the building of interest must support the different model types. In traditional CAD systems the models are generated from lines, surfaces and solids and those entities remain when the model is imported in Rhinoceros. The developed methods to generate the enclosing curves use specific Rhino commands that are able to find the intersections with lines, surfaces and solids directly. However, when building models are created with modern CAD systems, like object-oriented modellers, the models are not constructed from lines, surfaces or solids anymore. These models are constructed from objects that represent the elements from which a building or structure is built of, like beams, columns, floors and walls. As Rhinoceros is not an object-oriented CAD system, the objects are converted to meshes when such models are imported in Rhinoceros. A mesh is basically a collection of vertices and faces connected together to represent a surface. Unfortunately, the current version of Rhinoceros is not capable of determining the intersection between a line element and a mesh object. However, with the Preprocessing method the mesh objects can be converted to surfaces by deriving the vertices and faces from which the mesh is built. Through these entities a surface can be created, which makes it possible to determine the intersections with the line element. Nevertheless, converting the meshes to surfaces results in a significant growth of model elements. For large building models it can take quite some time then to find all outer points, as for each line it has to be checked if there is an intersection with one of the elements. Manually deleting some inner geometry before the methods are executed will quicken the process considerably, as less elements have to be checked.

Another option to determine the intersection with the line element and a mesh object is using the newest Beta release of Rhinoceros 4. Some additional commands and functions are implemented in this release that are able to determine the intersection with a mesh object directly. For each method to find the outer points of a model and create the enclosing curves, a variant is developed that supports these new commands. Together with those variants of the various methods it is possible to find the intersections for models that are constructed from lines, surfaces and solids as well as models that are constructed from meshes.

9.2.3 Generation of the computational domain

The final design tool that is developed in this thesis is a tool to generate the computational domain in which the CFD calculations are performed. To obtain reliable results from the calculations, a domain with sufficient dimensions around the research area is required. The domain must be large enough to avoid that the flow pattern at the boundaries of the domain is influenced by the buildings in the research area. The Van Nalta domain that is developed in recent graduation studies by Van Nalta [16] and Snijders [23] forms the base for the domain that is used in this thesis. The Van Nalta domain was developed to investigate the wind effects on simple objects, like cubes and cylinders. The size of the original domain depends on the height of the object placed in the central cylinder of the domain. In this thesis the domain is used to perform calculations on entire buildings and even areas containing multiple building models. The cylinder of the original domain is therefore too small and the structure of the domain had to be adapted. For calculations with a research area that consists of several buildings, the dimensions of the domain must depend on the radius of the research area and the maximum height of a building in the research area. A final design tool is developed that automatically creates the domain for a certain research area, depending on the dimensions of the area.

The Fluent pre-processor Gambit is used to generate and mesh the computational domain. Gambit can run so-called journal files in which text based commands can be scripted. When the journal file is run, Gambit works over the commands given in the file to generate the model that is scripted. The developed final design tool can be used to create the journal file with which the computational domain can be generated in Gambit, depending on the maximum height and radius of the research area. The journal file is generated using a Visual Basic macro that is run in Microsoft Excel. The file also contains commands to mesh the regions of the domain outside the central cylinder. After the domain is generated, the user can place the model of the research area into the cylinder. It is only this cylinder that has to be meshed manually then.


Microsoft Excel - Computational Domain.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
10
100%
G14
Sheet1
Sheet2
Sheet3
Ready
NUM
Generate the Journal File

Generation of the Journal File to create the Computational Domain

Maximum height of the buildings in the research area (m) 60
Radius of the research area (m) 300

Figure 9.9: Macro to generate the journal file

9.3 CFD calculations

9.3.1 Introduction

For several building models that were generated with the various developed design tools, some CFD calculations are performed to investigate the potential of the tools. The purpose of the calculations was to proof that the tools work for CFD applications and to demonstrate that calculations can be performed with the generated models. Two grid generation methods are developed that are able to mesh both simple and complex building geometry. Looking at the entire CFD process, the activities of this thesis concentrate on the first, second and last step: the description of geometry, the generation of a grid and the analysis of the calculation results. In the previous graduation studies by Van Nalta [16] and Snijders [23] the other steps of the CFD process are investigated and the results from these studies are used in this thesis to setup the CFD calculations. As there are still quite some uncertainties about the use of CFD in wind engineering, it was not the intention to obtain very accurate results from the calculations. Meshing complex building geometry also requires lots of cells, but the capacity of the current desktop computers is the limiting factor for the amount of cells that can be used. With the available hardware for this thesis it was therefore not possible to obtain very accurate results for complex models.

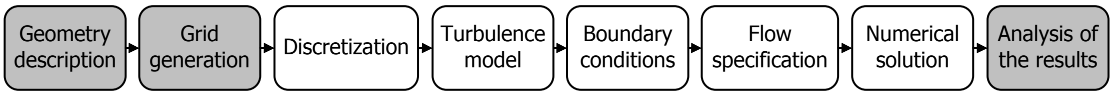


Figure 9.10: This thesis concentrates on the first, second and last step of the CFD process

9.3.2 Results

The CFD calculations that are performed in this thesis have demonstrated that the developed design tools work for CFD applications. The grid generation methods are able to mesh the various building geometry and results are obtained from the calculations. However, it is very hard to judge the accuracy of the results, especially for complex building models. The limited amount of cells to mesh the computational domain seriously influences the ability to obtain accurate results. The only way to verify the CFD results is to compare them with results from real wind tunnel tests or, in case of simple geometry, a calculation according to the building codes. For the simple shape of the faculty of Electrical Engineering of Delft University of Technology, a comparison is made between the codes and a CFD calculation. The comparison showed that the CFD results for the total wind load on the front façade are at the moment a factor 3,5 smaller than the results that are obtained from calculations according to the building codes. A comparison of the maximum pressure in N/m^2 on the façade showed a large difference of factor 2,7 between the methods as well. The reason of these large differences is not certain, but they are probably caused by an incorrect approach to setup a CFD calculation. An analysis of the results of a calculation performed by Snijders [23] for the original cube showed a difference of factor 3 as well in comparison with the building codes. As the same approach is used for the calculations, it is plausible that the current approach to setup a CFD calculation is inferior. Further research is therefore required to verify the results from a large number of CFD calculations with results from real wind tunnel studies in order to validate the current approach of the Virtual Wind Tunnel. If a certain trend can be derived from these comparisons, it could be used to outweigh the large difference between CFD and the building codes. The CFD results for the various forces in combination with the pressure distribution would be very valuable then for a first indication of the global wind loads in the early stage of the design process.

In this thesis several design tools are developed to create the geometry for CFD calculations. The design tools will assist the user of the Virtual Wind Tunnel to setup a simulation in order to predict the wind loads on a building or structure. Various tests with the design tools have shown that for many cases the tools succeed in generating the geometry. However, from the calculations that are performed in this thesis it can be concluded that a major bottleneck of the entire process to predict the wind loads is the CFD itself. The design tools are able to setup the geometry for the calculations, but the CFD technique is not able at the moment to obtain very accurate results from the calculations. To a large extent this has to do with the computer resources and it will probably take years before the wind effects on complex building models that are placed in a built environment can be simulated accurately.

9.4 Future of the Virtual Wind Tunnel

At the moment the structural engineer has to rely on the building codes or real wind tunnel studies to determine the wind loads on a building or structure. At the Structural Design Lab of Delft University of Technology, computational methods are proposed to calculate the wind effects numerically using CFD. The result of previous graduation studies by Van Nalta [16] and Snijders [23] and this Master's thesis is a computational domain in which the CFD calculations can be performed, a set of design tools to setup the building geometry for the calculations and some potential grid generation methods that are able to mesh both simple and complex building geometry. The actual status of the Virtual Wind Tunnel and its accompanying design tools is an application with which the wind effects on a single building or a built environment can be investigated, but the accuracy of the results is uncertain. Due to the limited computer resources, the amount of cells that can be used to mesh the research area is restricted. This seriously influences the ability to make accurate calculations. Further research is required to validate the results of a large amount of CFD calculations with results from real wind tunnel tests and the building codes. If reliable results can be obtained in the end, the accuracy of the CFD calculations should be superior over the building codes. If this can be achieved, the Virtual Wind Tunnel would be a very valuable tool for structural engineers to indicate the wind loads on a building or structure in the early stage of the design process. As it is this stage of the process where important design decisions are made, the increased insight in the wind effects can help the engineer to design more efficient structures. Shape optimization with respect to wind load is possible then.

The design tools that are developed in this thesis to setup the geometry for CFD calculations give promising results for many cases. The amount of detail of a 3D model of the surrounding buildings, that can be generated with the first design tools, is reasonable. As all buildings are modeled as straight blocks with a flat top surface, the accuracy of the model is not very high, but for global wind load determination the amount of detail of the environment is sufficient. The model of the building of interest requires more detail, but there is an ongoing battle between the amount of detail that is represented and the calculation speed of the CFD software. Very small details are not relevant to determine the global wind loads, but they will increase the calculation time considerably. Design tools are developed in this thesis to simplify highly detailed building models. A surface that is wrapped around the model replaces all internal and external geometry and heals eventual gaps in the model. For many cases the design tools seem to work properly, but there are also models imaginable where the methods fail. The time required to simplify very complex building model is also a point of concern. For an optimal use in the Virtual Wind Tunnel, the design tools should be developed further to support more complex shapes and to quicken the process to setup the geometry.

The solution of a CFD calculation is to a large extent influenced by the grid that is applied. For the calculations with the building models in this thesis, it was only possible to generate a coarse grid on the models. An accurate prediction of the flow pattern at specific locations is therefore difficult. Local grid adaption to refine the mesh in specific regions is possible, but this has to be done very carefully in order to retain a high grid quality. Further research is required to investigate the effects of local grid adaption for the results of the calculation and the quality of the grid. In the end, a grid independent solution is desired since it eliminates any errors that originate from the coarseness of a grid.

The Virtual Wind Tunnel has the possibility to become a valuable tool for structural engineers in the nearby future. With the actual capacity of desktop computers it is not possible yet to obtain accurate results for complex building models, but following the current trends in computer resources it can be concluded that the use of CFD for wind engineering problems is on the verge of being applicable. This off course under the condition that the large difference between CFD and the codes is solved. For final wind load determination the Virtual Wind Tunnel is not suitable yet and real wind tunnel studies remain necessary for the time being.

10. Conclusions and recommendations

10.1 Introduction

In this Master's thesis several design tools are developed to generate the geometry for CFD analysis. The purpose of the tools is to quickly setup a CFD calculation to determine the wind loads on a building or structure, without much interference of the user. The tools can be used to generate a 3D model of the environment and to simplify the geometry of the building of interest. A final tool can be used to generate the computational domain in which the calculations are executed. Many tests are performed with the various design tools to investigate the possibilities and restrictions of the tools.

For several models that were generated with the design tools, some CFD calculations are performed to test if the tools work for CFD applications. The results are promising, but the accuracy is still moderate. The capacity of the current desktop computers seems to be the limiting factor to obtain very accurate results.

10.2 Conclusions

10.2.1 Developed Design Tools

- Under the restrictions of the various methods, the developed design tools are very well able to generate the geometry for CFD calculations. A sufficient detailed 3D model of the environment can be created and complex building geometry can be simplified and eventually closed by wrapping a fitting surface around the model. For each case a computational domain with sufficient dimensions around the research area can be generated in which the calculations can be performed.
- The design tools assist the user of the Virtual Wind Tunnel to setup the geometry for the CFD calculations and can save valuable time. As physical changes in the building models are easy to perform, alternative geometries can be compared in a relatively short period. This makes shape optimization with respect to wind load possible.
- With the design tools to generate a 3D model of the environment it is not possible to model building parts of different height. In the Top10Vector dataset all objects are represented with only one polygon that encloses the entire object. If a building for example consists of several parts of different height, the polygon is extruded over a height that is the average of all separate building parts.
- The design tools to generate the 3D model of the environment are not able to model slope surfaces as well. All polygons are extruded straight up and have a flat top side. However, for a first indication of the global wind loads, the amount of detail of the surrounding buildings is sufficient. Most detail is required for the building of interest.
- The third design tool to simplify the building of interest contains four methods to create the enclosing curves. The *Rotating Lines* method, the *Rectangle* method, the *Rotated Square* method and the *Integrated* method all have their advantages in comparison with the other methods. The user must decide which method is the most appropriate for a certain building model.
- With the third design tool various building models that originate from different CAD systems can be simplified. The tool supports both traditional CAD systems and advanced object-oriented systems. When a model that originates from object-oriented systems is converted to meshes when imported in Rhinoceros 3, the meshes can be converted to surfaces with the *Preprocessing* method. Another option is to use the Rhinoceros 4 variant of the tool that is able to directly determine the intersection with a mesh element.
- The third design tool fails in generating the enclosing curves when a building consists of several parts with a different height that are just separated by air. As only one curve is generated at a certain height level, the air between the building parts will be modeled as a building element when a surface is lofted through the curves.

- The design tools to generate the geometry are scripted in the Rhinoceros Visual Basic language. As the tools work visually, the procedure of the tools is very clear. However, as only one task can be performed at the same time, the process to simplify the building of interest takes quite some time for complex building models that contain lots of elements. This is especially the case when meshes that originate from object-oriented modelers are converted to surfaces, as the amount of elements considerably increases then.
- For the simplification of the building of interest, valuable time can be saved by using the Rhinoceros Beta 4 variant of the methods in case the model is constructed from meshes. As this variant is capable of finding the intersections between a line element and a mesh directly, the meshes do not have to be converted to surfaces. As less elements have to be checked then, time will be saved.
- Significant time can be saved as well by manually filtering the various building models. When all irrelevant geometry, like furniture and internal structures, is removed from the model before the design tool is executed, the process can be quickened considerably.

10.2.2 CFD calculations

- The developed design tools work for CFD applications. The wrapped surface model of a building replaces all internal and external geometry and reduces the amount of data for the CFD calculations considerably. Models of the environment are easy to make, which makes it possible to take the surrounding buildings into account in the calculations. The models that are generated by the design tools can be meshed by the CFD software and results can be obtained from the calculations.
- The grid generation methods that are developed are able to mesh both simple and complex building geometry. However, the amount of cells that can be used is limited by the computer resources. The capacity of the current desktop computers influences the ability to make accurate calculations considerably. Only with the best available work stations or with a cluster of computers it could be possible to obtain very accurate results.
- Judging the results of the CFD calculations that are performed in this thesis is difficult as there was no reference information available. The only way to verify the results is to compare them with results from real wind tunnel studies. Judging the grid quality is also difficult as grid convergence studies can not be performed with the actual desktop computers.
- To obtain a solution with the highest accuracy that is possible with the current resources, the grid must be refined until a mesh is obtained with an amount of cells just below 1.2 million. Further refinement is not recommended as the calculation speed is slowed down considerably then.
- A comparison between the building codes and a CFD calculation for the simple shape of the Electrical Engineering building resulted in a large difference between the methods. The reason of the difference is not certain, but it is probably caused by an incorrect approach to setup a CFD calculation.
- In comparison with the building codes, a CFD calculation offers considerable advantages. Although the current accuracy is moderate, the CFD results can give lots of insight in the wind effects. The visualization of the flow pattern gives very useful information that can never be obtained with calculations according to the building codes.
- In the total process to predict the wind loads on a building or structure using CFD, the largest bottleneck still seems to be the CFD itself at the moment. The design tools are able to setup the geometry, but very accurate results can not be obtained yet from the calculations. To a large extent this has to do with the computer resources.
- For final wind load determination the Virtual Wind Tunnel is not suited yet because the accuracy is not high enough. Following the current trends in the development of computer resources, it will probably be possible in the nearby future to investigate the wind effects on entire built areas accurately. For the time being, real wind tunnel tests remain necessary.

10.3 Recommendations

10.3.1 Design Tools

- In order to generate more detailed models of the environment, it is recommended for further research to split the polygons that represent the buildings in smaller parts if a building consists of several building elements. The various building parts could then be extruded over different heights, what results in a more realistic description of the buildings in the environment. The generation of slope surfaces from the GIS datasets should also be investigated to simulate the flow pattern in the research area more accurately.
- In the 3D model of the environment the ground is modeled as a flat surface to save cells when meshing the domain. When the computer resources are sufficient, other elements in the environment, like roads, water, dykes and vegetation, should be modeled as well by adapting the script of the first design tool. This will result in a more complex ground surface and more cells are required to mesh the surface. However, especially for large differences in height between the various elements, reality will be described much better.
- The various methods of the design tool to simplify the building model of interest fail when a building model consists of elements of different height that are separated by air. It is recommended to develop other methods to solve the restrictions of the current methods. Adapting the methods to find the outer points on a vertical plane as well that moves in horizontal direction through the model could help to find the outer points on all boundaries of the model. Through the resulting point cloud on the boundary a surface can be created then.
- Other methods to wrap a surface around a building model are imaginable as well. It is suggested to use a shrinking sphere that has to be constructed around the model, with a certain elasticity. When it is not permitted to perforate the sphere, the building model could be wrapped by shrinking the sphere to create a vacuum. Another method is suggested that works similar to laser scanning equipment, generating a point cloud on the boundary of the model. Further research is required to develop these methods.
- At the moment, the only way to simplify the enclosing curves and get rid of little details is to fit a NURBS curve through the original curve. It is recommended to develop other methods as well to remove small details from a building model. The least squares method seems to be an appropriate method to simplify the enclosing curves, but a complete different approach of simplifying a building model could be valuable as well.
- The design tools are developed in Rhinoceros and written in the Microsoft's Visual Basic language. As the tools work visually and as only one task can be performed at the same time in Visual Basic, the process to generate the enclosing curves around a complex building model can take quite some time. It is recommended to convert the script of the tools to more advanced programming languages, like C++ or DotNET, in order to quicken the process.
- When a 3D model of the environment and a surface model of the building of interest are developed with the various design tools, the models have to be joined together manually at the moment. As the model of the environment uses a global coordinate system and the building model of interest a local coordinate system, some work is required to place both models at the right place when they are joined. An additional design tool could be developed that links the local coordinate system with the global coordinate system. The design tool should be able then to place the building of interest with the right orientation and at the desired location in the model of the environment.

10.3.2 CFD calculations

- It is very hard to judge the results of the various CFD calculations that are performed in this thesis as no reference information from real wind tunnel studies was available. For additional research it is recommended to compare the CFD results with real wind tunnel studies to verify the CFD calculations.
- The results of a CFD calculation are to a large extent influenced by the amount of cells that is used to mesh the domain. Due to memory restrictions of the available hardware, only a coarse grid could be used. If the computer resources allow, it is recommended to investigate the influence of the grid on the solution by performing a grid convergence study. In the end a grid independent solution is desired as it eliminates any errors that originate from the coarseness of the grid.
- The comparison between the building codes and a CFD calculation for the faculty of Electrical Engineering showed a large difference between the methods. The reason of this difference should be investigated, but it is possibly caused by an incorrect approach to setup a CFD calculation. Many comparisons should be made between real wind tunnel studies and CFD calculations for both simple and more complex geometry, in order to validate the current approach of the Virtual Wind Tunnel. Perhaps a certain trend can be derived that can be used to outweigh the large difference between CFD and the building codes. In the end one should proof that the results of a CFD calculation are profitable in comparison with the codes.
- The boundary conditions that are applied in this thesis on the building models and the ground surface are derived from the boundary conditions for the cube model, which were formulated by Van Nalta. With wall parameters a certain roughness can be given to the ground surface and the façades of the buildings. To model the reality accurately, further research is required to translate the structure of the real ground surface and the building façades into roughness parameters. The influence of different roughness parameters on the flow pattern around a building should be investigated as well.
- To simulate the flow pattern around specific small building parts accurately or to determine local forces, the computer capacity is too small as many cells are required for such calculations. It is recommended to adapt the grid locally in order to refine the mesh at critical places. Where possible, cells could be removed to obtain more cells for meshing certain important regions. However, one must be careful with local grid adaption in order to prevent large transitions between the local fine grid and the adjacent coarse grid. Large changes in cell size will have a negative influence on the accuracy of the solution, which can be larger than the profits of a locally finer grid. Other possibilities to save cells could be the reduction of the dimensions of the computational domain and the removal of the smallest objects in the research area.
- Additional research is required to investigate the influence of the radius of the central cylinder of the domain for the results of a CFD calculation. For single objects the distance between the object and the edge of the cylinder seems sufficient. However, when an entire built environment is placed in the cylinder, the radius of the cylinder must be increased considerably in order to provide a smooth transition between the mesh on the buildings and the edge of the cylinder. It is currently not known which ratio should be used between the radius of the research area and the radius of the central cylinder.

References

[1] Anderson, J.D. 2001, *Fundamentals of aerodynamics*, McGraw-Hill, New York
[2] Apsley, D. 2003, *The CFD Process*, The University of Manchester
[3] Arif, H. 1999, *Application of Computational Fluid Dynamics (CFD) to the modeling of flow in horizontal wells*, Stanford University
[4] Bernhardsen, T. 1999, *Geographic Information Systems, an introduction*, John Wiley & Sons, New York
[5] Blaha, M. & Rumbaugh, J. 2005, *Object-Oriented Modeling and Design with UML*, Pearson Prentice Hall, New Jersey
[6] Cauberg, J.J.M. 2003, *Toegepaste Bouwfysica*, Delft University of Technology
[7] Delaney, J. 1999, *Geographical Information Systems, an introduction*, Oxford University Press, New York
[8] Franke, J. 2004, *Recommendations on the use of CFD in Wind Engineering*, Cost Action C14
[9] Franke, J. 2004, *Recommendations on the use of CFD in predicting pedestrian wind environment*, Cost Action C14
[10] Geurts, C.P.W. & Staalduinen, van, P.C. 2001, *Windtunnelonderzoek altijd nuttig en soms noodzakelijk*, Bouwen met staal, vol. 163, december 2001, pp. 46-50
[11] Geurts, C.P.W. 2005, *Wind tunnel applications in building engineering*, Delft University of Technology Wind Laboratory
[12] Jiang, Z., Memarzadeh, F. & Weiran, X. 2004, *UVGI Interaction of Airborne Organisms*, ORF, Bethesda
[13] Kerklaan, R. & Jansen, H. 2003, *UML en ArchiCAD oefening*, Delft University of Technology
[14] Kim, S-E & Boysan, F. 1999, *Application of CFD to environmental flows*, Journal of wind engineering and industrial aerodynamics, vol. 81, pp. 145-158
[15] Kolbe, T.H., Groger, G. & Plumer, L. 2006, *3D City Models and their Potential for Emergency Response*, University of Bonn, Germany
[16] Nalta, van, R. 2004, *Computational Wind Engineering: Background, Approach and Validation*, Delft University of Technology
[17] NNI 2005, *Eurocode 1: Belastingen op constructies – Deel 1-4: Algemene belastingen – Windbelasting*, NEN-normenshop
[18] NNI 2001, *NEN 6702: Belastingen en vervormingen,* TGB 1990, NEN-normenshop
[19] Rogers, D.F. 2000, *An Introduction to NURBS with historical perspective*, Academic Press, London
[20] Sederberg, T.W. 2005, *An Introduction to B-Spline Curves*, Utah
[21] Schoonmaker, S.J. 2003, *The CAD guidebook*, Marcel Dekker Inc., New York
[22] Shaw, C.T. 1992, *Using Computational Fluid Dynamics*, Prentice Hall International (UK) Ltd, Hertfordshire
[23] Snijders, D.P. 2006, *Shaping the Virtual Wind Tunnel*, Delft University of Technology
[24] Thompson, J.F., Soni, B.K. & Weatherill, N.P. 1999, *Handbook of Grid Generation*, CRC Press, New York
[25] Tolman, F.P., Behesti, M.R. & Dado, E. 2001, *Bouwinformatica: Ontwerp en Constructie*, Delft University of Technology
[26] Toussaint, M.H. 2006, *Structural Design of a Timber Grid Shell*, Delft University of Technology
[27] Vambersky, J.N.J.A., Woudenberg, I.A.R. & Geurts, C.P.W. 2005, *CUR Aanbeveling 103: Windtunnelonderzoek*, Civieltechnisch Centrum Uitvoering Research en Regelgeving
[28] Ven, van der, K.W.M. & Uffelen, van, G.M. 2004, *Windtunnelonderzoek aan windbelasting op gebouwen*, Cement, no. 3, pp. 74-77
[29] Woudenberg, I.A.R. & Vambersky, J.N.J.A. 2003, *Windbelasting, hoogbouw en regelgeving*, Cement, no. 6, pp. 89-94

Internet pages

[1] Rhinoceros (s.d.) *Modeling tools for designers* [online]. Available from: http://www.rhino3d.com/ [Accessed 3 March 2006]
[2] TDN (s.d.) *Kadaster* [online]. Available from: http://www.tdn.nl/ [Accessed 3 March 2006]
[3] Reconstructivism (s.d.) *Reconstructivism* [online]. Available from: http://www.reconstructivism.net/ [Accessed 10 March 2006]
[4] Unesco (s.d.) *Training Module on GIS* [online]. Available from: http://gea.zvne.fer.hr/index.html/ [Accessed 7 April 2006]
[5] GISHydro (s.d.) *A GIS-Based Hydrologic Modeling Tool* [online]. Available from: http://www.gishydro.umd.edu/ [Accessed 7 April 2006]
[6] Moshplant (s.d.) *Bézier Curves* [online]. Available from: http://www.moshplant.com/direct-or/bezier/ [Accessed 14 April 2006]
[7] KNMI (s.d.) *De website van het KNMI* [online]. Available from: http://www.knmi.nl/ [Accessed 20 April 2006]
[8] U-Dispuut (s.d.) *Stichting Dispuut Utiliteitsbouw* [online]. Available from: http://www.udispuut.tudelft.nl/ [Accessed 26 April 2005]
[9] Fluent (s.d.) *CFD Flow Modeling Software & Solutions from Fluent* [online]. Available from: http://www.fluent.com/ [Accessed 27 April 2006]
[10] CADD (s.d.) *Computer Aided Detector Design* [online]. Available from: http://cadd.web.cern.ch/cadd/ [Accessed 1 May 2006]
[11] Geodan (s.d.) *Een wereld van geo-informatie* [online]. Available from: http://www.geodan.nl/nl/index.htm [Accessed 1 May 2006]
[12] Radboud Universtiteit Nijmegen (s.d.) *GISdesk* [online]. Available from: http://www.ru.nl/gisdesk/geo-data/algemeen_beschikbaar/ [Accessed 2 May 2006]
[13] AHN (s.d.) *Actueel Hoogtebestand Nederland* [online]. Available from: http://www.ahn.nl/ [Accessed 2 May 2006]
[14] McGill (s.d.) *School of Architecture* [online]. Available from http://www.mcgill.ca/architecture/ [Accessed 5 May 2006]
[15] TU Delft (s.d.) *TU Delft* [Online]. Available from: http://www.tudelft.nl [Accessed 12 May 2006]
[16] Pointwise (s.d.) *Reliable CFD Meshing and Grid Generation from Pointwise* [Online]. Available from: http://www.gridgen.com/ [Accessed 15 May 2006]
[17] Object Management Group (s.d.) *UML* [Online]. Available from: http://www.uml.org/ [Accessed 19 July 2006]
[18] Graphisoft (s.d.) *ArchiCAD* [Online]. Available from: http://www.graphisoft.com/ [Accessed 19 July 2006]
[19] Construsoft (s.d.) *Tekla Structures* [Online]. Available from: http://www.construsoft.nl/ [Accessed 20 July 2006]
[20] Wikipedia (s.d.) *Remote Sensing* [Online]. Available from: http://en.wikipedia.org/wiki/Remote_sensing/ Accessed 20 July 2006]
[21] Paul Bourke (1997) *Intersection of a line and a facet* [Online]. Available from: http://local.wasp.uwa.edu.au/~pbourke/geometry/linefacet/ [Accessed 11 August 2006]
[22] SDC Publications (s.d.) *Autodesk Architectural Desktop* [Online]. Available from: http://www.schroff1.com/revit/1585030996.htm [Accessed 13 August 2006]
[23] ONL (s.d.) *Oosterhuis_Lénárd* [Online]. Available from: http://www.oosterhuis.nl/ [Accessed 23 August 2006]
[24] Tecplot (s.d.) *CFD Post-Processing, Plotting, Graphing & Data Visualization Software* [Online]. Available from: http://www.tecplot.com/ [Accessed 26 September 2006]

Appendices

Appendix A: Additional information

In this appendix some additional information on the theory about CFD, GIS and CAD is given.

A.1 Fluids in motion

The basic equations describing fluid flow are derived from the mass, momentum and energy balance on a fluid element (Shaw, [22]). Three fundamental physical principles to solve the basic equations were already formulated in the 18th century by the well-known scientist Leonhard Euler:

▪ Mass is conserved;	No mass is created or destroyed in a fluid flow This principle is referred to as the *Continuity equation*
▪ Momentum is conserved;	Momentum is defined as the mass of a particle multiplied by its velocity. The amount of momentum is constant in a flow This principle is referred to as the *Momentum equation*
▪ Energy is conserved;	No energy is created or destroyed in a fluid flow This principle is referred to as the *Energy equation*

Combining these equations leads to the governing equations for fluid motion. According to Anderson [1] there are two main approaches for fluid modeling, with their own form of the governing equations:

▪ The Langragian approach:	the fluid motion is analyzed using an infinitesimal fluid element or control volume that is moving along a streamline
▪ The Eulerian approach:	the fluid motion is analyzed using a finite control volume V that is fixed in space. The volume is bounded by control surface S and the fluid is moving through the volume

On the next pages the three fundamental principles are evaluated inside a fixed control volume using the Eulerian approach, as in most CFD software packages the flow fields are computed in the Eulerian reference system (Jiang et all, [12]).

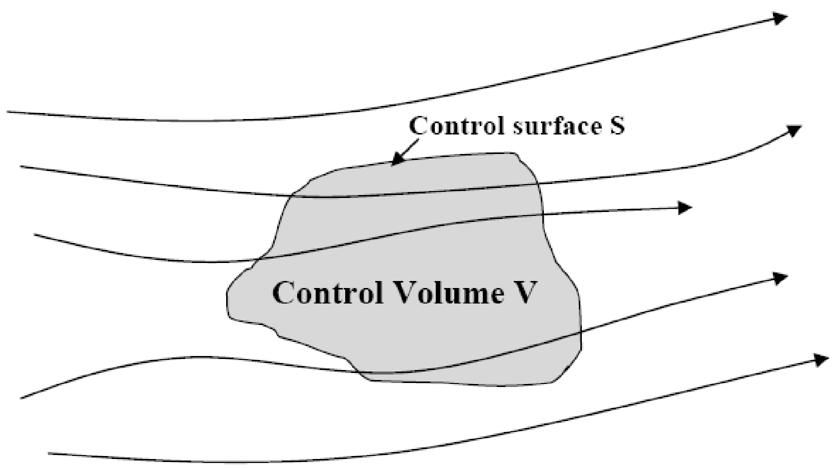


Figure A.1: Control volume fixed in space with the fluid moving through it, Eulerian approach (Anderson, [1])

Conservation of mass

The conservation law of mass is based upon the principle that the net mass flow out of the control volume through the surface is equal to the time rate of mass decrease inside the control volume. When this condition is satisfied, mass is conserved. The net mass flow out of the control volume is the product of the fluid density, the component of fluid perpendicular to the surface and the area of the surface and is given by the following equation:

$$\oiint_S \rho V \cdot dS \qquad \text{(A.1)}$$

The dot product of V and dS indicates that the velocity is taken perpendicular to the control surface. The total mass inside the control volume is:

$$\iiint_V \rho dV \qquad \text{(A.2)}$$

Because the size of the control volume remains constant in time, the mass decrease inside the volume is equal to the time-derived density, integrated over the volume:

$$-\frac{\partial}{\partial t}\iiint_V \rho dV \qquad \text{(A.3)}$$

The net mass flow out of the control volume must be equal to the mass decrease inside the volume. Combining these two equations leads to the following equation for the conservation of mass:

$$\oiint_S \rho V \cdot dS + \frac{\partial}{\partial t}\iiint_V \rho dV = 0 \qquad \text{(A.4)}$$

This equation is also known as the continuity equation.

Conservation of momentum

To create a change in momentum of a single particle, a force is required. As momentum is defined as the mass of a particle multiplied by its velocity, it follows from the conservation law of mass that a change in velocity is needed to change the momentum, as mass is constant. A change in velocity leads to an acceleration or deceleration of the particle. The required force to create a change in momentum is equal to the mass of a particle multiplied by its acceleration or deceleration and can be written as *F = m·a*. This equation is also known as Newton's second law. The equation *F = m·a* is a vector relation and can be split in three directions along the x, y and z-axis. In the following the expression for F in x-direction will be derived using the forces and pressures on the infinitesimal fluid element shown in Figure A.2.

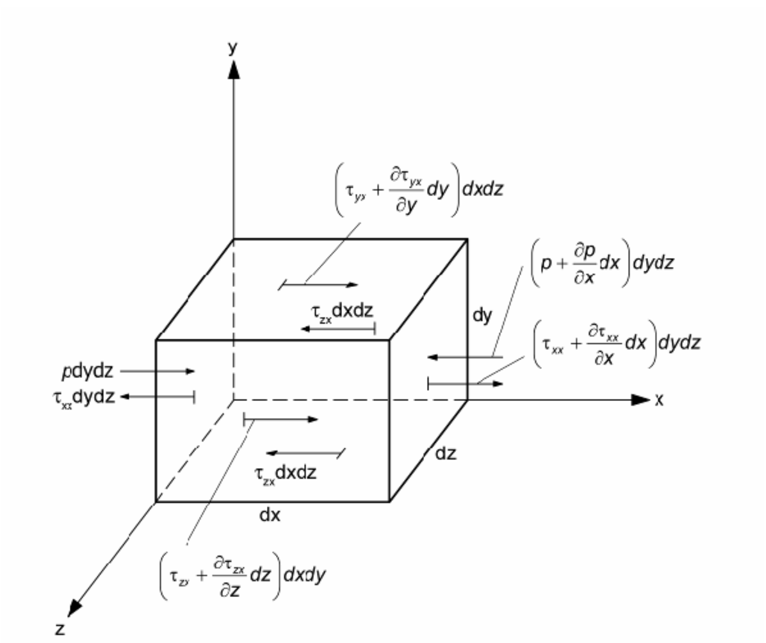

y
$\left(\tau_{yx} + \frac{\partial \tau_{yx}}{\partial y} dy\right) dxdz$
$\left(p + \frac{\partial p}{\partial x} dx\right) dydz$
$\tau_{zx}dxdz$
dy
$\left(\tau_{xx} + \frac{\partial \tau_{xx}}{\partial x} dx\right) dydz$
$p dydz$
$\tau_{xx}dydz$
x
$\tau_{zx}dxdz$
dz
dx
$\left(\tau_{zx} + \frac{\partial \tau_{zx}}{\partial z} dz\right) dxdy$
z

Figure A.2: Shear forces and pressures in x-direction on an infinitesimal fluid element (Anderson, [1])

It follows that the total force in x-direction is given by:

$$F_x = \left(-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} \right) dxdydz + \rho f_x dxdydz \tag{A.5}$$

Herein is f_x the x-component of the body force per unit mass acting on the fluid element.

From Newton's second law it follows that the net force exerted on the control volume is equal to the mass of a particle times its acceleration. Acceleration is the time rate of change of the velocity and is in x-direction given by:

$$a_x = Du / Dt \tag{A.6}$$

The mass of an infinitesimal fluid element is constant and given by:

$$m = \rho dxdydz \tag{A.7}$$

From Equations (A.6) and (A.7) it follows for the product of mass and acceleration:

$$ma_x = \rho \frac{Du}{Dt} dxdydz \tag{A.8}$$

Combining Equations (A.5) and (A.8) results in the x-component of the momentum equation:

$$\rho \frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x \tag{A.9}$$

The y and z-components can be derived in a similar way:

$$\rho \frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_y \tag{A.10}$$

$$\rho \frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_z \tag{A.11}$$

These momentum equations are called Navier-Stokes equations. However, in CFD literature, the Navier-Stokes equations usually refer to the complete set of mass, momentum and energy equations. Finally the magnitude of the shear stresses on the fluid element of Figure A.2 can be defined. In Newtonian fluids, such as air, the shear stress is proportional to the rate of deformation. Herein is the viscosity μ the proportion factor:

$$\tau_x = \mu \frac{\partial u}{\partial y} \tag{A.12}$$

The shear stresses for Newtonian fluids are defined as:

$$\tau_{xx} = \lambda \nabla \cdot V + 2\mu \frac{\partial u}{\partial x} \tag{A.13}$$

$$\tau_{yy} = \lambda \nabla \cdot V + 2\mu \frac{\partial v}{\partial y} \tag{A.14}$$

$$\tau_{zz} = \lambda \nabla \cdot V + 2\mu \frac{\partial w}{\partial z} \tag{A.15}$$

$$\tau_{xy} = \tau_{yx} = \mu \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) \tag{A.16}$$

$$\tau_{xz} = \tau_{zx} = \mu \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \right) \tag{A.17}$$

$$\tau_{yz} = \tau_{zy} = \mu \left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} \right) \tag{A.18}$$

The stresses $\tau_{xy}, \tau_{xz}, \tau_{yx}, \tau_{yz}, \tau_{zx}, \tau_{zy}$ are called the *viscous shear stresses* and the stresses $\tau_{xx}, \tau_{yy}, \tau_{zz}$ are called the *viscous bulk stresses*. Bulk stresses cause compression or extension of the fluid. The bulk viscosity coefficient is given by:

$$\lambda = -\frac{2}{3}\mu \tag{A.19}$$

Conservation of energy
The conservation law of energy is based upon the principle that energy can only change in form; it can not be created or destroyed. Every change in energy over time inside the control volume must result from an amount of heat added to the flow from the surroundings and an amount of work that is performed on the fluid inside the control volume. The energy, heat and work flux have to be in equilibrium:

$$\frac{\partial E}{\partial t}=\frac{\partial Q}{\partial t}+\frac{\partial W}{\partial t} \qquad \text{(A.20)}$$

The rate of heat addition is determined by the rate of volumetric heating and the rate of heat addition to the control volume due to viscous effects:

$$\frac{\partial Q}{\partial t}=\frac{\partial}{\partial t}\iiint_V q\rho dV+\frac{\partial Q_{viscous}}{\partial t} \qquad \text{(A.21)}$$

Herein is q the rate of heat addition per unit mass and $Q_{viscous}$ the total of viscous effects. The amount of work that is performed is determined by the amount of work due to pressure, body forces and shear stress on the control surface:

$$\frac{\partial W}{\partial t}=\iiint_V \rho\left(f\cdot U\right)dV-\iint_S pU\cdot dS+\frac{\partial W_{viscous}}{\partial t} \qquad \text{(A.22)}$$

Herein is $W_{viscous}$ the total of viscous effects. The rate of change of total energy is the rate of energy flow across the control surface and the time rate of change of energy in the volume:

$$\frac{\partial E}{\partial t}=\frac{\partial}{\partial t}\iiint_V \rho\left(e+\frac{U^2}{2}\right)dV+\iint_S\left(\rho U\cdot dS\right)\left(e+\frac{U^2}{2}\right) \qquad \text{(A.23)}$$

When Equations (A.21), (A.22) and (A.23) are combined, the following energy equation is obtained:

$$\frac{\partial}{\partial t}\iiint_V \rho\left(e+\frac{U^2}{2}\right)dV+\iint_S \rho\left(e+\frac{U^2}{2}\right)U\cdot dS=\frac{\partial}{\partial t}\iiint_V q\rho dV+\frac{\partial Q_{viscous}}{\partial t}-\iint_S pU\cdot dS$$

$$+\iiint_V \rho(f\cdot U)dV+\frac{\partial W_{viscous}}{\partial t} \qquad \text{(A.24)}$$

Navier-Stokes equations

The continuity, momentum and energy equation are the basic equations of fluid dynamics. For wind engineering, the fluid is considered incompressible due to the relatively low speed of wind. As a consequence of the incompressibility the density ρ and viscosity μ of the fluid are constant and the equations describing the flow can be simplified. With constant mass and density, the change of volume of a fluid element in time is zero. Another consequence of the incompressibility and constant density is that the continuity and momentum equations are sufficient to describe the flow. The energy equation does not have to be taken into account. The simplified set equations describing the flow are known as the Navier-Stokes equations. When Equations (A.13), (A.16) and (A.17) for the stresses are substituted in the momentum Equation (A.9), it follows for the momentum equation in x-direction:

$$\rho\frac{Du}{Dt}=-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}(\lambda\nabla\cdot V)+\mu\left(2\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial z^2}+\frac{\partial^2 v}{\partial x\partial y}+\frac{\partial^2 w}{\partial x\partial z}\right)+\rho f_x \tag{A.25}$$

The Continuity Equation (A.4) can be written as:

$$\frac{D\rho}{Dt}+\rho\nabla\cdot V=0 \tag{A.26}$$

As a consequence of the constant density, the continuity equation can be simplified to:

$$\nabla\cdot V=\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0 \tag{A.27}$$

$$\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)=0 \tag{A.28}$$

From Equations (A.25), (A.27) and (A.28) it follows that the momentum equation in x, y and z-direction can be simplified to:

$$\frac{Du}{Dt}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+f_x \tag{A.29}$$

$$\frac{Dv}{Dt}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)+f_y \tag{A.30}$$

$$\frac{Dw}{Dt}=-\frac{1}{\rho}\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right)+f_z \tag{A.31}$$

These three equations are known as the Navier-Stokes equations for an incompressible fluid.

In the theory about fluids in motion, no attention was paid to turbulence so far. Turbulence however leads to rapid velocity fluctuations in both space and time. Unfortunately, the amount of calculation effort required to capture both time and space variations of the variables is extremely large. This has led to the concept of turbulence modeling. The subject of turbulence modeling is further discussed on the next page.

A.2 Turbulence Modeling

Turbulence can be described as the variation of the wind speed around its mean. Due to this unsteadiness the flow hardly remains at equilibrium. To model the turbulence behaviour of the flow a turbulence model is needed. With a turbulence model the Navier-Stokes equations are simplified with a new set of equations that model the unsteadiness of the flow. There are several turbulence models available ranging from simple algebraic models to highly sophisticated models like Reynolds-stress transport models (RSTM). The reliability of CFD solutions for turbulent flows depends on the turbulence model used. The most important methods to deal with turbulence behaviour are:

Direct Numerical Simulation (DNS)
The most accurate method to deal with turbulence is DNS. All scales in a turbulent flow are simulated without any modeling of turbulence. The grid must be fine enough to solve the smallest scales. However, as the flows encountered in urban areas involve scales ranging from small gusts to vortices comparable to the scale of the building or structure itself, the computational demand for this method is too high for use in wind engineering (Franke, [9]).

Large Eddy Simulation (LES)
To reduce the computational demands the Navier-Stokes equations can be simplified by filtering the small eddies. This filtering is also knows as *time averaging* and leads to the LES method. By using a grid that is too coarse to capture the small scales of the flow, only the large eddies are taken into account. The result of filtering the small eddies is a modified set of Navier-Stokes equations forming a turbulence model, which are computationally less expensive to solve. The LES method gives good results comparable with experimental data, but the computational effort is still too high for use in a design application. However, rapid evolution of CPU hardware will surely overcome this restriction in the near future.

Reynolds Averaged Navier-Stokes (RANS)
The generally used method for the computation of turbulent flow is the Reynolds Averaged Navier-Stokes (RANS) method. All scales of turbulence are modeled in this method. The RANS equations are derived by splitting the velocity into a mean and a fluctuating component. The Navier-Stokes equations are then averaged in time over all scales of turbulence to directly produce the solution of the flow variables. Averaging results in an additional set of six unknown terms in the momentum conservation equations that are known as *Reynold stresses* (Arif, [3]). The additional equations that are derived to determine the Reynold stresses are known as the turbulence model. If these six terms are ignored, the time-averaged momentum equations are the same as the original momentum equations. RANS modeling is much faster than LES and DNS but it is assumed to be unfit when high accuracy is required.

The derivation of the additional equations requires an approximation due to the complexities of the turbulence. This has resulted in a variety of turbulence models for the RANS equations. The available RANS turbulence models range from a large number of variants of the *κ-ε* models to Reynolds stress transport models. The original *κ-ε* model seems to be the most widely used turbulence model in building aerodynamics for their simplicity. However, the conventional *κ-ε* models have some drawbacks. Analysis of the flow field around a bluff body with the standard *κ-ε* model results in the overproduction of turbulence energy around the frontal corners of the bluff body. As a result, the predicted values of velocity or pressure differ from the experimental results. Many attempts have been made to revise the *κ-ε* model and several new models have been proposed, under which the *realizable κ-ε* model and the *RNG κ-ε* model. The prediction accuracy of these new models has significantly improved.

A.3 Historical overview of GIS

GIS has evolved out of a long tradition of map making. The earliest known maps were drawn on parchment to show the gold mines at Coptes (1292-1225 B.C.). Around 300 B.C. the Greeks acquired cartographic skills and made the first realistic maps. These maps were drawn almost exclusively to facilitate commercial sea voyages.

Around 1400 A.D. several developments in cartography were made in Europe. The invention of the printing press made maps more widely available. In stead of copying by hand, many identical copies could be produced now. Besides, more and more exploration voyages were undertaken, which increased knowledge of the world and the interest in map making. During the 16th century, many maps were produced from the ever-increasing information brought back by navigators and explorers. From now on, the accuracy of the maps was greatly increased by more precise determinations of the size and shape of the earth.

Figure A.3: Ptolemy's map of the world (Internet, [1])

As in the 18th century the prosperous countries evolved to more organized societies, the needs for geographic information increased. Because maps became also important in military operations, military agencies became the leading makers for all civilian and military maps.

In history, geographical information was usually used for trade and exploration expeditions. When in the 19th century new infrastructures were evolved, such as roads, railways and gas and water supplies, the demand for geographical information arose. The exact location of cities, mountains, lakes and rivers became very important.

With the technological innovations in the 20th century the progress of map making became also more developed. Especially the development of satellites with photographic equipment increased the speed and accuracy of map making. With the introduction of the computer geographical data could now also be automatically processed. This has led to a wide variety of Geographic Information Systems.

A.4 Recent developments in GIS

Nowadays GIS is used in many technologies. Compared with a couple of years ago, acquiring data for use in a GIS is no longer a major problem. Trends in GIS data show that remote sensing will become an important source of GIS data in the nearby future. In the broadest sense, remote sensing is the measurement of information of an object by a recording device that is not in direct contact with the object (Internet, [20]). In practice, remote sensing is the measurement of any device for gathering information about the environment. The devices used for GIS applications are usually aircrafts and satellites. Another trend shows that the exchange of GIS data will become more common and has been facilitated by exchange standards.

Another major development of the last two years is the growing awareness of GIS in the general public. The tsunami in South Asia in 2004 and hurricane Katrina in the United States in 2005 showed how important GIS is in order to protect lives. When viewing satellite imagery and comparing the changes of land cover before and after the disasters, very important information can be obtained about nature's behaviour. Also rescue efforts are heavily relying on GIS technology.

The integration of GIS and the internet is another recent development. In the past, static pictures were used to present maps of a certain area on the internet. Nowadays it is possible to use dynamic pictures where the user itself can decide what has to be mapped and how it must be presented. It is possible to chart neighbourhoods on the basis of population, real estate, income distribution and busyness. Other applications are websites where a whole environment can be mapped on the basis of a specified address or postal code. The most famous integration of GIS and internet is Google Earth. With this technology it is possible to take a look at each arbitrary place on earth by means of satellite and air photos. An additional option is the possibility to plot certain data on a map, to chart crime numbers for example. Another option is to construct 3D buildings with a special plug-in, to create a 3D environment of a certain area. GIS is also integrated in mobile internet nowadays. With this technology, actual traffic information can be asked for and it is even possible to search for the nearest parking garage with a free place for example. When a route planner is integrated, the driver can be guided to the garage also.

In the nearby future, other applications of GIS are expected to be introduced. An example is the registration of property. Until recently, maps were only presented in two dimensions. Because multiple usage of space is often necessary nowadays, flat maps are not sufficient anymore. The registration of property demands a three dimensional approach when properties of various owners are placed on each other. If for example a tunnel is built underneath a certain district with another owner, the properties can not be registered under the same owner. Research is done at the moment to separate the several properties and its various owners by registering property in three dimensions.

A.5 Industry Foundation Classes

Standardization of information models and information exchange is very important to provide interchangeability between the software used by all building project participants. In 1995 Bentley and Autodesk formed the International Alliance for Interoperability (IAI) to develop a method to exchange a complete, accurate building data model from one computer application to another, with no loss of information (Tolman et al, [25]). The IAI developed the Industry Foundation Classes (IFC), which are data elements that represent the parts of a building or elements of the building process and contain relevant information about those parts. IFC's are used by computer applications to provide a computer readable project model that contains all the information of the building parts and process elements and their relationships. The goal is that all project participants exchange information with this project model, not with each other. The project model provides an object-oriented database of the information shared among project participants and continues to grow as the project goes through all phases, from design and construction to operation.

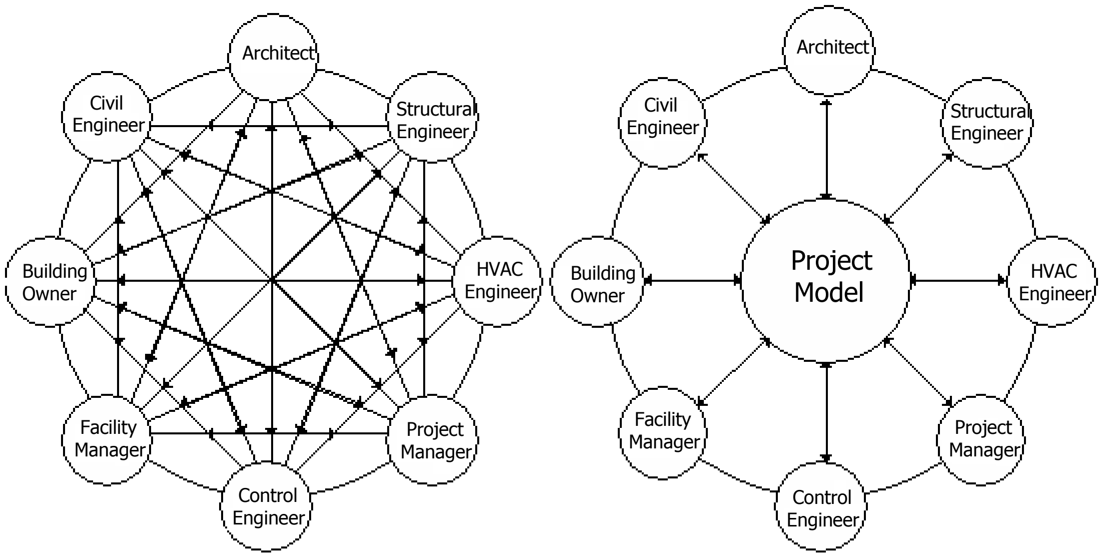


Figure A.4: Exchange of information in the past and future (Tolman et al, [25])

Appendix B: TDN-Code

The Top10Vector dataset that is used to generate a 3D model of the environment is composed of closed polygons that represent the various elements on the earth's surface. Every element type has its own layer with a unique code, wherein all polygons are subdivided. The several layers can be turned on and off separately. This appendix gives a list of all layers in the Top10Vector dataset, the so-called TDN code.

1 Bebouwing

1.1	Gebouw, huis	1003
1.2	Bebouwd gebied	1013
1.3	Hoogbouw	1033
1.4	Muur	1043
1.5	Kassen	1073
1.6	Opslagtank	1083
1.7	Politiebureau	1103
1.8	Postkantoor	1113
1.9	Gemeentehuis	1123
1.10	Hospitaal	1213
1.11	Markant Object	1303
1.12	Jaknikker	1313
1.13	Zuiveringsinstallatie	1373
1.14	Paal	1413
1.15	Vlampijp	1423
1.16	Schietbaan	1453
1.17	Seinmast	1463
1.18	Energiemolen	1503
1.19	Windmolen	1513
1.20	Watermolen	1533
1.21	Windmolentje	1543
1.22	Gemaal	1553
1.23	Religieus gebouw	1703
1.24	Kapel	1753
1.25	Kruis	1763
1.26	Hunebed	1773
1.27	Gedenkteken	1783
1.28	Toren	1803
1.29	Watertoren	1823
1.30	Vuurtoren	1853

2 Wegen cat 2

2.1	Autosnelweg	2003
2.2	Lokale autoweg	2083
2.3	Autoweg als verbindingsroute	2103
2.4	Verbindingsroute met gescheiden rijbanen	2203
2.5	Verbindingsroute > 7 m	2303
2.6	Autoweg met gescheiden rijbanen als verbindingsroute	2343
2.7	Verbindingsroute > 4 m	2403
2.8	Lokale autoweg met gescheiden rijbanen	2443
2.9	Verbindingsroute > 2 m	2503
2.10	Autoweg met gescheiden rijbanen als overig aanbev. r.	2803
2.11	Lokale weg met gescheiden rijbanen	2873
2.12	Autoweg als overig aanbevolen route	2903

3 Wegen cat 3

3.1	Overig aanbevolen route met gescheiden rijbanen	3003
3.2	Overig aanbevolen route > 7 m	3103
3.3	Lokale weg > 7 m	3143
3.4	Overig aanbevolen route > 4 m	3203
3.5	Lokale weg > 4 m	3243
3.6	Overig aanbevolen route > 2 m	3303
3.7	Lokale weg > 2 m	3343
3.8	Overige weg > 2 m	3403
3.9	Gedeeltelijk verhard > 2 m	3413
3.10	Onverharde weg > 2 meter	3433
3.11	Overkluizing	3453
3.12	Passage	3463
3.13	Voetgangersgebied	3473
3.14	Voetpad < 2 m	3523
3.15	Straat	3533
3.16	Fietspad > 2 m	3603
3.17	Fietspad < 2 m	3623
3.18	Met fietspad	3633
3.19	Pad	3643
3.20	Pontveer	3663
3.21	Voetveer	3673
3.22	Veerdienst	3683
3.23	Brug	3713
3.24	Pijlers van een brug	3733
3.25	Vonder	3743
3.26	Beweegbaar brugdeel	3763
3.27	Tankstation	3813
3.28	Parkeerfaciliteit	3833
3.29	Wegafsluiting	3853
3.30	Kilometerpaal	3863
3.31	Wegwijzer	3873
3.32	Afritnummerblok	3893
3.33	Parkeerterrein	3903
3.34	Wegnummerblok A-route	3913
3.35	Wegnummerblok N-route	3923
3.36	Wegnummerblok E-route	3933
3.37	Rijstrookcirkel	3953

4 Spoor

4.1	Enkelspoor	4003
4.2	Dubbelspoor	4043
4.3	Driespoor	4103
4.4	Vierspoor	4143
4.5	Tramroute	4233
4.6	Smalspoor	4253
4.7	Metro bovengronds	4263
4.8	Station	4303
4.9	Metro / sneltramstation	4333
4.10	Laadperron	4353
4.11	Kilometerpaal spoorweg	4393
4.12	Kabelbaanmast	4403
4.13	Kabelbaan	4413
4.14	Zend- / ontvangstmast	4733
4.15	Hoogspanningsmast	4803
4.16	Hoogspanningsleiding	4813

5 Vegetatie

5.1	Boom	5003
5.2	Loofbos	5023
5.3	Naaldbos	5053
5.4	Gemengd bos	5063
5.5	Griend	5073
5.6	Populierenopstand	5083
5.7	Heg	5113
5.8	Bomenrij	5123
5.9	Bouwland	5203
5.10	Grasland	5213
5.11	Boomgaard	5223
5.12	Kwekerij	5233
5.13	Heide	5243
5.14	Zand	5253
5.15	Overig bodemgebruik	5263
5.16	Begraafplaats	5303
5.17	Fruitkwekerij	5313
5.18	Contour tbv erven	5333
5.19	Hulplijn (afsluiter)	5393
5.20	Contour	5403
5.21	Damlijn	5453
5.22	Hulplijn (eilandverbinder)	5463

6 Hydrografie

6.1	Greppel	6003
6.2	Sloot < 3 m	6013
6.3	Sloot tussen 3 en 6 m	6023
6.4	Water	6103
6.5	Oeverlijn	6113
6.6	Hoogwaterlijn	6203
6.7	Laagwaterlijn	6213
6.8	Dieptelijn	6223
6.9	Dieptepunt	6233
6.10	Steenglooiing	6293
6.11	Draslanden	6303
6.12	Riet	6313
6.13	Paalwerk	6393
6.14	Aanlegsteiger > 2 m	6513
6.15	Aanlegsteiger < 2 m	6523
6.16	Dok	6543
6.17	Dukdalf	6573
6.18	Kilometerraaipaal	6613
6.19	Kilometerraaibord	6623
6.20	Peilschaal	6633
6.21	Baak	6643
6.22	Lichtopstand	6653
6.23	Lichttoren	6673
6.24	Sluisdeur	6723
6.25	Stuw	6743
6.26	Duiker	6763
6.27	Grondduiker	6773
6.28	Dam	6793
6.29	Stroompijl groot	6813
6.30	Stroompijl klein	6823
6.31	Eb en vloedpijl	6833

7 Relief

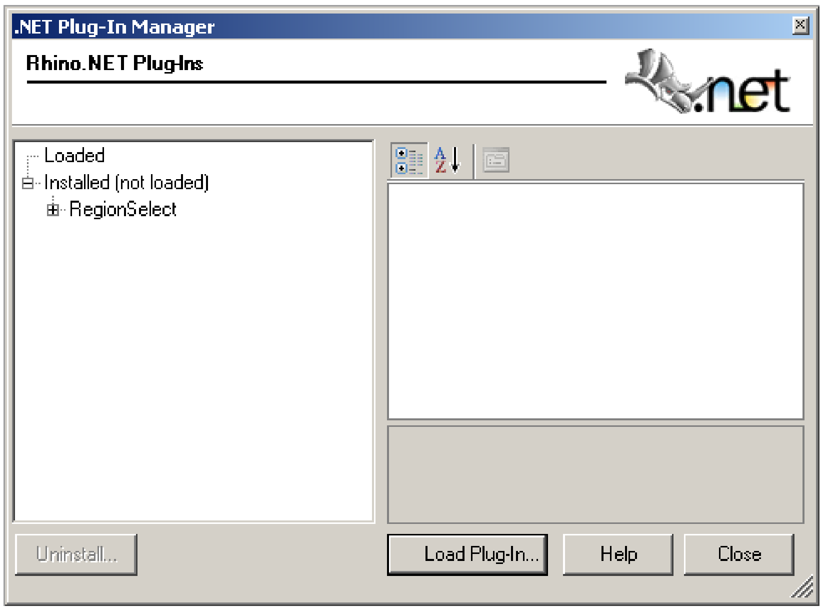
7.1	Dijk > 2,5 m	7103
7.2	Dijk 1 – 2,5 m	7113
7.3	Boezemkade	7143
7.4	Wal	7153
7.5	Geluidswering	7163
7.6	Ingraving	7203
7.7	Hoogteverschil	7223
7.8	Aardrand	7253
7.9	Recht omhoog	7263
7.10	Recht omlaag	7273
7.11	Schuin omhoog	7283
7.12	Schuin omlaag	7293

8 Grenzen

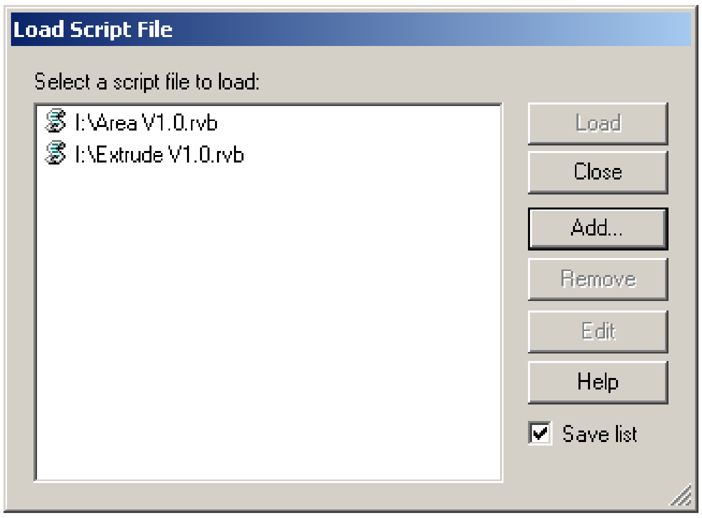
8.1	Grenspunt	8123
8.2	Hek	8193
8.3	Camping	8783
8.4	Sportcomplex	8893

Appendix C: Script to generate the 2D research area

In this appendix the script that generates the 2D research area from the Top10Vector and AHN datasets is discussed in detail. The script uses an external plug-in for Rhinoceros that automatically select the points that lie in any closed curve. The plug-in is called *RegionSelect* and can be downloaded from http://www.reconstructivism.net/. The plug-in is a so-called DotNET plug-in, what means that it is written on the VisualBasic.NET platform. In order to run these RhinoDotNET plug-ins, at least V1.1 of the Microsoft.NET framework has to be installed. This is free downloadable from the Microsoft website: http://www.microsoft.com/. Once the .NET framework is installed, the RhinoDotNetManager that runs DotNET plug-ins needs yet to be installed. This in itself is a plug-in for Rhinoceros that can be downloaded from http://www2.rhino3d.com/wip_plugins/. Once the DotNetManager is installed, it can be run by entering *_DotNetManager* inside the Rhinoceros command line. The manager can add the *RegionSelect* plug-in to Rhino.



Before running the script to generate the research area, the user has to import both the Top10Vector and AHN datasets for a certain location in Rhinoceros. The script can be loaded by entering *LoadScript* in the command line. A new window is opened with which the scripts can be loaded. With the *Add...* button the various script files can be added to the program.



On the basis of the script code, the tool will now extensively be discussed. The script starts with the *Option Explicit* command. With this command Rhino or any other program that runs the script, will check if all variables are properly defined. If this command is not added, it will be very hard to find errors in the script. Before the script can do anything, it needs a subroutine. The subroutine controls all the code that is needed to perform the tasks of the script. In this case the subroutine is called *Area*. In the following step the variable and constant terms are declared that will be used in the subroutine.

```
Option Explicit

Sub Area()

Dim radius, r
Dim arrCenter, arrEnd(2)
Dim arrLayers
Dim strLayer
Dim iAllObj, i
Dim arrPoints, arrPoint
Dim arrPnts, arrPnt
Dim dx, dy, d, dx2, dy2, d2, dmax
Dim x1, x2, y1, y2
Dim xmin, xmax, ymin, ymax
Dim allBuildings, building
Dim arrObjects, Obj
Dim Circ, Crc
Dim invert
Dim extrPolygons, p
Dim rectangle, Rect

Const extHeight = -2
```

Because the height layers of the AHN dataset contain points in stead of comma's to separate the decimals, the Visual Basic script can give errors when it is not set to the right region settings. With the setLocale command the local ID is set to English – United Kingdom.

```
setLocale(2057)
```

After defining the variables and local settings, the script starts with unselecting all eventually selected objects. Then it asks the user to give a radius of the research area in meters. The radius is 300 m by default, but the user can enter any radius. After that the user is asked to pick a location in the research area where the building of interest will arise. He can do this by simply mouse-clicking in the model.

```
Rhino.UnselectAllObjects
radius = Rhino.Realbox("Give radius of the research area
                       (meters)",300.0)
Rhino.MessageBox ("Pick center location of the new building")
arrCenter = Rhino.GetPoint
```

When the user has picked a center location, the script will add a new layer to the model called *Center_point* and makes it current. Now a new point will be created with the coordinates of the center location that was given by the user. Other tools that will use this model now always can find the center location of the new building. After the point is created, its layer is turned off so it cannot be accidentally removed anymore. The current layer is set back to *default*.

```
Rhino.AddLayer("Center_point")
Rhino.CurrentLayer("Center_point")
If IsArray(arrCenter) Then
  Rhino.AddPoint arrCenter
End If
Rhino.CurrentLayer "Default"
Rhino.Layermode ("Center_point") ,1
```

Because only the polygons that represent buildings will be extruded, the layers that contain other elements, like roads, water and vegetation will be turned off. Nevertheless, if one also wants to model other elements, the numbers of the layers that are turned off have to be changed here.

```
arrLayers = Rhino.LayerNames
If IsArray(arrLayers) Then
  For strLayer = 2000 to 7000
  Rhino.LayerMode strLayer , 1
  Next
End If
```

For the remaining polygons is checked if it lies at the in- or outside of the circular research area. With the *Rhino.CurvePoints* command the control points that lie on all corners of a polygon are obtained. For each control point will now be checked if its distance to the new building's center location is larger or smaller than the user-defined radius of the research area. If the distance of one or more control points of the polygon is smaller than the radius, the polygon lies completely or partly in the research area. The polygon must remain and it will be selected. If for all control points of a polygon the distance to the center location is larger than the radius, the polygon will not be selected. So, in conclusion, all polygons that are located at the boundary of the research area are selected, just as the polygons that lie completely in the research area.

```
allBuildings = Rhino.NormalObjects
If IsArray(allBuildings) Then
  For Each building in allBuildings
    If Rhino.IsCurve(building) Then
      arrPoints = Rhino.CurvePoints(building)
      If IsArray(arrPoints) Then
        For Each arrPoint in arrPoints
          dx = arrPoint(0) - arrCenter(0)
          dy = arrPoint(1) - arrCenter(1)
          d = Sqr((dx)^2 + (dy)^2)
          If d < radius Then
            Rhino.SelectObject(building)
          End If
        Next
      End If
    End If
  Next
End If
```

Subsequently a new layer is added to the model, called *ExtrPolygons*. The layers of the selected polygons are changed to this new layer. The polygons stay selected.

```
Rhino.AddLayer("ExtrPolygons")
Rhino.CurrentLayer("ExtrPolygons")
Rhino.Command("_ChangeToCurrentLayer")
```

For each polygon of the new layer, the script will select all height points that are surrounded by the polygons, using the *Region_Select* plug-in. So, all polygons that must remain and the necessary height points are now selected. With the *Rhino.InvertSelectedObjects* the selection can be inverted. So all points that are not surrounded by a polygon and all polygons that do not lie partly or completely in the research area, are now selected. The selected objects are then deleted.

```
extrPolygons = Rhino.ObjectsByLayer("ExtrPolygons")
For p = 0 to UBound(extrPolygons)
  Rhino.Command "_RegionSelect '_SelID " & extrPolygons(p)
Next
Rhino.SelectObjects(extrPolygons)
invert = Rhino.InvertSelectedObjects
Rhino.DeleteObjects invert
```

What remains are the polygons with their height points that lie in the region of interest. Because some polygons lie at the boundary of the circular research area, the circular area can not be taken as basis for the ground surface anymore. The actual area is larger, so a new plane has to be created. With the *Rhino.CurvePoints* command the control points at the corners are again obtained for the remaining polygons. The minimum and maximum x and y value of all control points are then determined.

```
Rhino.SelectObjects(extrPolygons)
If IsArray(extrPolygons) Then
  dmax = 0
  xmin = 0
  xmax = 0
  ymin = 0
  ymax = 0
  For Each Obj in extrPolygons
    If Rhino.IsCurve(Obj) Then
      arrPnts = Rhino.CurvePoints(Obj)
      If IsArray(arrPnts) Then
        For Each arrPnt in arrPnts
          dx2 = arrPnt(0) - arrCenter(0)
          dy2 = arrPnt(1) - arrCenter(1)
          If dx2 < 0 Then
            If dx2 < x1 Then
              x1 = dx2
              xmin = arrPnt(0)
            End If
          End If
          If dx2 > 0 Then
            If dx2 > x2 Then
              x2 = dx2
              xmax = arrPnt(0)
            End If
          End If
          If dy2 < 0 Then
            If dy2 < y1 Then
              y1 = dy2
              ymin = arrPnt(1)
            End If
          End If
          If dy2 > 0 Then
            If dy2 > y2 Then
              y2 = dy2
              ymax = arrPnt(1)
            End If
          End If
        Next
      End If
    End If
  Next
End If
```

From the minimum and maximum x and y values of the control points the radius of a new circle can be calculated that fits all polygons. First a new layer is added to the model, *Circle*. The new layer is made current and the circle is created. If some polygons are then still selected, they are unselected with the *Rhino.UnselectAllObjects* command.

```
Rhino.AddLayer "Circle", RGB(105, 105, 105)
Rhino.CurrentLayer("Circle")
r = 1/2*(sqr((CStr(xmax)-CStr(xmin))^2 + (CStr(ymax)-CStr(ymin))^2))
Rhino.AddCircle arrCenter, r
Rhino.UnselectAllObjects
```

The circular research area is then moved to the origin of the coordinate system. Because the computational domain that will be constructed around the research area to perform the CFD calculations is also generated around the origin, the area is directly placed right when imported in the domain. First all objects of the model are placed in the *iAllObj* array. Then the point is defined where the model has to be moved to. Finally the model is moved from the center location of the circular area to the origin of the coordinate system.

```
iallObj = Rhino.NormalObjects
arrEnd(0) = 0
arrEnd(1) = 0
arrEnd(2) = 0
Rhino.MoveObjects iAllObj, arrCenter, arrEnd
```

Because the model will be disappeared from the current view now, the window is zoomed to the extents of the visible objects.

```
strView = Rhino.CurrentView
Rhino.ZoomExtents strView
```

Then the radius of the research area is given to the user. This radius is needed in a later stage to create the computational domain.

```
Rhino.MessageBox ("Radius (m) of the research area for use in the
                  Virtual Wind Tunnel: " & CInt(r))
```

The subroutine is finally finished by using the *End Sub* code. With the *Area* command Rhinoceros is told to run the subroutine.

```
End Sub

Area
```

The result of the script is a 2D model of the research area with polygons that represent the buildings in the research area. The polygons contain several height points that hold information about the elevation level of the buildings. The research area is bounded by a circular polygon.

Appendix D: Script to generate the 3D model

In this appendix the script that generates a 3D model from the 2D research area created by the first tool, is discussed in detail. This script also uses the *Region_Select* plug-in, so it has to be installed first if it is not already done. Before running the script that generates the 3D model, the user has to open the 2D model of the research area in Rhinoceros first.

Again the script starts with the *Option Explicit* command and the definition of the subroutine. In the following step the variable and constant terms are declared that will be used in the subroutine. The local settings are then again set at English – United Kingdom.

```
Option Explicit

Sub Extrude()

Dim iallObjects, iPoints, i
Dim layName, arrLayers, strLayer
Dim strTekst
Dim maxHeight, extHeight, height
Dim Pnt, Crv
Dim selPolygon
Dim arrHeight(), arrSorted, median, m, u
Dim arrOptions(2)

Const circHeight = -2

setLocale(2057)
```

The script now sets the current layer to *Default* and turns off the layer that contains the circle, which forms the ground surface. Else the circle would also be extruded if the script goes through all polygons of the model to extrude them.

```
Rhino.CurrentLayer "Default"
Rhino.LayerMode "Circle" , 1
```

The script will now ask the user over which height the polygons have to be extruded. The polygons can be extruded over the actual height, over a truncated height if a building is higher than 15 m or over no height at all.

```
arrOptions(0) = "1) Full height"
arrOptions(1) = "2) Truncated height"
arrOptions(2) = "3) No height"

strTekst = Rhino.ListBox (arrOptions, "Choose extrusion height of the
                          surroundings:" )
```

If the user chooses the first option the script will go through an extensive procedure for each polygon. First the maximum height is set to zero. Then all polygons of the model are placed in an array. The script will then loop through all these polygons. The first step in the loop is to set the length of the *arrHeight* array to -1. In this array, all heights of the points of a certain polygon will be stored and sorted. Because on forehand it is not known how many values have to be stored in the array, the length of it is first set to -1. Before a height is added to the array, the array is lengthened by 1 and the height is stored at this new location.

```
If strTekst = "1) Truncated height" Then
  maxHeight = 0
  iAllObjects = Rhino.ObjectsByLayer("ExtrPolygons")
  If IsNull(iAllObjects) Then Exit Sub
  For i = 0 to UBound(iAllObjects)
    Rhino.UnselectAllObjects
    ReDim arrHeight(-1)
    median = 0
```

Then all points that lie in a certain polygon are selected with the *RegionSelect* command. The polygon is given a name to be able to call it later.

```
    If Rhino.IsCurve(iAllObjects(i)) Then
      If Rhino.IsCurveClosed(iAllObjects(i)) Then
        Rhino.Command "_RegionSelect '_SelID " & iAllObjects(i)
        Rhino.ObjectName iAllObjects(i), i
```

For each point that is selected the height is derived from the accompanying layer. The height is then stored in the *arrHeight* array.

```
        iPoints = Rhino.SelectedObjects
        If IsArray(iPoints) Then
          For Each Pnt in iPoints
            layName = Rhino.ObjectLayer(Pnt)
            If IsNumeric(layName) Then
              height = CDbl(layName)
              ReDim Preserve arrHeight(UBound(arrHeight) + 1)
              arrHeight(UBound(arrHeight)) = height
            End If
          Next
```

The height of each point is derived from the accompanying layers and added up:

```
        If IsArray(iPoints) Then
          For Each Pnt in iPoints
            layName = Rhino.ObjectLayer(Pnt)
            If IsNumeric(layName) Then
              height = CDbl(layName)
              totHeight = totHeight + height
            End If
          Next
```

After this is done for all points, the *arrHeight* array contains all height information for the polygon. Due to the inaccuracies of the AHN dataset, some points differ considerably from the real height. Taking the mean height of all points will lead to a lower height than reality because of these inaccuracies. The problem can be solved by taking the median height in stead of the mean height. As the median is the middle of a distribution, the array containing the height information has to be sorted first. Sorting the array is easy in Rhinoceros because of the *Rhino.SortNumbers* command.

After the array is sorted it will be checked if the amount of values in the array is even or uneven. When the amount is uneven, the median height is simply the middle value of the array. With an even amount of values, the median height is the mean of the middle two values in the array.

```
          arrSorted = Rhino.SortNumbers(arrHeight, vbTrue)
          u = UBound(arrSorted)
          If u MOD 2 = 0 Then
            median = arrSorted(u/2)
          Else
            m = arrSorted(u/2 - 1/2) + arrSorted(u/2 + 1/2)
            median = m/2
          End If
```

To be able to give the user the maximum height all buildings in the environment, it must be checked for each polygon if the median height is larger than the maximum height that was already defined. If so, the maximum height is set to this new value.

```
          If median > maxHeight Then
            maxHeight = median
          End If
```

Then the concerning polygon is called and extruded over the median height. The polygon itself is deleted then. What remains is a solid.

```
          Rhino.UnselectAllObjects
          Rhino.Command "SelName " & i
          selPolygon = Rhino.SelectedObjects
          Rhino.Command ("_extrudeCrv c=yes " & median)
          For Each Crv in selPolygon
            Rhino.DeleteObject(Crv)
          Next
        End If
      End If
    End If
  Next
End If
```

If the user chooses the second option that extrudes the polygons over a maximum height of 15 m, the script will go through almost the same procedure as for the first option.

```
If strTekst = "2) Truncated height" Then
  maxHeight = 0
  ReDim arrHeight(-1)
  iAllObjects = Rhino.ObjectsByLayer("ExtrPolygons")
  If IsNull(iAllObjects) Then Exit Sub
  For i = 0 to UBound(iAllObjects)
    Rhino.UnselectAllObjects
    median = 0
    If Rhino.IsCurve(iAllObjects(i)) Then
      If Rhino.IsCurveClosed(iAllObjects(i)) Then
        Rhino.Command "_RegionSelect '_SelID " & iAllObjects(i)
        Rhino.ObjectName iAllObjects(i), i
        iPoints = Rhino.SelectedObjects
        If IsArray(iPoints) Then
          For Each Pnt in iPoints
            layName = Rhino.ObjectLayer(Pnt)
            If IsNumeric(layName) Then
              height = CDbl(layName)
              ReDim Preserve arrHeight(UBound(arrHeight) + 1)
              arrHeight(UBound(arrHeight)) = height
            End If
          Next
          arrSorted = Rhino.SortNumbers(arrHeight, vbTrue)
          u = UBound(arrSorted)
          If u MOD 2 = 0 Then
            median = arrSorted(u/2)
          Else
            m = arrSorted(u/2 - 1/2) + arrSorted(u/2 + 1/2)
            median = m/2
          End If
```

The only difference is that the extrusion height is set to 15 m if the median height of all points for a polygon is higher than 15 m.

```
          If median > 15 Then
            extHeight = 15
            If extHeight > maxHeight Then
              maxHeight = extHeight
            End If
          End If
          Rhino.UnselectAllObjects
          Rhino.Command "SelName " & i
          selPolygon = Rhino.SelectedObjects
          Rhino.Command ("_extrudeCrv c=yes " & extHeight)
          For Each Crv in selPolygon
            Rhino.DeleteObject(Crv)
          Next
        End If
      End If
    End If
  Next
End If
```

Also for the third option the script will go through a more or less same procedure.

```
If strTekst = "3) No height" Then
  maxHeight = 0
  iAllObjects = Rhino.ObjectsByLayer("ExtrPolygons")
  If IsNull(iAllObjects) Then Exit Sub
  For i = 0 to UBound(iAllObjects)
    Rhino.UnselectAllObjects
    If Rhino.IsCurve(iAllObjects(i)) Then
      If Rhino.IsCurveClosed(iAllObjects(i)) Then
        Rhino.Command "_RegionSelect '_SelID " & iAllObjects(i)
        Rhino.ObjectName iAllObjects(i), i
```

The extrusion height is only not determined anymore, but it is permanently set to zero. If for some reason one wants to extrude all buildings in the research area over a certain equal height that is not zero, only the value for the extrusion height has to be changed here then.

```
        extHeight = 0
        If extHeight > maxHeight Then
          maxHeight = extHeight
        End If
        Rhino.UnselectAllObjects
        Rhino.Command "SelName " & i
        selPolygon = Rhino.SelectedObjects
        Rhino.Command ("_extrudeCrv c=yes " & extHeight)
        For Each Crv in selPolygon
          Rhino.DeleteObject(Crv)
        Next
      End If
    End If
  Next
End If
```

The following final procedures are run through for all three options. First the layer that contains the circle for the ground plan is turned on again and made current. With the *RegionSelect* command, all points that are enclosed by the circle are selected. Differently said, all height points that lie in the polygons are selected, as the circle encloses all polygons. The selected height points are then deleted as they are not useful for the CFD software.

```
Rhino.LayerMode "Circle" , 0
Rhino.CurrentLayer("Circle")
circ = Rhino.ObjectsByLayer("Circle")
For c = 0 to Ubound(circ)
  Rhino.Command "_RegionSelect '_SelID " & circ(c)
Next
iPoints = Rhino.SelectedObjects
Rhino.DeleteObjects iPoints
```

The circle is then extruded over a constant height to generate the ground surface.

```
Rhino.SelectObjects(circ)
Rhino.Command ("_extrudeCrv " & circHeight)
Crv = Rhino.SelectedObjects
Rhino.DeleteObjects Crv

Rhino.UnselectAllObjects
```

Finally the script returns the maximum extrusion height of all buildings in the research area. The largest value of the height of the building of interest and the maximum extrusion height determines the dimensions of the computational domain. The script ends with finishing the subroutine and the command that tells Rhinoceros to run the subroutine.

```
Rhino.MessageBox("The maximum height of the surrounding buildings is
                 " & CInt(maxHeight) & " m.")

End Sub

Extrude
```

Appendix E: Script Rotating Lines

In this appendix the script of the *Rotating Lines* method to find the outer points of a model and generate the enclosing curves for several height levels is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable and constant terms are declared that will be used in the subroutine.

```
Option Explicit

Sub Rotating_Lines_3D

Dim lijn, length
Dim angle, parts
Dim midpoint, midp, endpoint, point(), maxPoint, maxPnt()
Dim cpnt, arrCpoint, bpnt, arrBpoint
Dim dxc, dyc, dxb, dyb
Dim mx, my, mz, x, y, z
Dim cafst, cmaxAfst, bafst, bmaxAfst
Dim cxmax, cymax, bxmax, bymax
Dim selected
Dim iAllObjects, i
Dim arrCCX, j, arrCBX, b
Dim p, h, height, step, h_end, h_start
Dim surrounded, sur(), surround
Dim delete
```

With the following code the osnap mode is set to zero. This means that it is no longer possible to snap to for example endpoints, center points or midpoints of the model parts in Rhinoceros. Else this could give problems when picking the center location of the model with the mouse.

```
Rhino.OsnapMode 0
```

In the next step some new layers are added to the model. The layers of all objects of the model are changed to a new layer, called *Building*. The layer *Points* will be used to store all outer points of the model. The enclosing curves that will be generated through these points will be stored in the *Surrounded* layer.

```
Rhino.AddLayer "Building"
Rhino.CurrentLayer("Building")
selected = Rhino.Command ("_SelAll")
Rhino.Command ("_ChangeToCurrentLayer " & selected)
Rhino.UnselectAllObjects
Rhino.AddLayer "Points"
Rhino.AddLayer "Surrounded", RGB(190,190,190)
```

The user is then asked to pick the center location of the building in top view, from where the rotating lines can be generated. The coordinates of this location are stored in the *midpoint* variable.

```
Rhino.MessageBox ("Pick center location of the building in Top-view")
midpoint = Rhino.GetPoint
```

Then the user is asked for the length of the rotating lines and the amount of parts for which the outer points have to be determined. From this amount the rotation angle of the lines can be calculated. The amount of parts is 360 by default.

```
length = Rhino.RealBox("Give the length of the rotating lines")
parts = Rhino.IntegerBox("Give amount of parts",360)
angle = 360/parts
```

Finally the user is asked for the height of the building, the starting height from where the script has to start and the step size. The starting height will usually be the bottom of the model. The user always has to specify the full height of the building, even if the starting height lies on a different level. The amount of loops *h_end* that the script has to do to determine the outer points over the full height of the building can be derived from the height of the building and the step size.

```
height = Rhino.RealBox("Give the height of the building")
h_start = Rhino.RealBox("Give starting height")
step = Rhino.RealBox("Give step size")
h_end = (height / step)
```

Then the identifiers of all objects in the layer *Building* are stored in the variable *iAllObjects*. The various objects can now be called on from here. The layer *Building* contains all model parts. If there are no parts at all the script will stop.

```
iAllObjects = Rhino.ObjectsByLayer("Building")
If IsNull(iAllObjects) Then Exit Sub
```

Before the loop process to find all outer points starts, the variable *sur* that was declared already will be re-declared now. In this variable all the curves that will be generated through the outer points are stored. Because on forehand it is not known how many curves will be stored in this array, the length of the array is first set to -1. When the first curve is generated, the length of the array will be redefined and the curve will be stored at that position in the array then.

```
ReDim sur(-1)
```

The process to find all outer points for a certain height will be repeated for the amount of loops *(h_end – h_start)*. In each loop the variable *maxPnt* will also be re-declared first. In this variable all outer points that are found will be stored. Because it is again not known on forehand how many outer points will be found, the length of the array is first set to -1. Then the x, y and z coordinates of the starting point of the rotating lines are defined and stored in an array. These values follow from the user-given center location, where the z-value is raised for each loop with the step size.

```
For h = 0 To (h_end – h_start)
  ReDim maxPnt(-1)
  x = midpoint(0)
  y = midpoint(1)
  z = midpoint(2) + h_start + (h*step)
  midp = Array(x, y, z)
```

Then the endpoint of the lines is defined. The x-coordinate of the endpoint is the same as the x-coordinate of the start point. The y-coordinate is increased with the user-specified length of the line. The z-coordinate of the endpoint is raised for each loop with the step size.

```
  mx = midpoint(0)
  my = midpoint(1) + length
  mz = midpoint(2) + h_start + (h*step)
  endpoint = Array(mx, my, mz)
```

Now the following loop is gone through for all parts that are defined by the user. Some maximum distances are first set to zero. Then a line will be added to the model from the just defined midpoint to the endpoint. The line is then rotated around the midpoint with an angle that is derived from the user-defined parts for which the outer points have to be determined.

```
For p = 0 To parts
  cmaxAfst = 0
  bmaxAfst = 0
  cxmax = 0
  cymax = 0
  bxmax = 0
  bymax = 0
  lijn = Rhino.AddLine(midp, endpoint)
  Rhino.RotateObject lijn, midpoint, angle*p
```

A new loop will be gone through for all objects of the model which are stored in the *iAllObjects* variable. For each object it will be checked if there is an intersection between the object and the rotating line. The intersection between two lines can be found with the Rhino command *Rhino.CurveCurveIntersection*. If there is an intersection, it will be stored in the *arrCCX* array. The procedure for an intersection between the rotating line and a surface or solid is scripted later.

```
  For i = 0 To UBound(iAllObjects)
    arrCCX = Rhino.CurveCurveIntersection(lijn, iAllObjects(i))
```

If *arrCCX* is an array then one or more intersections are found. An intersection can exist of an overlap or one single point. For all elements in the array it will be checked if the intersection is one single point with the *arrCCX(j,0) = 1* command. If so, then the distance in x-direction and y-direction between the intersection point and the midpoint is determined. From these two distances the total distance can be determined.

```
    If IsArray(arrCCX) Then
      For j = 0 To UBound(arrCCX)
        If arrCCX(j,0) = 1 Then
        arrCpoint = arrCCX(j,1)
        If IsArray(arrCPoint) Then
            dxc = arrCpoint(0) - midpoint(0)
            dyc = arrCpoint(1) - midpoint(1)
            cafst = sqr(dxc^2 + dyc^2)
```

Now it will be checked if the calculated distance between the intersection point and the midpoint is larger than the specified maximum distance. Because this distance was first set to zero, this will soon be the case. The maximum distance is then set to this new distance and the x- and y-coordinates of that intersection point are stored. When the loop has gone through all objects of the model for the concerning rotating line, the coordinates of the most far away intersection point between the rotating line and other lines of the model are obtained.

```
            If cafst > cmaxAfst Then
              cmaxAfst = cafst
              cxmax = arrCpoint(0)
              cymax = arrCpoint(1)
            End If
          End If
        End If
      Next
    End If
```

Because a building model usually not only consists of lines, but also of surfaces and solids, the above procedure has to be repeated to determine the intersections between the rotating line and the surfaces and solids. Such an intersection can be found with the Rhino command *Rhino.CurveBrepIntersect*. If there is an intersection, it will be stored in the *arrCBX* array. Unlike the *Rhino.CurveCurveIntersection* command, the *Rhino.CurveBrepIntersect* command adds real points to the model at the intersection location. The *Rhino.CurveCurveIntersection* command only returns the coordinates of the intersection. For each generated point the coordinates have to be returned first.

```
      arrCBX = Rhino.CurveBrepIntersect(lijn, iAllObjects(i))
      If IsArray(arrCBX) Then
        For Each b in arrCBX
          arrBpoint = Rhino.PointCoordinates(b)
```

Then the distances in x- and y-direction between the intersection point and the midpoint can be determined. From these two distances the total distance can be determined. Again it will be checked if this total distance is larger than the specified maximum distance. If so, then the maximum distance is set to this new distance and the x and y-coordinates of that intersection point are stored. Then the generated points are deleted from the model. The coordinates of the most far away intersection point between the concerning rotating line and surfaces or solids of the model are obtained when the loop has gone through all objects of the model.

```
          If IsArray(arrBpoint) Then
            dxb = arrBpoint(0) - midpoint(0)
            dyb = arrBpoint(1) - midpoint(1)
            bafst = sqr(dxb^2 + dyb^2)
            If bafst > bmaxAfst Then
              bmaxAfst = bafst
              bxmax = arrBpoint(0)
              bymax = arrBpoint(1)
            End If
          End If
          Rhino.DeleteObject(b)
        Next
      End If
    Next
```

For the concerning rotating line it will now be checked which distance is larger: the distance between the intersection with a line of the distance between the intersection with a surface or solid. If the distance between the midpoint and the intersection with a line is larger, then a 3D array is generated with the coordinates of this intersection location. A point is added to the model then at that location with the layer *Points*.

```
    If cmaxAfst > bmaxAfst Then
      If cmaxAfst > 0 Then
        Rhino.CurrentLayer "Points"
        maxPoint = Array(cxmax, cymax, z)
        Rhino.AddPoint(maxPoint)
```

The identifier of the point is then added to the *maxPnt* array. In the beginning the length of this array was re-declared at -1. Before the point is added to the array, the upper boundary of the array is raised by 1. The point is then added to this location in the array.

```
        ReDim Preserve maxPnt(UBound(maxPnt) + 1)
        maxPnt(UBound(maxPnt)) = maxPoint
        Rhino.CurrentLayer "Building"
      End If
    End If
```

If the distance between the midpoint and the intersection with a surface or solid is larger, a 3D array is generated with the coordinates of this intersection location. A point is added to the model with the layer *Points* and the identifier of the point is added to the *maxPnt* array. After the most far away intersection location is determined for a certain rotating line, the line is removed from the model.

```
    If bmaxAfst >= cmaxAfst Then
      If bmaxAfst > 0 Then
        Rhino.CurrentLayer "Points"
        maxPoint = Array(bxmax, bymax, z)
        Rhino.AddPoint(maxPoint)
        ReDim Preserve maxPnt(UBound(maxPnt) + 1)
        maxPnt(UBound(maxPnt)) = maxPoint
        Rhino.CurrentLayer "Building"
      End If
    End If
    Rhino.DeleteObject lijn
  Next
```

When for all parts the most far away intersection points are determined, a polyline can be generated through these points. First it will be checked if the intersection distance is larger than 0. If not, there is no intersection found and no polyline can be generated. If there are intersections found, then a polyline with layer *Surrounded* is added through all points of the *maxPnt* array.

```
  If cmaxAfst > 0 OR bmaxAfst > 0 Then
    Rhino.CurrentLayer("Surrounded")
    surround = Rhino.addPolyline(maxPnt)
```

The generated polyline is then added to the *sur* array. The length of this array was also declared to -1 and has to be lengthened each time a polyline is added to the array. This is because it is not known on forehand how many polylines have to be added to the array. The upper boundary of the array is raised by 1 and the polyline is added to the array at this location.

```
    ReDim Preserve sur(Ubound(sur) + 1)
    sur(UBound(sur)) = surround
```

Then all intersection points are selected that were added to the model. They will be deleted.

```
    selected = Rhino.ObjectsByLayer("Points")
    Rhino.DeleteObjects(selected)
  End If
  Rhino.CurrentLayer("Building")
Next
```

If this process is done for all height steps, the original building model can also be removed. In the beginning, the layer of all model parts was changed to the layer *Building*. When all objects in the *Building* layer are removed, the enclosing curves at the various height levels remain.

```
delete = Rhino.ObjectsByLayer("Building")
Rhino.DeleteObjects(delete)

End Sub

Rotating_Lines_3D
```

Appendix F: Script Rectangle

In this appendix the script of the *Rectangle* method to find the outer points of a model and generate the enclosing curves for several height levels is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable and constant terms that will be used in the subroutine are declared. The osnap mode is also set to zero again.

```
Option Explicit

Sub Rectangle_3D

Dim lijn, p, parts
Dim iAllObjects, i
Dim arrCCX, arrCBX, j, k
Dim lo, ro, lb, rb
Dim distx, disty
Dim startpoint, endpoint, startp, endp
Dim selected
Dim arrRect
Dim h, height, step, h_end, h_start, z
Dim cminAfst, cxmin, cymin, cmaxAfst, cxmax, cymax
Dim cpnt, arrCpoint
Dim dxb, dyb, bafst, bminAfst, bxmin, bymin, bmaxAfst, bxmax, bymax
Dim b, arrBpoint
Dim dxc, dyc
Dim cafst
Dim minPoint, Pnt(), maxPoint
Dim strID
Dim arrMin, arrSorted
Dim delete
Dim sur(), boundary()
Dim dist

Rhino.OsnapMode 0
```

In the next step some new layers are added to the model. The layers of all objects of the model are changed to a new layer, called *Building*. The layer *Surrounded* will be used to store the curves that will be generated through the outer points.

```
Rhino.AddLayer "Building"
Rhino.CurrentLayer("Building")
selected = Rhino.Command ("_SelAll")
Rhino.Command ("_ChangeToCurrentLayer " & selected)
Rhino.UnselectAllObjects
Rhino.AddLayer "Surrounded", RGB(190, 190, 190)
```

The user is then asked to draw a rectangle around the building model in top view. The coordinates of the corners of the rectangle are stored in the *arrRect* array.

```
Rhino.MessageBox("Draw a rectangle around the model in Top-view.")
arrRect = Rhino.GetRectangle
If IsArray(arrRect) Then
  lo = arrRect(0)
  ro = arrRect(1)
  rb = arrRect(2)
  lb = arrRect(3)
End If
```

Then the user is asked for the distance between the line elements. For each distance the outer points will be determined as the line elements will be moved over this distance each time.

```
dist = Rhino.RealBox("Give distance between the line elements")
```

Finally the user is asked for the height of the building, the starting height from where the script has to start and the step size. The starting height will usually be the bottom of the model. The user always has to specify the full height of the building, even if the starting height lies on a different level. The amount of loops *h_end* that the script has to do to determine the outer points over the full height of the building can be derived from the height of the building and the step size.

```
height = Rhino.RealBox("Give the height of the building")
h_start = Rhino.RealBox("Give starting height")
step = Rhino.RealBox("Give step size")
h_end = (height / step)
```

Then the identifiers of all objects in the layer *Building* are stored in the variable *iAllObjects*. The various objects can now be called on from here. The layer *Building* contains all model parts. If there are no parts at all, the script will stop.

```
iAllObjects = Rhino.ObjectsByLayer("Building")
If IsNull(iAllObjects) Then Exit Sub
```

Finally the variable *boundary* that was declared already is re-declared. In this variable all curves that will be generated through the outer points are stored. Because on forehand it is not known how many curves will be stored, the length of the array is first set to -1.

```
ReDim boundary(-1)
```

Now the loop process to find all outer points for a certain height starts. The process will be repeated for the amount of loops *(h_end – h_start)*. In each loop the variables *Pnt* and *sur* will be re-declared first. In the variable *Pnt* all outer points that are found will be stored. These points however do not lie in the right sequence to directly draw the enclosing polyline. To order the outer points, the variable *sur* will be used. The ordering process will be explained later in detail. In each loop the level of concern *z* will also be defined first. This level is derived from the level of the drawn rectangle, the starting height and the step size.

```
For h = 0 To (h_end - h_start)
  ReDim Pnt(-1)
  ReDim sur(-1)
  z = lo(2) + h_start + h*step
```

The process to search for the outer points is divided in two parts, as the lines will first move from left to right and then from the bottom to the top. The script starts with lines moving from left to right. First the distances in x- and y-direction between the boundaries of the rectangle are determined. The distance in x-direction is the distance from the lower left corner to the lower right corner. The distance from the lower left corner to the upper left corner is the distance in y-direction. For lines moving from left to right, the amount of lines that have to be drawn is the distance in x-direction divided by the user specified distance between the line elements.

```
  'Left to right
  distx = Rhino.Distance(lo, ro)
  disty = Rhino.Distance(lo, lb)
  parts = distx / dist
```

The following loop is gone through for the amount of parts that were just determined. The loop starts with the definition of the startpoint and the endpoint of the line. The x-coordinate of the start- and endpoint is the coordinate of the lower left corner of the rectangle plus the distance between the line elements. The y-coordinate of the startpoint is the y-coordinate of the lower left corner of the rectangle. The y-coordinate of the lower left rectangle plus the distance in y-direction between the lower and upper boundary of the rectangle is the y-coordinate of the endpoint. The z-coordinate of the start- and endpoint is the z-coordinate of the lower left corner of the drawn rectangle plus the starting height and the step size. Finally a line is added between these points.

```
For p = 0 To parts
  startpoint = Array((lo(0) + p*dist), lo(1), z)
  endpoint = Array((lo(0) + p*dist), (lo(1) + disty), z)
  lijn = Rhino.AddLine(startpoint, endpoint)
```

Because the lines will determine the closest and most far away outer points related to the bottom of the rectangle in one step, some minimum and maximum distances are first defined. The minimum distances are defined as the distance between the endpoint and the startpoint of the line. If an outer point is found, the distance to the bottom of the rectangle is always smaller then. The maximum distances are set to zero. If an outer point is found, the distance to the bottom of the rectangle is always larger.

```
  cminAfst = endpoint(1)- startpoint(1)
  cxmin = endpoint(0) - startpoint(0)
  cymin = endpoint(1) - startpoint(1)
  cmaxAfst = 0
  cxmax = 0
  cymax = 0
  bminAfst = endpoint(1)- startpoint(1)
  bxmin = endpoint(0) - startpoint(0)
  bymin = endpoint(1) - startpoint(1)
  bmaxAfst = 0
  bxmax = 0
  bymax = 0
```

The following loop goes through all objects of the model, which are stored in the *iAllObjects* variable. For each object it will be checked if there is an intersection between the object and the line. The procedure for an intersection between the line and a surface or solid is scripted later. The intersection between the line and other lines of the model is found with the *Rhino.CurveCurveIntersection* command. If there is an intersection, it will be stored in the *arrCCX* array. As in the script for the *Rotating Lines* method, it will be checked for all elements in the array if the intersection is one single point or an overlap. For single point intersections the distances in x- and y-direction between the intersection point and the startpoint of the line are determined. From these two distances the total distance can be derived.

```
  For i = 0 To UBound(iAllObjects)
    arrCCX = Rhino.CurveCurveIntersection(lijn, iAllObjects(i))
    If IsArray(arrCCX) Then
      For j = 0 to UBound(arrCCX)
        If arrCCX(j,0) = 1 Then
          arrCpoint = arrCCX(j,1)
          If IsArray(arrCpoint) Then
            dxc = arrCpoint(0) - startpoint(0)
            dyc = arrCpoint(1) - startpoint(1)
            cafst = sqr(dxc^2 + dyc^2)
```

Now it will be checked if the calculated distance between the intersection point and the startpoint of the line is smaller than the specified minimum distance. It will also be checked if the distance is larger than the specified maximum distance. If one of these situations occurs, the minimum or maximum distance is set to this new distance and the x- and y-coordinates of that intersection point are stored. When the loop has gone through all objects of the model for the concerning line, the coordinates of the closest and most far away intersection point related to the bottom of the rectangle are obtained.

```
              If cafst < cminAfst Then
                cminAfst = cafst
                cxmin = arrCpoint(0)
                cymin = arrCpoint(1)
              End If
              If cafst > cmaxAfst Then
                cmaxAfst = cafst
                cxmax = arrCpoint(0)
                cymax = arrCpoint(1)
              End If
            End If
          End If
        Next
      End If
```

The discussed procedure can now be repeated to determine the intersections between the line and surfaces or solids. Such an intersection can be found with the Rhino command *Rhino.CurveBrepIntersect*. Because this command adds real points to the model in stead of returning only the coordinates as the *Rhino.CurveCurveIntersection* command, the coordinates of each generated point have to be returned first. Then the distances in x- and y-direction between the intersection point and the startpoint of the line can be determined. The rest of the procedure is the same as for the intersection between the line and other lines of the model. In the end the coordinates of the closest and most far away intersection between the line and a surface or solid related to the bottom of the rectangle are obtained.

```
      arrCBX = Rhino.CurveBrepIntersect(lijn, iAllObjects(i))
      If IsArray(arrCBX) Then
        For Each b in arrCBX
          arrBpoint = Rhino.PointCoordinates(b)
          If IsArray(arrBpoint) Then
            dxb = arrBpoint(0) - startpoint(0)
            dyb = arrBpoint(1) - startpoint(1)
            bafst = sqr(dxb^2 + dyb^2)
            If bafst < bminAfst Then
              bminAfst = bafst
              bxmin = arrBpoint(0)
              bymin = arrBpoint(1)
            End If
            If bafst > bmaxAfst Then
              bmaxAfst = bafst
              bxmax = arrBpoint(0)
              bymax = arrBpoint(1)
            End If
          End If
          Rhino.DeleteObject(b)
        Next
      End If
    Next
```

For the concerning line it will now be checked which minimum and maximum distances that are obtained by the two methods are normative. 3D arrays are generated with the coordinates of the most minimum and maximum intersection locations. Points are added to the model at these locations then. The identifiers of these points are added to the *Pnt* array. Before a point is added to the array, the upper boundary of the array is raised by 1. The point is then added to this location in the array. After the minimum and maximum points are added, the line is removed from the model.

```
    If cminAfst < bminAfst Then
      If cminAfst < (endpoint(1) - startpoint(1)) Then
        minPoint = Array(cxmin, cymin, z)
        strID = Rhino.AddPoint(minPoint)
        ReDim Preserve Pnt(UBound(Pnt) + 1)
        Pnt(Ubound(Pnt)) = strID
      End If
    End If
    If bminAfst <= cminAfst Then
      If bminAfst < (endpoint(1) - startpoint(1)) Then
        minPoint = Array(bxmin, bymin, z)
        strID = Rhino.AddPoint(minPoint)
        ReDim Preserve Pnt(UBound(Pnt) + 1)
        Pnt(Ubound(Pnt)) = strID
      End If
    End If
    If cmaxAfst > bmaxAfst Then
      If cmaxAfst > 0 Then
        maxPoint = Array(cxmax, cymax, z)
        strID = Rhino.AddPoint(maxPoint)
        ReDim Preserve Pnt(UBound(Pnt) + 1)
        Pnt(Ubound(Pnt)) = strID
      End If
    End If
    If bmaxAfst >= cmaxAfst Then
      If bmaxAfst > 0 Then
        maxPoint = Array(bxmax, bymax, z)
        strID = Rhino.AddPoint(maxPoint)
        ReDim Preserve Pnt(UBound(Pnt) + 1)
        Pnt(Ubound(Pnt)) = strID
      End If
    End If
    Rhino.DeleteObject lijn
  Next
```

When the line is finally moved from the left boundary of the rectangle to the right boundary, the process can be repeated for lines moving from the bottom to the top of the rectangle. The process is nearly the same, only the definition of the start- and endpoints of the line and the definition of some distances differ. For lines moving from the bottom to the top, the amount of lines that have to be drawn is the distance in y direction divided by the user specified distance between the line elements. The x-coordinate of the startpoint is the coordinate of the lower right corner of the rectangle. The x-coordinate of the lower right corner minus the distance in x direction between the left and right boundary of the rectangle is the x-coordinate of the endpoint. The y-coordinate of the start- and endpoint is the coordinate of the lower right corner of the rectangle plus the distance between the line elements. The z-coordinate of the start- and endpoint was already defined at the beginning of the script. It is the z-coordinate of the lower left corner of the drawn rectangle plus the starting height and the step size. Finally, the minimum distances are defined as the distance between the startpoint and the endpoint of the line. Without further explanation, the script for lines moving from the bottom to the top of the rectangle is given on the following pages.

```
'Bottom to top
distx = Rhino.Distance(lo, ro)
disty = Rhino.Distance(lo, lb)
parts = disty / dist
For p = 0 To parts
  startpoint = Array(ro(0), (ro(1) + p*dist), z)
  endpoint = Array((ro(0) - distx), (ro(1) + p*dist), z)
  lijn = Rhino.AddLine(startpoint, endpoint)
  cminAfst = startpoint(0) - endpoint(0)
  cxmin = startpoint(0) - endpoint(0)
  cymin = startpoint(1) - endpoint(1)
  cmaxAfst = 0
  cxmax = 0
  cymax = 0
  bminAfst = startpoint(0) - endpoint(0)
  bxmin = startpoint(0) - endpoint(0)
  bymin = startpoint(1) - endpoint(1)
  bmaxAfst = 0
  bxmax = 0
  bymax = 0
  For i = 0 To UBound(iAllObjects)
    arrCCX = Rhino.CurveCurveIntersection(lijn, iAllObjects(i))
    If IsArray(arrCCX) Then
      For j = 0 to UBound(arrCCX)
        If arrCCX(j,0) = 1 Then
          arrCpoint = arrCCX(j,1)
          If IsArray(arrCpoint) Then
            dxc = startpoint(0) - arrCpoint(0)
            dyc = startpoint(1) - arrCpoint(1)
            cafst = sqr(dxc^2 + dyc^2)
            If cafst < cminAfst Then
              cminAfst = cafst
              cxmin = arrCpoint(0)
              cymin = arrCpoint(1)
            End If
            If cafst > cmaxAfst Then
              cmaxAfst = cafst
              cxmax = arrCpoint(0)
              cymax = arrCpoint(1)
            End If
          End If
        End If
      Next
    End If
    arrCBX = Rhino.CurveBrepIntersect(lijn, iAllObjects(i))
    If IsArray(arrCBX) Then
      For Each b in arrCBX
        arrBpoint = Rhino.PointCoordinates(b)
        If IsArray(arrBpoint) Then
          dxb = startpoint(0) - arrBpoint(0)
          dyb = startpoint(1) - arrBpoint(1)
          bafst = sqr(dxb^2 + dyb^2)
          If bafst < bminAfst Then
            bminAfst = bafst
            bxmin = arrBpoint(0)
            bymin = arrBpoint(1)
          End If
          If bafst > bmaxAfst Then
            bmaxAfst = bafst
            bxmax = arrBpoint(0)
            bymax = arrBpoint(1)
          End If
        End If
        Rhino.DeleteObject(b)
      Next
    End If
  Next
  If cminAfst < bminAfst Then
    If cminAfst < (startpoint(0) - endpoint(0)) Then
      minPoint = Array(cxmin, cymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(Ubound(Pnt)) = strID
    End If
  End If
  If bminAfst <= cminAfst Then
    If bminAfst < (startpoint(0) - endpoint(0)) Then
      minPoint = Array(bxmin, bymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(Ubound(Pnt)) = strID
    End If
  End If
  If cmaxAfst > bmaxAfst Then
    If cmaxAfst > 0 Then
      maxPoint = Array(cxmax, cymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(Ubound(Pnt)) = strID
    End If
  End If
  If bmaxAfst >= cmaxAfst Then
    If bmaxAfst > 0 Then
      maxPoint = Array(bxmax, bymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(Ubound(Pnt)) = strID
    End If
  End If
  Rhino.DeleteObject lijn
Next
```

For all parts the outer points are determined now. The next step is to generate a polyline through these points. As mentioned before, the outer points do not lie in the right sequence in the *Pnt* array. Ordering of the array is required before the enclosing polyline can be generated. The procedure of ordering the array of outer points is based on finding the closest point for a certain outer point of the *Pnt* array. Starting from the first point of the array, the distances from all other points in the array to this first point are calculated. When the closest point is found, it is placed in the *sur* array. From this point the distances to all other points of the *Pnt* array are determined again to find the closest point for this one. That point is again placed in the *sur* array. When this procedure is done for all points of the *Pnt* array, the *sur* array will contain all outer points in the right sequence. Neighboring points lie next to each other in the array. Now a polyline can be generated through these points. On the next page the script continues with the ordering of the points.

First some new variables are declared that will be used for the ordering process. There are also three new layers added to the model. In the layer *Surround* the ordered points in the *sur* array will be stored. The original points in the *Pnt* array that are copied to the *sur* array will be stored in the layer *Done*. The script will not use them anymore for the determination of the closest point. In the *Boundary* layer the enclosing polyline through the outer points will be stored.

```
'Curve through closest point
Dim s, sx, sy, e, ex, ey, t, tx, ty
Dim min
Dim q, r, n
Dim dx, dy, minx, miny
Dim maxAfst, afst
Dim layer
Dim surrounding

Rhino.AddLayer "Surround"
Rhino.AddLayer "Done"
Rhino.AddLayer "Boundary", RGB(190,190,190)
```

If the *Pnt* array is not empty, the x- and y-coordinates of the first point in the array are stored in the new variables *sx*, *sy*, *ex* and *ey*. *Sx* and *sy*, together with the earlier defined value *z*, will form the x-, y- and z-coordinates of the first point in the *sur* array. The point is added to the model in the *Surround* layer. *Ex* and *ey* are the same as *sx* and *sy* and will form the last point in the *sur* array. In the end this point will be added to the array and the first and the last point will have the same coordinates then. This will give a closed curve when a polyline is generated through these points. Before any point is added to the *sur* array and the model, the layer *Surround* is turned on and set to the current layer. After adding the point, the layer *Surround* is turned off, removing all objects in that layer temporarily from the model. In this way the newly added points are not taken into account by the determination of the closest points. This procedure is now first done for the first point of the *Pnt* array.

```
If UBound(Pnt) > 0 Then
  s = Rhino.PointCoordinates(Pnt(0))
  sx = s(0)
  sy = s(1)
  ex = s(0)
  ey = s(1)
  Rhino.LayerMode "Surround",0
  Rhino.CurrentLayer "Surround"
  min = Array(sx, sy, z)
  Rhino.AddPoint(min)
  ReDim Preserve sur(UBound(sur) + 1)
  sur(UBound(sur)) = min
  Rhino.CurrentLayer "Building"
  Rhino.LayerMode "Surround",1
```

As the first point is now added to the *sur* array, the layer of the original point is changed to the layer *Done*. When searching for the closest points, the points in the layer *Done* can be ignored.

```
  Rhino.CurrentLayer "Done"
  Rhino.SelectObject(Pnt(0))
  Rhino.Command ("_ChangeToCurrentLayer")
  Rhino.CurrentLayer "Building"
```

Now the first point of the *Pnt* array is copied to the *sur* array and marked as done, the script continues with the search for the closest point. Two loops have to be going through: for all points in the *Pnt* array the closest point has to be found, and for finding the closest point, the distances of all other points to the concerning point have to be calculated. For each point where a closest point has to be searched for, a maximum distance is set.

```
For q = 0 To UBound(Pnt)
  maxAfst = 10 * (Rhino.Distance(lo,ro))
```

For all other points of the *Pnt* array will then be checked if the layer of the point is *Done*.

```
  For r = 0 To UBound(Pnt)
    layer = Rhino.ObjectLayer(Pnt(r))
    If layer = "Done" Then
    Else
```

If this is the case, nothing is done with the point. Else the coordinates of the point are returned and the distance in x- and y-direction to the concerning point are determined. A total distance of zero means that there is another point with the same coordinates. This point will be marked as *Done* then.

```
      t = Rhino.PointCoordinates(Pnt(r))
      tx = t(0)
      ty = t(1)
      dx = tx - sx
      dy = ty - sy
      afst = sqr(dx^2 + dy^2)
      If afst = 0 Then
        Rhino.CurrentLayer"Done"
        Rhino.SelectObject(Pnt(r))
        Rhino.Command("_ChangeToCurrentLayer")
        Rhino.UnselectAllObjects
        Rhino.CurrentLayer"Building"
      End If
```

If the total distance between the concerning point and another point in the *Pnt* array is smaller than the defined maximum distance, the maximum distance is set to this new distance and the x- and y-coordinates are stored. When the loop has gone through all points of the *Pnt* array, the coordinates of the closest point are obtained.

```
      If afst < maxAfst Then
        maxAfst = afst
        minx = tx
        miny = ty
        n = r
      End If
    End If
  Next
```

A new point is now added to the model with the coordinates of this closest point. The point is added in the *Surround* layer and also placed in the *sur* array.

```
  min = Array(minx, miny, z)
  Rhino.LayerMode "Surround",0
  Rhino.CurrentLayer "Surround"
  Rhino.AddPoint(min)
  ReDim Preserve sur(UBound(sur) + 1)
  sur(UBound(sur)) = min
```

The layer *Surround* is then turned off again and the layer of the concerning point is changed to *Done*. The variables *Sx* and *Sy* are the coordinates of the concerning point and they are set to the x- and y- coordinates of the just found closest point. This point now becomes the new concerning point for which a closest point has to be found.

```
      Rhino.CurrentLayer "Done"
      Rhino.LayerMode "Surround",1
      Rhino.SelectObject(Pnt(n))
      Rhino.Command("_ChangeToCurrentLayer")
      Rhino.CurrentLayer "Building"
      sx = minx
      sy = miny
    Next
```

When all closest points are found and placed in the *sur* array, the array contains only points that lie in sequence. Neighboring points in the model also lie next to each other in the array. The last point that is added to the array is the point with the same coordinates as the first point of the array. This will give a closed curve when creating a polyline through the points.

```
    Rhino.LayerMode "Surround",0
    Rhino.CurrentLayer "Surround"
    e = Array(ex, ey, z)
    Rhino.AddPoint e
    ReDim Preserve sur(UBound(sur) + 1)
    sur(UBound(sur)) = e
```

All points that are marked as *Done* can be deleted from the model now.

```
    delete = Rhino.ObjectsByLayer("Done")
    Rhino.DeleteObjects delete
```

Then a polyline can be added through the points of the *sur* array. The polyline is added to the *Boundary* array.

```
    Rhino.CurrentLayer "Boundary"
    surrounding = Rhino.AddPolyline(sur)
    ReDim Preserve boundary(UBound(boundary) + 1)
    boundary(UBound(boundary)) = surrounding
    Rhino.CurrentLayer"Building"
```

Also the points of the *sur* array, which are placed in the *Surround* layer, can be deleted from the model.

```
    delete = Rhino.ObjectsByLayer("Surround")
    Rhino.DeleteObjects delete
    Rhino.UnselectAllObjects
  End If
Next
```

If this process is done for all height steps and the various enclosing curves are created, the original building model, which is placed in the *Building* layer, can be removed. What remain are the enclosing curves at the various height levels.

```
delete = Rhino.ObjectsByLayer("Building")
Rhino.DeleteObjects(delete)

End Sub
Rectangle_3D
```

Appendix G: Script Rotated Square

In this appendix the script of the *Rotated Square* method to find the outer points of a model and generate the enclosing curves for several height levels is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable and constant terms that will be used in the subroutine are declared. The osnap mode is also set to zero.

```
Option Explicit

Sub Rotated_Square_3D

Dim lijn, p, parts
Dim arrRect, lo, ro, rb, lb
Dim distx, disty
Dim startPx, startPy, startPz, startP, start, eindPx, eindPy, eindPz,
    eindP, eind, endPx, endPy, endPz
Dim rot(), rotated
Dim SPx, SPy, SPz, SP, EPx, EPy, EPz, EP
Dim dist, selected
Dim iAllObjects, i
Dim arrCCX, arrCBX, j
Dim cminAfst, cxmin, cymin, cmaxAfst, cxmax, cymax, cafst
Dim bminAfst, bxmin, bymin, bmaxAfst, bxmax, bymax, bafst
Dim cpnt, arrCpoint, b, arrBpoint
Dim dxc, dyc, dxb, dyb
Dim minPoint, Pnt(), maxPoint
Dim arrMin, arrSorted, strID
Dim h, height, step, h_end, h_start, z
Dim sur(), boundary()
Dim s, sx, sy, e, ex, ey, t, tx, ty
Dim min
Dim q, r, n
Dim dx, dy, minx, miny
Dim maxAfst, afst
Dim layer
Dim delete
Dim surrounding
Dim loft, crv

Rhino.OsnapMode 0
```

In the next step some new layers are added to the model. The layers of all objects of the model are changed to a new layer, called *Building*. The layer *Surround* will be used to store the ordered points. The layer *Done* will be used to mark the points that are ordered.

```
Rhino.AddLayer "Building"
Rhino.CurrentLayer "Building"
selected = Rhino.Command ("_SelAll")
Rhino.Command ("_ChangeToCurrentLayer " & selected)
Rhino.UnselectAllObjects
Rhino.AddLayer "Surround"
Rhino.AddLayer "Done"
Rhino.AddLayer "Boundary", RGB(190,190,190)
```

The user is then asked to draw a rectangle around the model in top view. The coordinates of the corners of the rectangle are stored in the *arrRect* array.

```
Rhino.MessageBox("Draw a rectangle around the model in Top-view.")
arrRect = Rhino.GetRectangle
If IsArray(arrRect) Then
  lo = arrRect(0)
  ro = arrRect(1)
  rb = arrRect(2)
  lb = arrRect(3)
End If
```

From the corner coordinates the distance in x- and y-direction between the boundaries of the rectangle are determined.

```
distx = Rhino.Distance(lo, ro)
disty = Rhino.Distance(lo, lb)
```

Now the user is asked for the distance between the line elements. From that distance the amount of lines that have to be drawn can be calculated. Because the lines will move between the boundaries of a rotated square, the amount of parts is the length of an edge of the square divided by the user specified distance between the line elements.

```
dist = Rhino.RealBox("Give distance between the line elements")
parts = (sqr(2*(1/2*distx + 1/2*disty)^2))/dist
```

Finally the user is asked for the height of the building, the starting height from where the script has to start and the step size. The amount of loops *h_end* that the script has to do to determine the outer points over the full height of the building can be derived from the height of the building and the step size.

```
height = Rhino.RealBox("Give the height of the building")
h_start = Rhino.RealBox("Give starting height")
step = Rhino.RealBox("Give step size")
h_end = (height / step)
```

Then the identifiers of all objects in the layer *Building* are stored in the variable *iAllObjects*. The various objects can now be called on from here. The layer *Building* contains all model parts. If there are no parts at all, the script will stop.

```
iAllObjects = Rhino.ObjectsByLayer("Building")
If IsNull(iAllObjects) Then Exit Sub
```

Before the loop process to find the outer points starts, the variable *boundary* that was declared already is re-declared. In this variable all curves that will be generated through the outer points are stored. Because on forehand it is not known how many curves have to be stored, the length of the array is first set to -1.

```
ReDim boundary(-1)
```

The loop process to find all outer points for a certain height will be repeated for the amount of loops *(h_end – h_start)*. In each loop the variables *Pnt*, *sur* and *rot* will be re-declared first. In the variable *Pnt* all outer points that are found will be stored. These points however do not lie in the right sequence to directly draw the enclosing polyline. To order the outer points, the variable *sur* will be used. The variable *rot* will be used to store the corner points of the rotated square. In each loop the level of concern *z* will also be defined first. This level is derived from the level of the drawn rectangle, the starting height and the step size.

```
For h = 0 To (h_end - h_start)
  ReDim Pnt(-1)
  ReDim sur(-1)
  ReDim rot(-1)
  z = lo(2) + h_start + h*step
```

The loop process continues with the generation of the rotated square. For this the dimensions of the user-specified rectangle are important. The corner points determine the final shape of the rotated square. For the definition of the corner points of the square it must first be determined if the width of the rectangle is larger than the length of the rectangle. If so, then the x- and y-coordinates of the left corner of the square follow from the coordinates of the lower left corner of the rectangle. The x-coordinate of the left square corner point is the x-coordinate of the lower left rectangle corner point minus half the distance between the top and bottom of the original rectangle. The y-coordinate of this point is the y-coordinate of the lower left rectangle corner point plus half the distance between the top and bottom of the rectangle. The z-coordinate follows from the z-coordinate of the rectangle, the starting height and the step size. These three coordinates form the location of the left corner of the square.

```
  If distx > disty Then
    startPx = lo(0) - 1/2*disty
    startPy = lo(1) + 1/2*disty
    startPz = z
    startP = Array(startPx, startPy, startPz)
```

To form a closed square when a polyline is added through the corner points, the coordinates of the end point of the square are taken from the startpoint. This point will be added later to the array of corner points *rot*. The startpoint is the first point that is added to the array.

```
    endPx = startP(0)
    endPy = startP(1)
    endPz = z
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = startP
```

Then the next corner point of the rotated square is determined. The top corner point follows from an addition of some distances in x- and y-direction to the coordinates of the starting point. This second point is also added to the array of corner points then.

```
    eindPx = startP(0) + (1/2*distx + 1/2*disty)
    eindPy = startP(1) + (1/2*distx + 1/2*disty)
    eindPz = z
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
```

The other two corner points can also be determined now. The coordinates for each of these points follow from the previous determined point with some distances in x- and y-direction added to or subtracted from these coordinates. The points are then added to the *rot* array.

```
    startP = eindP
    eindPx = startP(0) + (1/2*distx + 1/2*disty)
    eindPy = startP(1) - (1/2*distx + 1/2*disty)
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
    startP = eindP
    eindPx = startP(0) - (1/2*distx + 1/2*disty)
    eindPy = startP(1) - (1/2*distx + 1/2*disty)
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
```

Finally the endpoint, which has the same coordinates as the startpoint, is added to the array. A polyline can be added through these points then, creating a rotated, closed square.

```
    eindP = Array(endPx, endPy, endPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
    rotated = Rhino.AddPolyline(rot)
  End If
```

The same procedure works if the width of the user-specified rectangle is equal to or smaller than the length of the rectangle. The first corner point that is determined now only is the top corner of the square. From that point on all other corner points can be defined. Without further explanation the script to create the rotated square in case of these conditions is given below.

```
  If disty >= distx Then
    startPx = lb(0) + 1/2*distx
    startPy = lb(1) + 1/2*distx
    startPz = z
    startP = Array(startPx, startPy, startPz)
    endPx = startP(0)
    endPy = startP(1)
    endPz = z
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = startP
    eindPx = startP(0) + (1/2*distx + 1/2*disty)
    eindPy = startP(1) - (1/2*distx + 1/2*disty)
    eindPz = z
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP

    startP = eindP
    eindPx = startP(0) - (1/2*distx + 1/2*disty)
    eindPy = startP(1) - (1/2*distx + 1/2*disty)
    eindPz = z
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
    startP = eindP
    eindPx = startP(0) - (1/2*distx + 1/2*disty)
    eindPy = startP(1) + (1/2*distx + 1/2*disty)
    eindPz = z
    eindP = Array(eindPx, eindPy, eindPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
    eindP = Array(endPx, endPy, endPz)
    ReDim Preserve rot(UBound(rot) + 1)
    rot(UBound(rot)) = eindP
    rotated = Rhino.AddPolyline(rot)
  End If
```

Now the rotated square is generated, the process to search for the outer points can start. The process is divided in two parts, as the lines will first move from the lower left edge of the square to the upper right edge of the square. After that the lines will move from the upper left edge of the square to the lower right edge of the square. The following loop is gone through for the amount of parts that were already determined. The loop starts with the definition of the startpoint and the endpoint of the line. Then a line is added between these points.

```
    'Lower left to upper right edge
    For p = 0 To parts
      SPx = (lo(0) - 1/2*disty) + p*dist*0.707107
      SPy = (lo(1) + 1/2*disty) + p*dist*0.707107
      SPz = z
      SP = Array(SPx, SPy, SPz)
      EPx = SPx + (1/2*distx + 1/2*disty)
      EPy = SPy - (1/2*distx + 1/2*disty)
      EPz = z
      EP = Array(EPx, EPy, EPz)
      lijn = Rhino.AddLine(SP, EP)
```

Because the lines will determine the closest and most far away outer points related to the startpoint of the line in one step, some minimum and maximum distances are first defined.

```
      cminAfst = sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2)
      cxmin = EP(0) - SP(0)
      cymin = EP(1) - SP(1)
      cmaxAfst = 0
      cxmax = 0
      cymax = 0
      bminAfst = sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2)
      bxmin = EP(0) - SP(0)
      bymin = EP(1) - SP(1)
      bmaxAfst = 0
      bxmax = 0
      bymax = 0
```

The following loop goes through all objects of the model, which are stored in the *iAllObjects* variable. For each object it will be checked if there is an intersection between the object and the line. The procedure for the intersection between the line and a surface or solid is scripted later. If an intersection is found, the distances in x- and y-direction between the intersection point and the startpoint of the line are determined. From these two distances the total distance can be derived.

```
For i = 0 To UBound(iAllObjects)
  arrCCX = Rhino.CurveCurveIntersection(lijn, iAllObjects(i))
  If IsArray(arrCCX) Then
    For j = 0 to UBound(arrCCX)
      If arrCCX(j,0) = 1 Then
        arrCpoint = arrCCX(j,1)
        If IsArray(arrCpoint) Then
          dxc = arrCpoint(0) - SP(0)
          dyc = arrCpoint(1) - SP(1)
          cafst = sqr(dxc^2 + dyc^2)
```

Then it will be checked if the total distance between the intersection point and the startpoint of the line is smaller than the specified minimum distance. It will also be checked if the distance is lager than the specified maximum distance. If one of these situations occurs, the minimum or maximum distance is set to this new distance and the x- and y-coordinates of that intersection point are stored. When the loop has gone through all objects of the model for the concerning line, the coordinates of the closest and most far away intersection point related to the startpoint of the line are obtained.

```
          If cafst < cminAfst Then
            cminAfst = cafst
            cxmin = arrCpoint(0)
            cymin = arrCpoint(1)
          End If
          If cafst > cmaxAfst Then
            cmaxAfst = cafst
            cxmax = arrCpoint(0)
            cymax = arrCpoint(1)
          End If
        End If
      End If
    Next
  End If
```

This procedure can now be repeated to determine the intersections between the line and a surface or solid. The procedure is already discussed for the script of the *Rectangle* method. In the end the coordinates of the closest and most far away intersections between the line and a surface or solid related to the startpoint of the line are obtained.

```
  arrCBX = Rhino.CurveBrepIntersect(lijn, iAllObjects(i))
  If IsArray(arrCBX) Then
    For Each b in arrCBX
      arrBpoint = Rhino.PointCoordinates(b)
      If IsArray(arrBpoint) Then
        dxb = arrBpoint(0) - SP(0)
        dyb = arrBpoint(1) - SP(1)
        bafst = sqr(dxb^2 + dyb^2)
        If bafst < bminAfst Then
          bminAfst = bafst
          bxmin = arrBpoint(0)
          bymin = arrBpoint(1)
        End If
```

```
          If bafst > bmaxAfst Then
            bmaxAfst = bafst
            bxmax = arrBpoint(0)
            bymax = arrBpoint(1)
          End If
        End If
        Rhino.DeleteObject(b)
      Next
    End If
  Next
```

For the concerning line it will now be checked which minimum and maximum distances that are obtained by the two methods are normative. Points are added at these locations then. The identifiers of these points are added to the *Pnt* array. After the minimum and maximum points are added to the model, the line is removed.

```
  If cminAfst < bminAfst Then
    If cminAfst < sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2) Then
      minPoint = Array(cxmin, cymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If bminAfst <= cminAfst Then
    If bminAfst < sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2) Then
      minPoint = Array(bxmin, bymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If cmaxAfst > bmaxAfst Then
    If cmaxAfst > 0 Then
      maxPoint = Array(cxmax, cymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If bmaxAfst >= cmaxAfst Then
    If bmaxAfst > 0 Then
      maxPoint = Array(bxmax, bymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  Rhino.DeleteObject lijn
Next
```

When the line is finally moved from the lower left edge to the upper right edge of the square, the process is repeated for lines moving from the upper left edge to the lower right edge of the square. The process is nearly the same, only the definition of the start- and endpoints of the line and the definition of some distances differ. Without further explanation, the script for these lines is given on the following pages.

```
'Upper left to lower right edge
For p = 0 To parts
  SPx = (lb(0) + 1/2*distx) + p*dist*0.707107
  SPy = (lb(1) + 1/2*distx) - p*dist*0.707107
  SPz = z
  SP = Array(SPx, SPy, SPz)
  EPx = SPx - (1/2*distx + 1/2*disty)
  EPy = SPy - (1/2*distx + 1/2*disty)
  EPz = z
  EP = Array(EPx, EPy, EPz)
  lijn = Rhino.AddLine(SP, EP)
  cminAfst = sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2)
  cxmin = EP(0) - SP(0)
  cymin = EP(1) - SP(1)
  cmaxAfst = 0
  cxmax = 0
  cymax = 0
  bminAfst = sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2)
  bxmin = EP(0) - SP(0)
  bymin = EP(1) - SP(1)
  bmaxAfst = 0
  bxmax = 0
  bymax = 0
  For i = 0 To UBound(iAllObjects)
    arrCCX = Rhino.CurveCurveIntersection(lijn, iAllObjects(i))
    If IsArray(arrCCX) Then
      For j = 0 To UBound(arrCCX)
        If arrCCX(j,0) = 1 Then
          arrCpoint = arrCCX(j,1)
          If IsArray(arrCpoint) Then
            dxc = arrCpoint(0) - SP(0)
            dyc = arrCpoint(1) - SP(1)
            cafst = sqr(dxc^2 + dyc^2)
            If cafst < cminAfst Then
              cminAfst = cafst
              cxmin = arrCpoint(0)
              cymin = arrCpoint(1)
            End If
            If cafst > cmaxAfst Then
              cmaxAfst = cafst
              cxmax = arrCpoint(0)
              cymax = arrCpoint(1)
            End If
          End If
        End If
      Next
    End If
    arrCBX = Rhino.CurveBrepIntersect(lijn, iAllObjects(i))
    If IsArray(arrCBX) Then
      For Each b in arrCBX
        arrBpoint = Rhino.PointCoordinates(b)
        If IsArray(arrBpoint) Then
          dxb = arrBpoint(0) - SP(0)
          dyb = arrBpoint(1) - SP(1)
          bafst = sqr(dxb^2 + dyb^2)
          If bafst < bminAfst Then
            bminAfst = bafst
            bxmin = arrBpoint(0)
            bymin = arrBpoint(1)
          End If
          If bafst > bmaxAfst Then
            bmaxAfst = bafst
            bxmax = arrBpoint(0)
            bymax = arrBpoint(1)
          End If
        End If
        Rhino.DeleteObject(b)
      Next
    End If
  Next
  If cminAfst < bminAfst Then
    If cminAfst < sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2) Then
      minPoint = Array(cxmin, cymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If bminAfst <= cminAfst Then
    If bminAfst < sqr((EP(0) - SP(0))^2 + (EP(1) - SP(1))^2) Then
      minPoint = Array(bxmin, bymin, z)
      strID = Rhino.AddPoint(minPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If cmaxAfst > bmaxAfst Then
    If cmaxAfst > 0 Then
      maxPoint = Array(cxmax, cymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  If bmaxAfst >= cmaxAfst Then
    If bmaxAfst > 0 Then
      maxPoint = Array(bxmax, bymax, z)
      strID = Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = strID
    End If
  End If
  Rhino.DeleteObject lijn
Next
```

For all parts the outer points are determined now. The next step is to generate a polyline through these points. Just as the *Rectangle* method, the outer points do not lie in the right sequence in the *Pnt* array. Ordering of the array is required before the enclosing polyline can be generated. The procedure of ordering the array is already given for the script of the *Rectangle* method. On the next pages the script for the ordering of the array is given without further explanation.

```
'Curve through closest point
Rhino.DeleteObject rotated
If UBound(Pnt) > 0 Then
  s = Rhino.PointCoordinates(Pnt(0))
  sx = s(0)
  sy = s(1)
  ex = s(0)
  ey = s(1)
  Rhino.LayerMode "Surround",0
  Rhino.CurrentLayer "Surround"
  min = Array(sx, sy, z)
  Rhino.AddPoint(min)
  ReDim Preserve sur(UBound(sur) + 1)
  sur(UBound(sur)) = min
  Rhino.CurrentLayer "Building"
  Rhino.LayerMode "Surround",1

  Rhino.CurrentLayer "Done"
  Rhino.SelectObject(Pnt(0))
  Rhino.Command ("_ChangeToCurrentLayer")
  Rhino.CurrentLayer "Building"

  For q = 0 To UBound(Pnt)
    maxAfst = 10 * (Rhino.Distance(lo,ro))
    For r = 0 To UBound(Pnt)
      layer = Rhino.ObjectLayer(Pnt(r))
      If layer = "Done" Then
      Else
        t = Rhino.PointCoordinates(Pnt(r))
        tx = t(0)
        ty = t(1)
        dx = tx - sx
        dy = ty - sy
        afst = sqr(dx^2 + dy^2)
        If afst = 0 Then
          Rhino.CurrentLayer"Done"
          Rhino.SelectObject(Pnt(r))
          Rhino.Command("_ChangeToCurrentLayer")
          Rhino.UnselectAllObjects
          Rhino.CurrentLayer"Building"
        End If
        If afst < maxAfst Then
          maxAfst = afst
          minx = tx
          miny = ty
          n = r
        End If
      End If
    Next
    min = Array(minx, miny, z)
    Rhino.LayerMode "Surround",0
    Rhino.CurrentLayer "Surround"
    Rhino.AddPoint(min)
    ReDim Preserve sur(UBound(sur) + 1)
    sur(UBound(sur)) = min
    Rhino.CurrentLayer "Done"
    Rhino.LayerMode "Surround",1
    Rhino.SelectObject(Pnt(n))
    Rhino.Command("_ChangeToCurrentLayer")
    Rhino.CurrentLayer "Building"
```

```
      sx = minx
      sy = miny
    Next
    Rhino.LayerMode "Surround",0
    Rhino.CurrentLayer "Surround"
    e = Array(ex, ey, z)
    Rhino.AddPoint e
    ReDim Preserve sur(UBound(sur) + 1)
    sur(UBound(sur)) = e
    delete = Rhino.ObjectsByLayer("Done")
    Rhino.DeleteObjects delete
```

After a new array *sur* is obtained with the outer points lying in the right sequence, a polyline can be drawn through the points of the *sur* array. The polyline is then added to the *Boundary* array.

```
    Rhino.CurrentLayer "Boundary"
    surrounding = Rhino.AddPolyline(sur)
    ReDim Preserve boundary(UBound(boundary) + 1)
    boundary(UBound(boundary)) = surrounding
    Rhino.CurrentLayer"Building"
```

Now the points of the *sur* array, which are placed in the *Surround* layer, can be deleted from the model.

```
    delete = Rhino.ObjectsByLayer("Surround")
    Rhino.DeleteObjects delete
    Rhino.UnselectAllObjects
  End If
Next
```

If this process is done for all height steps and the various enclosing curves are created, the original building model, which is placed in the *Building* layer, can be removed. The enclosing curves at the various height levels remain.

```
delete = Rhino.ObjectsByLayer("Building")
Rhino.DeleteObjects delete

End Sub

Rotated_Square_3D
```

Appendix H: Script Preprocessing

In this appendix the script of the *Preprocessing* method to convert a mesh object into surfaces is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable terms that will be used in the subroutine are declared.

```
Option Explicit

Sub Preprocessing

Dim strTekst, arrOptions(3)
Dim arrMesh, arrCurve, arrBlock, arrSurface
Dim m, c, b, s, i, j
Dim blocks, meshes, curves, obj
Dim arrFaces, arrFace(3)
```

In the next step some constant terms are declared. In Rhinoceros, the several types of geometry objects are numbered. Mesh objects have number 32, curve objects number 4, block objects number 4096 and surface objects have number 8.

```
Const rhMesh = 32
Const rhCurve = 4
Const rhBlock = 4096
Const rhSurface = 8
```

The several options where the user can choose from will now be declared. With the first option, the user can count the number of blocks, meshes, face curves and surfaces in the model. With that information the user can check if the model contains elements which have to be converted to surfaces. With the second option, all block instances of a model are converted to mesh objects. The face curves of these mesh objects can be returned with the third option. From these face curves the surfaces can be created with the fourth option.

```
arrOptions(0) = "1) Count number of Blocks, Meshes, Face Curves and
                    Surfaces"
arrOptions(1) = "2) Split Block Instances into Meshes"
arrOptions(2) = "3) Split Meshes into Face Curves"
arrOptions(3) = "4) Create Surfaces from Face Curves"
```

Then a popup box with the list of options is placed on the screen.

```
strTekst = Rhino.ListBox(arrOptions, "Choose an option:")
```

If the user chooses for option 1, four arrays are created and filled with the various geometry object types. Some variables that are used for the counting process are set to zero.

```
If strTekst = arrOptions(0) Then
  arrMesh = Rhino.ObjectsByType(rhMesh)
  arrCurve = Rhino.ObjectsByType(rhCurve)
  arrBlock = Rhino.ObjectsByType(rhBlock)
  arrSurface = Rhino.ObjectsByType(rhSurface)
  m = 0
  c = 0
  b = 0
  s = 0
```

Four loops will be gone through then for the various object types. In each loop the amount of objects are counted and stored in a variable.

```
  If Not IsNull(arrMesh) Then
    For i = 0 To UBound(arrMesh)
      m = m + 1
    Next
  End If
  If Not IsNull(arrCurve) Then
    For i = 0 To UBound(arrCurve)
      c = c + 1
    Next
  End If
  If Not IsNull(arrBlock) Then
    For i = 0 To UBound(arrBlock)
      b = b + 1
    Next
  End If
  If Not IsNull(arrSurface) Then
    For i = 0 To UBound(arrSurface)
      s = s + 1
    Next
  End If
```

After all objects of the various geometry types are counted, a message box finally gives the amount of the various objects to the user.

```
  Rhino.MessageBox("Number of Blocks: " & b & vbCrLf & "Number of
                    Meshes: " & m & vbCrLf & "Number of Face Curves: "
                    & c & vbCrLf & "Number of Surfaces: " & s)
End If
```

If the user chooses for option 2, all block instances in the model will be split in its underlying meshes. Each block has to be exploded three times to return all its meshes. The explosion process is looped three times for each block instance. In each loop all block instances of a model are first placed in an array.

```
If StrTekst = arrOptions(1) Then
  b = 0
  For i = 0 To 2
    blocks = Rhino.ObjectsByType(rhBlock)
```

Each object in the array will then be exploded and the amount of explosions is counted. After a block instance is exploded, it is directly deleted from the model.

```
    If IsArray(blocks) Then
      For Each obj in blocks
        If Rhino.IsBlockInstance(obj) Then
          Rhino.ExplodeBlockInstance obj
          b = b + 1
          Rhino.Print b
        End If
        Rhino.DeleteObject(obj)
      Next
    End If
  Next
```

After all objects are exploded the amount of generated meshes is finally given to the user. If no block instances are exploded at all, the model did not contain any block instances and this is reported to the user. Else the amount of generated meshes is reported.

```
  If b = 0 Then
    Rhino.MessageBox("The model contains no block instances")
  Else
    Rhino.MessageBox("Number of generated meshes: " & b)
  End If
End If
```

If the user chooses for option 3, all meshes will be split into its face curves. First all meshes of the model are placed in an array.

```
If strTekst = arrOptions(2) Then
  m = 0
  meshes = Rhino.ObjectsByType(rhMesh)
```

For each mesh in the model the faces are returned then.

```
  If IsArray(meshes) Then
    For Each obj in meshes
      If Rhino.IsMesh(obj) Then
        arrFaces = Rhino.MeshFaces(obj, vbFalse)
```

Each face goes through a loop then to return the corner points of the triangle. Through the corner points a face curves can be created then. The script counts each added curve. In the end the original mesh object is deleted from the model.

```
        If IsArray(arrFaces) Then
          i = 0
          While i <= UBound(arrFaces)
            arrFace(0) = arrFaces(i)
            arrFace(1) = arrFaces(i+1)
            arrFace(2) = arrFaces(i+2)
            arrFace(3) = arrFaces(i)
            Rhino.AddPolyline(arrFace)
            i = i + 3
            m = m + 1
            Rhino.Print m
          Wend
        End If
        Rhino.DeleteObject(obj)
      End If
    Next
  End If
```

After all meshes are split into its face curves, the amount of generated face curves is given to the user. If no meshes are split at all, the model did not contain any meshes and this is reported to the user. Else the amount of generated face curves is reported.

```
  If m = 0 Then
    Rhino.MessageBox("The model contains no meshes")
  Else
    Rhino.MessageBox("Number of generated face curves: " & m)
  End If
End If
```

If the user chooses for option 4, planar surfaces will be created between the face curves that are obtained from option 3. First all face curves of the model are placed in an array.

```
If strTekst = arrOptions(3) Then
  c = 0
  curves = Rhino.ObjectsByType(rhCurve)
```

For each curve a planar surface will be created then. The original face curve is deleted from the model and the amount of planer surfaces is counted.

```
  If IsArray(curves) Then
    For Each obj in curves
      Rhino.SelectObject obj
      Rhino.Command "_PlanarSrf"
      Rhino.DeleteObject obj
      c = c + 1
      Rhino.Print c
    Next
  End If
```

Finally the amount of generated surfaces is given to the user. If no surfaces are generated at all, the model did not contain any face curves and this is reported to the user. Else the amount of generated surfaces is reported.

```
  If c = 0 Then
    Rhino.MessageBox("The model contains no face curves")
  Else
    Rhino.MessageBox("Number of generated surfaces: " & c)
  End If
End If

End Sub

Preprocessing
```

Appendix I: Script Rotating Lines for Meshes

In this appendix the script for the *Rotating Lines* method to find the outer points of a model that is built up of meshes and generate the enclosing curves for several height levels is discussed in detail. Most of the script is the same as the script of the original method for buildings that are built up of lines, surfaces and solids. These parts of the script are given without further explanation. Only the procedure to find an intersection with a mesh differs from the original script. This procedure will be explained in detail then.

```
Option Explicit

RotatingLines_Mesh_3D
Sub RotatingLines_Mesh_3D

Dim idMesh, m, m_F
Dim P1, P2, rot
Dim xPt, i
Dim angle, parts, p
Dim midpoint, endpoint, point, maxPoint, Pnt()
Dim dx, dy, ex, ey
Dim endx, endy, endz
Dim midx, midy, midz, midp
Dim afst, maxAfst, xmax, ymax
Dim selected
Dim h, height, step, h_end, h_start
Dim surrounded, sur(), surround
Dim delete
Dim loft, crv

Rhino.OsnapMode 0

Rhino.AddLayer "Points"
Rhino.AddLayer "Surrounded", RGB(190,190,190)
Rhino.CurrentLayer "Points"

Rhino.MessageBox ("Pick center location of the building in Top-view")
midpoint = Rhino.GetPoint

parts = Rhino.IntegerBox("Give amount of parts",360)
angle = 360/parts

height = Rhino.RealBox("Give the height of the building")
h_start = Rhino.RealBox("Give starting height")
step = Rhino.RealBox("Give step size")
h_end = (height / step)

ReDim sur(-1)
```

With the following code all meshes of the model are placed in an array. If there are no meshes at all, the script will stop.

```
idMesh = Rhino.ObjectsByType(32)
If IsNull(idMesh) Then Exit Sub
```

```
For h = 0 To (h_end – h_start)
  ReDim Pnt(-1)
  midx = midpoint(0)
  midy = midpoint(1)
  midz = midpoint(2) + h*step
  midp = Array(midx, midy, midz)
  endx = midpoint(0)
  endy = midpoint(1) + 2
  endz = midpoint(2) + h*step
  endpoint = Array(endx, endy, endz)
  For p = 0 To parts
    maxAfst = 0
    xmax = 0
    ymax = 0
```

In stead of lines that rotate around the center location of the model to determine the intersection point, only the direction from the center location is needed to find an intersection with the mesh. To specify the concerning direction, points are used that rotate around the center location with a certain radius.

```
    P1 = midp
    point = Rhino.AddPoint(endpoint)
    rot = Rhino.RotateObject(point, midp, angle*p)
    P2 = Rhino.PointCoordinates(rot)
```

To find the intersection with a mesh, the mesh faces are used.

```
    For m = 0 To UBound(idMesh)
      m_F = Rhino.MeshFaces(idMesh(m), False)
      If IsNull(m_F) Then Exit Sub
```

The intersection is determined with a special function, called *IntersectLineFace*. The function is scripted after the subroutine and called from here. The function depends on the concerning direction and the mesh faces. For a certain direction each mesh face will be checked to see if it lies through the direction of concern. If so, the intersection is added to the *Xpt* array. The code of the function and some further explanation is given at the end of this script.

```
      For i = 0 To UBound(m_F) Step 3
        xPt = IntersectLineFace(P1, P2, m_F(i+0), m_F(i+1), m_F(i+2))
```

When the rotating point lies in a specific direction, the function does not only look for intersections in the direction from the center location to the rotating point. It looks in the lengthened direction, thus also in the direction from the center location to the point that is mirrored in the center location. If, for example, the coordinates of the rotating point at a specific moment are (-2, 2, 0), the function searches for all intersections that lie in the lengthened directions of the imaginary line (-2, 2, 0) , (2, -2, 0). Or, in other words, if the rotating point lies in the second quadrant of the axis system, the function searches for intersections with a mesh in the second and fourth quadrant of the system. This introduces errors for the determination of the most far away intersection point from the center location, because it always lies in one of the two quadrants then. The most far away intersection point in the opposite quadrant is skipped. To solve this problem the script will first check if the intersection point lies in the same quadrant as the rotating point. If so, the intersection point with the largest distance from the center location can be found out of all other intersections that lie in the same quadrant.

```
        If IsArray(xPt) Then
          If (angle*p < 180) Then
            If (xPt(0) - P1(0)) < 0 Then
              dx = xPt(0) - P1(0)
              dy = xPt(1) - P1(1)
              afst = sqr(dx^2 + dy^2)
              If afst > maxAfst Then
                maxAfst = afst
                xmax = xPt(0)
                ymax = xPt(1)
              End If
            End If
          End If
          If (angle*p >= 180) AND (angle*p <= 360) Then
            If (xPt(0) - P1(0)) > 0 Then
              dx = xPt(0) - P1(0)
              dy = xPt(1) - P1(1)
              afst = sqr(dx^2 + dy^2)
              If afst > maxAfst Then
                maxAfst = afst
                xmax = xPt(0)
                ymax = xPt(1)
              End If
            End If
          End If
        End If
      Next
    Next
    If maxAfst > 0 Then
      maxPoint = Array(xmax, ymax, midz)
      Rhino.AddPoint(maxPoint)
      ReDim Preserve Pnt(UBound(Pnt) + 1)
      Pnt(UBound(Pnt)) = maxPoint
    End If
    Rhino.DeleteObject point
  Next
```

When for all directions the most far away intersections are found, the array *Pnt* is filled with all outer points. The first point of the array is then copied and added to the end of the array, so that a closed polyline can be created through the points.

```
  If IsArray(Pnt) AND UBound(Pnt) > 0 Then
    ex = Pnt(0)(0)
    ey = Pnt(0)(1)
    maxPoint = Array(ex, ey, midz)
    Rhino.AddPoint(maxPoint)
    ReDim Preserve Pnt(UBound(Pnt) + 1)
    Pnt(UBound(Pnt)) = maxPoint
    Rhino.CurrentLayer("Surrounded")
    surround = Rhino.AddPolyline(Pnt)
    ReDim Preserve sur(Ubound(sur) + 1)
    sur(UBound(sur)) = surround
    delete = Rhino.ObjectsByLayer("Points")
    Rhino.DeleteObjects delete
    Rhino.CurrentLayer"Points"
  End If
Next
```

As in the original script of the *Rotating Lines* method, the original building model can be removed when for all height steps the enclosing curves are determined.

```
delete = Rhino.ObjectsByType(32)
Rhino.DeleteObjects delete

End Sub
```

After the subroutine the function *IntersectLineFace* is scripted. The function searches for the intersection point between a line segment and the mesh triangles. It is dependent on the coordinates of the center location P1, the rotating point P2 and the corner points Pa, Pb and Pc of the triangles. The following figure shows the principle of the function.

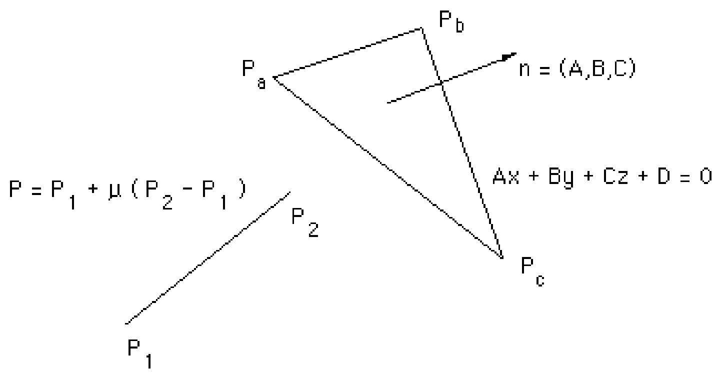


Figure I.1: Principle of the IntersectLineFace function (Internet, [21])

The function searches for the intersection of the line on which the line segment (P1-P2) lies, with the plane on which the triangle lies. It then checks if the intersection point lies along the line segment and if it lies within the triangle. The intersection point can be found by substituting the equation for the line P:

$$P = P_1 + \mu \cdot (P_2 - P_1)$$

into the equation for the plane:

$$A_x + B_y + C_z + D = 0$$

The values of A, B and C are the components of the normal to the plane, which can be found by taking the cross product of two edge vectors of the triangle. The vector cross product gives a vector which is perpendicular to both the vectors that are multiplied. In general, the resulting vector (A * B) is defined by:

$$x = A_y \cdot B_z - B_y \cdot A_z$$
$$y = A_z \cdot B_x - B_z \cdot A_x$$
$$z = A_x \cdot B_y - B_x \cdot A_y$$

Herein are x, y and z the components of the vector (A * B). When this theory is applied to the vectors of the triangle in Figure I.1, it follows:

$$n = (A, B, C) = (P_b - P_a) \text{ cross } (P_c - P_a)$$
$$n_x = (P_{b,y} - P_{a,y}) \cdot (P_{c,z} - P_{a,z}) - (P_{b,z} - P_{a,z}) \cdot (P_{c,y} - P_{a,y})$$
$$n_y = (P_{b,z} - P_{a,z}) \cdot (P_{c,x} - P_{a,x}) - (P_{b,x} - P_{a,x}) \cdot (P_{c,z} - P_{a,z})$$
$$n_z = (P_{b,x} - P_{a,x}) \cdot (P_{c,y} - P_{a,y}) - (P_{b,y} - P_{a,y}) \cdot (P_{c,x} - P_{a,x})$$

The variable D in the equation of the plane follows from substituting one vertex into the equation for the plane:

$$AP_{a,x} + BP_{a,y} + CP_{a,z} = -D$$

Now the variable μ of the equation for the line P can be determined. From that value the intersection point can be found using the equation of the line.

$$\mu = \frac{(D + AP_{1,x} + BP_{1,y} + CP_{1,z})}{A(P_{1x} - P_{2x}) + B(P_{1y} - P_{2y}) + C(P_{1z} - P_{2z})}$$

If the line P is parallel to the plane, they will not intersect and the denominator in the equation for μ is 0. The intersection point lies on the line segment if the value of μ is between 0 and 1.

Finally it must be checked if the intersection point between the line and the plane of the triangle lies within the triangle that is bounded by the corner points P_a, P_b and P_c. If a point P lies on the interior of the triangle, the sum of the internal angles of the point is 2Π. If a point lies outside the triangle, the sum of the internal angles is smaller than 2Π. The following picture shows this principle.

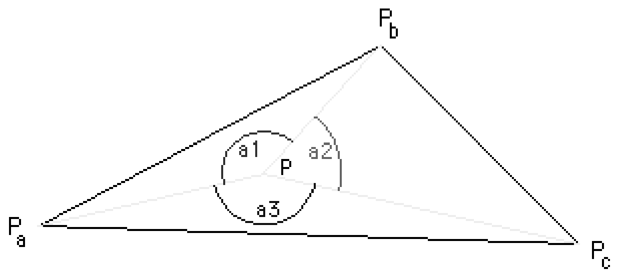


Figure I.2: Internal angles (Internet, [21])

If the point that is checked to see if it lies in the interior is P, than the unit vectors of the point are:

$$P_{a,1} = \frac{P_a - P}{|P_a - P|} \quad ; \quad P_{a,2} = \frac{P_b - P}{|P_b - P|} \quad ; \quad \mathrm{P}_{a,3} = \frac{P_c - P}{|P_c - P|}$$

The angles are then:

$$a_1 = a\cos(P_{a1}P_{a2})$$
$$a_2 = a\cos(P_{a2}P_{a3})$$
$$a_3 = a\cos(P_{a3}P_{a1})$$

The above explained procedure is implemented in the *IntersectLineFace* function. The script of this function is given below. The function starts with the declaration of some variables.

```
Function IntersectLineFace(ByVal P1, ByVal P2, ByVal Pa, ByVal Pb,
                           ByVal Pc)
  IntersectLineFace = Null
  Dim d
  Dim a1, a2, a3
  Dim total, denom, mu
  Dim n, p(2)
```

The components of the normal to the plane are found by taking the cross product of two edge vectors of the triangle. In Rhinoceros the vector between two points is created with the *Rhino.VectorCreate* command. The cross product of the two vectors is determined with the *Rhino.VectorCrossProduct* command. With the *Rhino.VectorUnitize* command the vector n is unitized, resulting in a 3D vector.

```
  n = Rhino.VectorCrossProduct(Rhino.VectorCreate(Pb, Pa),
                               Rhino.VectorCreate(Pc, Pa))

  Rhino.VectorUnitize(n)
```

Then the variable d from the equation of the plane is defined.

```
  d = -n(0) * pa(0) - n(1) * pa(1) - n(2) * pa(2)
```

After that the position on the line that intersects the plane is calculated.

```
  denom = n(0) * (p2(0) - p1(0)) + n(1) * (p2(1) - p1(1)) + n(2) *
          (p2(2) - p1(2))
```

Then it will be checked if the line and the plane are parallel. If |P1 - P2| is very small, no intersection will be returned and the function stops.

```
  If (Abs(denom) < 0.00001) Then Exit Function
```

If the line and the plane are not parallel, the value μ of the equation for the line P can be derived:

```
  mu = -(d + n(0) * p1(0) + n(1) * p1(1) + n(2) * p1(2)) / denom
```

With the value for μ the equation of the line P can be defined:

```
  p(0) = p1(0) + mu * (p2(0) - p1(0))
  p(1) = p1(1) + mu * (p2(1) - p1(1))
  p(2) = p1(2) + mu * (p2(2) - p1(2))
```

Finally it is checked if the intersection point between the line and the plane of the triangle lies within the triangle. To determine the internal angles of the intersection point another function is used. This function is called *ThreePointAngle* and is scripted after the *IntersectLineFace* function. The function uses the intersection point p and two corner points of the triangle.

```
  a1 = ThreePointAngle(p, Pa, Pb)
  a2 = ThreePointAngle(p, Pb, Pc)
  a3 = ThreePointAngle(p, Pc, Pa)
  total = a1 +  a2 + a3
```

If the total of the internal angles is larger than 360, the corner points P_a, P_b and P_c do not clearly indicate a triangle. The function stops then.

```
  If (Abs(total - 360) > 0.0001) Then Exit Function
```

If the intersection point is finally found, the result is applied to the function. The superior *Rotating Lines* script uses this result then to determine the outer point for a certain location.

```
  IntersectLineFace = p

End Function
```

Finally the *ThreePointAngle* function is scripted. The function calculates the smallest angle in degrees between two line segments. With the *Rhino.Angle2* command, the angle between two lines is measured. The values *pBase*, *pLeft* and *pRight* correspond to respectively *p*, *Pa* and *Pb* that were defined when the *ThreePointAngle* function was called on in the *IntersectLineFace* function.

```
Function ThreePointAngle(ByVal pBase, ByVal pLeft, ByVal pRight)

  Dim A
  A = Rhino.Angle2(Array(pBase, pLeft), Array(pBase, pRight))
```

The *Rhino.Angle2* command returns an array containing the angle in degrees and the reflex angle in degrees. The reflex angle is an angle of more than 180 but less than 360 degrees. To make sure that the function is not set to a value that is larger then 360 degrees, the script checks whether the angle is smaller or larger than the reflex angle. If the reflex angle is larger then the function is set to this value, else the function is set to the normal angle.

```
  If A(0) < A(1) Then
    ThreePointAngle = A(0)
  Else
    ThreePointAngle = A(1)
  End If

End Function
```

Appendix J: Script Curve simplification

In this appendix the script of the *Curve simplification* method to simplify the enclosing curves with the NURBS fitting technique is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable terms are declared that will be used in the subroutine. All curves of the model are placed in the *arrCurves* array. If there is no curve at all, the script gives a message to the user and the process stops.

```
Option Explicit

Sub Curve_simplification

Dim arrCurves, crv, strObject, arrPoints, arrPoint
Dim ContrPoints(), p
Dim arrOptions(1), strTekst
Dim intDegree, intKnotCount, intCPCount
Dim arrKnots(), arrWeights(), weight
Dim k, w

arrCurves = Rhino.ObjectsByType(4)
If Not IsArray(arrCurves) Then
  Rhino.MessageBox("The model contains no curves.")
  Exit Sub
End If
```

The user is asked for the weight value that has to be given to the control points. Then a loop starts that goes through all enclosing curves of the model. In each loop the length of *ContrPoints* array is first set to -1. In this array the control points of the enclosing curves will be stored. Because on forehand it is not known how many control points have to be stored in the array, the length is first set to -1. Before a point is added, the array is lengthened by 1. The point is then stored at this new location.

```
Weight = Rhino.RealBox("Give the weight value of the control
                       points")
For Each strObject in arrCurves
  ReDim ContrPoints(-1)
```

For each curve the control points will be returned first. If some control points are found, they are added to the *ContrPoints* array.

```
  If Rhino.IsCurve(strObject) Then
    arrPoints = Rhino.CurvePoints(strObject)
    If IsArray(arrPoints) Then
      For Each arrPoint in arrPoints
        ReDim Preserve ContrPoints(UBound(ContrPoints) + 1)
        ContrPoints(UBound(ContrPoints)) = arrPoint
      Next
    End If
  End If
```

The next step is to add a NURBS curve through the control points. Because a cubic curve is the best option for fitting a smooth polynomial, the degree of the NURBS curve is set to 3. The required number of knots is the number of control points plus the degree of the curve minus 1. To determine the number of knots, first the number of control points is derived. Then the number of knots is calculated.

```
'-------------------------------------------------------
'Add NURBS curve
'-------------------------------------------------------
intDegree = 3
intCPCount = UBound(ContrPoints) + 1
intKnotCount = intCPCount + intDegree - 1
```

The *arrWeights* array is then re-declared to a length that is equal to the number of control points of the curve. The weight value for each control point will be stored in this array. The weight values for the first and last control point are set to 1.0 by default. The other weight values are set to the user-specified weight value.

```
ReDim arrWeights(intCPCount-1)
arrWeights(0) = 1.0
arrWeights(UBound(arrWeights)) = 1.0
For w = 1 To (UBound(arrWeights) - 1)
  arrWeights(w) = weight
Next
```

Then the *arrKnots* array is re-declared to a length that is equal to the number of required knots. This number depends on the amount of control points of the curve and was defined already. Because the NURBS curve has to go through the first and last control point, full-multiplicity is required there. This means that the first and last three knot values of the knot list have to be equal. The first three knot values are therefore set to 0.

```
ReDim arrKnots(intKnotCount-1)
For k = 0 to 2
  arrKnots(k) = 0
Next
```

Because the knot list may only contain knot values in increasing order, it has to be calculated which value the last three knots have to obtain. Because the inner knots are all single knots, the value of each knot can be raised by 1. From this fact the value of the last three knots can be determined. First a value is assigned to all inner knots.

```
For k = 3 To (UBound(arrKnots) - 3)
  arrKnots(k) = k - 2
Next
```

Then the values to the last three knots are assigned. Because of the required full-multiplicity, the value for these three knots is the same.

```
For k = (UBound(arrKnots) - 2) To (UBound(arrKnots))
  arrKnots(k) = (UBound(arrKnots) - 4)
Next
```

Finally the NURBS curve can be added through the control points. The curve depends on the knot values, degree of the curve and the weight values. When for each enclosing curve a NURBS curve is generated, the original curves are removed from the model.

```
  Rhino.AddNurbsCurve ContrPoints, arrKnots, intDegree, arrWeights
Next

For Each strObject in arrCurves
  Rhino.Deleteobject strObject
Next

End Sub

Curve_simplification
```

Appendix K: Script Loft

In this appendix the script of the *Loft* method to loft a surface through the enclosing curves and create a planar surface through the topmost and bottom curves is discussed in detail. The script starts with the *Option Explicit* command and the definition of the subroutine. In the following steps the variable terms are declared that will be used in the subroutine. Then all eventual selected objects will be unselected and all enclosing curves of the model are placed in the *arrCurves* array. If the model contains no curves at all, the script gives a message to the user and the process stops.

```
Option Explicit

Sub Loft

Dim strObject
Dim arrSurfaces, arrCurves, arrPoints, crvPoint
Dim arrSorted
Dim arrZ()
Dim i, crv, z_val
Dim coord, z, l
ReDim arrZ(-1)

Rhino.UnselectAllObjects

arrCurves = Rhino.ObjectsByType(4)
If Not IsArray(arrCurves) Then
  Rhino.MessageBox("The model contains no curves.")
  Exit Sub
End If
```

Then the script will determine the order of the curves. The lofting process has to start with the lowest curve and end with the highest curve. The ordering starts with returning the control points of the various curves. For each curve, the z-coordinate of the first control point is returned and placed in the *arrZ* array. Because on forehand it is not know how many curves have to be placed in the array, the array is lengthened each time before the value is added to the array.

```
If UBound(arrCurves) > 0 Then
  For Each crv in arrCurves
    If Rhino.IsCurve(crv) Then
      arrPoints = Rhino.CurvePoints(crv)
      If IsArray(arrPoints) Then
        z = arrPoints(0)(2)
        ReDim Preserve arrZ(UBound(arrZ) + 1)
        arrZ(UBound(arrZ)) = z
```

Then the curve is named to its height level z. While lofting, the various curves can now be called on in the right order.

```
        Rhino.ObjectName crv, z
      End If
    End If
  Next
```

The array with the z values of the various curves is ordered. The values are placed in a new array, *arrSorted*.

```
  arrSorted = Rhino.SortNumbers(arrZ)
```

In the next loop the surfaces will be lofted through the enclosing curves. The *arrSorted* array contains the height levels of the curves. Starting from the lowest value, this height level is now first asked for. Because the corresponding curve has the same name as its height, this value is now used to call the curve itself. The curve is then selected.

```
  For i = 0 To (UBound(arrSorted) - 1)
    z_val = arrSorted(i)
    crv = Rhino.ObjectsByName(z_val)
    Rhino.SelectObjects crv
```

Because the surface is lofted between two curves, the curve that lies one level higher is selected in the same way. Then a straight surface is lofted through the curves. After that the two curves are unselected again.

```
    z_val = arrSorted(i+1)
    crv = Rhino.ObjectsByName(z_val)
    Rhino.SelectObjects crv
    Rhino.Command("-loft enter type=straight enter")
    Rhino.UnselectAllObjects
  Next
```

The above explained process is looped until the final surface is lofted trough the two upmost curves. To close the model, two planar surfaces have to be created through the bottom and top curves. These curves are first selected.

```
  z_val = arrSorted(0)
  crv = Rhino.ObjectsByName(z_val)
  Rhino.SelectObjects crv
  z_val = arrSorted(UBound(arrSorted))
  crv = Rhino.ObjectsByName(z_val)
  Rhino.SelectObjects crv
```

Then the planar surfaces are added through the curves. The original curves are removed from the model.

```
  Rhino.Command("_PlanarSrf")
  For Each crv in arrCurves
    Rhino.DeleteObject crv
  Next
```

Finally the lofted surface and the two planar surfaces are joined to form one single surface model.

```
  arrSurfaces = Rhino.ObjectsByType(8 + 16, vbTrue)
  Rhino.Command"_Join"
  Rhino.UnselectAllObjects
End If

End Sub

Loft
```

Appendix L: Script to generate the journal file

In this appendix the Visual Basic macro script to generate the journal file for the creation of the computational domain is discussed in detail. The macro is part of an excel sheet that contains the user-specified maximum height and radius of the research area. The dimensions of the computational domain are dependent on these values. The script starts with the definition of the subroutine and the declaration of some variables.

```
Private Sub CommandButton1_Click()

Dim h As Double
Dim r As Double
```

With the following commands the journal file Domain.jou is created. The *FileSystemObject* command is used to access the server's file system. The journal file is saved on the D drive. If the user wants to save the file on another drive, it has to be changed here.

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("d:\Domain.jou", True)
```

The maximum height of all buildings and the radius of the research area are derived from cells C4 and C5 of the Excel sheet.

```
h = Range("C4").Value
r = Range("C5").Value
```

Gambit can be used to generate meshes for several CFD solvers. Because Fluent 6 is used to solve the flow problems, the solver is set to Fluent 5/6.

```
a.Writeline ("   solver select ""Fluent 5/6""    ")
```

Now the first four vertices of the domain are created. The vertices belong to the inlet boundary and are located at -20h in x-direction from the origin. The two outer vertices have a y-coordinate of -10h respectively +10h. The y-coordinate of the two inner vertices depend on the radius of the research area. The y-coordinate of the vertices is the distance in y-direction between the endpoint of a line with radius r and the origin when the line is drawn under 45 degrees. The y-coordinates are then – ((r^2)/2)^1/2 respectively + ((r^2)/2)^1/2. The z-coordinate of all vertices is 0.

```
a.Writeline ("   vertex create coordinates -" & 20 * h & " -" & 10 *
             h & " 0  ")
a.Writeline ("   vertex create coordinates -" & 20 * h & " -" & ((r ^
             2) / 2) ^ (1 / 2) & " 0  ")
a.Writeline ("   vertex create coordinates -" & 20 * h & " " & ((r ^
             2) / 2) ^ (1 / 2) & " 0   ")
a.Writeline ("   vertex create coordinates -" & 20 * h & " " & 10 * h
             & " 0   ")
```

The four vertices are then copied three times in the positive x-direction. The vertices are first copied to the negative x-coordinate with the same value as the y-coordinate of the inner two vertices. Then the vertices are copied to the positive x-coordinate with the same value as the y-coordinate of the inner vertices. Finally the vertices are copied to the positive x-coordinate +20h.

```
a.Writeline ("   vertex cmove ""vertex.1"" ""vertex.2"" ""vertex.3""
             ""vertex.4"" multiple 1 offset \    ")
a.Writeline ("     " & (20 * h - ((r ^ 2) / 2) ^ (1 / 2)) & " 0 0  ")
a.Writeline ("   vertex cmove ""vertex.5"" ""vertex.6"" ""vertex.7""
             ""vertex.8"" multiple 1 offset \    ")
a.Writeline ("     " & 2 * ((r ^ 2) / 2) ^ (1 / 2) & " 0 0   ")
a.Writeline ("   vertex cmove ""vertex.9"" ""vertex.10""
             ""vertex.11"" ""vertex.12"" multiple 1 offset \ ")
a.Writeline ("     " & (20 * h - ((r ^ 2) / 2) ^ (1 / 2)) & " 0 0  ")
```

Then a vertex is created at the origin. This point is the center location of the domain.

```
a.Writeline ("   vertex create coordinates 0 0 0 ")
```

Now edges are created between the vertices.

```
a.Writeline ("   edge create straight ""vertex.1"" ""vertex.5""  ")
a.Writeline ("   edge create straight ""vertex.5"" ""vertex.9""  ")
a.Writeline ("   edge create straight ""vertex.9"" ""vertex.13"" ")
a.Writeline ("   edge create straight ""vertex.13"" ""vertex.14""
             ""vertex.15"" ""vertex.16""    ")
a.Writeline ("   edge create straight ""vertex.1"" ""vertex.2""
             ""vertex.3"" ""vertex.4"" ""vertex.8"" \ ")
a.Writeline ("     ""vertex.12"" ""vertex.16""   ")
a.Writeline ("   edge create straight ""vertex.5"" ""vertex.6""  ")
a.Writeline ("   edge create straight ""vertex.9"" ""vertex.10"" ")
a.Writeline ("   edge create straight ""vertex.7"" ""vertex.8""  ")
a.Writeline ("   edge create straight ""vertex.11"" ""vertex.12""  ")
a.Writeline ("   edge create straight ""vertex.2"" ""vertex.6""  ")
a.Writeline ("   edge create straight ""vertex.10"" ""vertex.14""  ")
a.Writeline ("   edge create straight ""vertex.3"" ""vertex.7""  ")
a.Writeline ("   edge create straight ""vertex.11"" ""vertex.15""  ")
```

Around the center location of the domain, a circular edge is created through the four inner vertices. The circle now automatically has the radius r. After the circle is created, the center point is removed from the model.

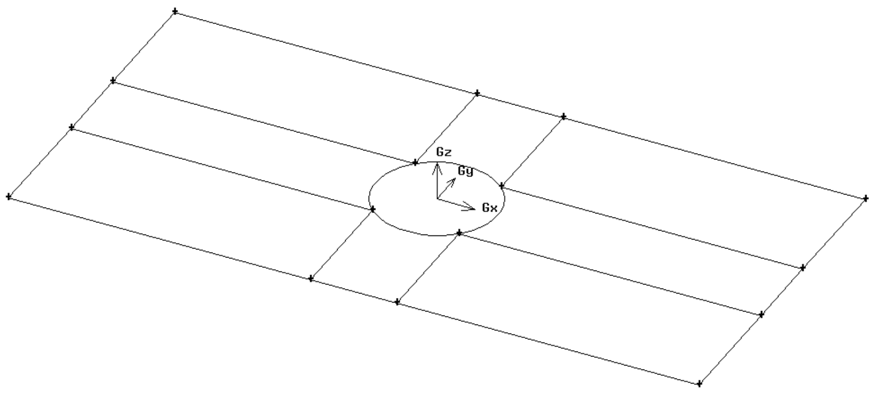
```
a.Writeline ("   edge create center2points ""vertex.17"" ""vertex.6""
             ""vertex.7"" circle    ")
a.Writeline ("   vertex delete ""vertex.17"" ")
```

Now faces are created by stitching the edges together.

```
a.Writeline ("   face create wireframe ""edge.1"" ""edge.7""
             ""edge.13"" ""edge.17"" real    ")
a.Writeline ("   vertex create coordinates 0 0 0 ")
a.Writeline ("   edge delete ""edge.21"" lowertopology   ")
a.Writeline ("   edge create center2points ""vertex.17"" ""vertex.6""
             ""vertex.7"" minarc arc    ")
a.Writeline ("   edge create center2points ""vertex.17"" ""vertex.7""
             ""vertex.11"" minarc arc   ")
a.Writeline ("   edge create center2points ""vertex.17""
             ""vertex.11"" ""vertex.10"" minarc arc  ")
a.Writeline ("   edge create center2points ""vertex.17""
             ""vertex.10"" ""vertex.6"" minarc arc   ")
a.Writeline ("   vertex delete ""vertex.17"" ")
a.Writeline ("   face create wireframe ""edge.2"" ""edge.13""
             ""edge.14"" ""edge.24"" real   ")
```

```
a.Writeline ("   face create wireframe ""edge.3"" ""edge.4""
             ""edge.14"" ""edge.18"" real    ")
a.Writeline ("   face create wireframe ""edge.8"" ""edge.17""
             ""edge.19"" ""edge.21"" real   ")
a.Writeline ("   face create wireframe ""edge.21"" ""edge.22""
             ""edge.23"" ""edge.24"" real  ")
a.Writeline ("   face create wireframe ""edge.5"" ""edge.18""
             ""edge.20"" ""edge.23"" real   ")
a.Writeline ("   face create wireframe ""edge.9"" ""edge.10""
             ""edge.15"" ""edge.19"" real   ")
a.Writeline ("   face create wireframe ""edge.11"" ""edge.15""
             ""edge.16"" ""edge.22"" real  ")
a.Writeline ("   face create wireframe ""edge.6"" ""edge.12""
             ""edge.16"" ""edge.20"" real   ")
```

This first phase of the script has created the vertices, edges and faces of the bottom of the computational domain. The result is shown in the following picture.



The faces are now copied twice in z-direction. First over a distance 5h; then these new faces are copied over a distance 15h.

```
a.Writeline ("   face cmove ""face.1"" ""face.2"" ""face.3""
             ""face.4"" ""face.5"" ""face.6"" ""face.7"" \   ")
a.Writeline ("     ""face.8"" ""face.9"" multiple 1 offset 0 0 " & 5
             * h & "   ")
a.Writeline ("   face cmove ""face.10"" ""face.11"" ""face.12""
             ""face.13"" ""face.14"" ""face.15"" \    ")
a.Writeline ("     ""face.16"" ""face.17"" ""face.18"" multiple 1
             offset 0 0 " & 15 * h & " ")
```

Then all edges are selected and connected together.

```
a.Writeline ("   edge connect ""edge.1"" ""edge.3"" ""edge.4""
             ""edge.5"" ""edge.6"" ""edge.7"" ""edge.8"" \ ")
a.Writeline ("     ""edge.9"" ""edge.10"" ""edge.11"" ""edge.12""
             ""edge.13"" ""edge.14"" ""edge.15"" \  ")
a.Writeline ("     ""edge.16"" ""edge.17"" ""edge.18"" ""edge.19""
             ""edge.20"" ""edge.21"" ""edge.22"" \ ")
a.Writeline ("     ""edge.23"" ""edge.24"" ""edge.25"" ""edge.26""
             ""edge.27"" ""edge.28"" ""edge.29"" \ ")
a.Writeline ("     ""edge.30"" ""edge.31"" ""edge.32"" ""edge.33""
             ""edge.34"" ""edge.35"" ""edge.36"" \ ")
a.Writeline ("     ""edge.37"" ""edge.38"" ""edge.39"" ""edge.40""
             ""edge.41"" ""edge.42"" ""edge.43"" \ ")
```

```
a.Writeline ("     ""edge.44"" ""edge.45"" ""edge.46"" ""edge.47""
             ""edge.48"" ""edge.49"" ""edge.50"" \ ")
a.Writeline ("     ""edge.51"" ""edge.52"" ""edge.53"" ""edge.54""
             ""edge.55"" ""edge.56"" ""edge.57"" \ ")
a.Writeline ("     ""edge.58"" ""edge.59"" ""edge.60"" ""edge.61""
             ""edge.62"" ""edge.63"" ""edge.64"" \ ")
a.Writeline ("     ""edge.65"" ""edge.66"" ""edge.67"" ""edge.68""
             ""edge.69"" ""edge.70"" ""edge.71"" \ ")
a.Writeline ("     ""edge.72"" ""edge.73"" ""edge.74"" ""edge.75""
             ""edge.76"" ""edge.77"" ""edge.78"" \ ")
a.Writeline ("     ""edge.79"" ""edge.80"" ""edge.81"" ""edge.82""
             ""edge.83"" ""edge.84"" ""edge.85"" \ ")
a.Writeline ("     ""edge.86"" ""edge.87"" ""edge.88"" ""edge.89""
             ""edge.90"" ""edge.91"" ""edge.92"" \ ")
a.Writeline ("     ""edge.94"" ""edge.95"" ""edge.96"" ""edge.2""
             ""edge.93"" real   ")
```

After that edges are drawn in vertical direction between the various vertices.

```
a.Writeline ("   edge create straight ""vertex.1"" ""vertex.17"" ")
a.Writeline ("   vertex modify ""vertex.1"" ""vertex.3"" ""vertex.4""
             ""vertex.5"" ""vertex.6"" \    ")
a.Writeline ("     ""vertex.7"" ""vertex.8"" ""vertex.9""
             ""vertex.10"" ""vertex.11"" ""vertex.12"" \    ")
a.Writeline ("     ""vertex.13"" ""vertex.14"" ""vertex.15""
             ""vertex.16"" ""vertex.2"" color ""green""  ")
a.Writeline ("   edge create straight ""vertex.2"" ""vertex.20"" ")
a.Writeline ("   edge create straight ""vertex.3"" ""vertex.30"" ")
a.Writeline ("   edge create straight ""vertex.4"" ""vertex.42"" ")
a.Writeline ("   edge create straight ""vertex.5"" ""vertex.18"" ")
a.Writeline ("   edge create straight ""vertex.6"" ""vertex.19"" ")
a.Writeline ("   edge create straight ""vertex.7"" ""vertex.31"" ")
a.Writeline ("   edge create straight ""vertex.8"" ""vertex.43"" ")
a.Writeline ("   edge create straight ""vertex.9"" ""vertex.22"" ")
a.Writeline ("   edge create straight ""vertex.10"" ""vertex.23""  ")
a.Writeline ("   edge create straight ""vertex.11"" ""vertex.35""  ")
a.Writeline ("   edge create straight ""vertex.12"" ""vertex.46""  ")
a.Writeline ("   edge create straight ""vertex.13"" ""vertex.26""  ")
a.Writeline ("   edge create straight ""vertex.14"" ""vertex.27""  ")
a.Writeline ("   edge create straight ""vertex.15"" ""vertex.38""  ")
a.Writeline ("   edge create straight ""vertex.16"" ""vertex.50""  ")
a.Writeline ("   edge create straight ""vertex.17"" ""vertex.53""  ")
a.Writeline ("   edge create straight ""vertex.18"" ""vertex.54""  ")
a.Writeline ("   edge create straight ""vertex.19"" ""vertex.55""  ")
a.Writeline ("   edge create straight ""vertex.20"" ""vertex.56""  ")
a.Writeline ("   edge create straight ""vertex.22"" ""vertex.58""  ")
a.Writeline ("   edge create straight ""vertex.23"" ""vertex.59""  ")
a.Writeline ("   edge create straight ""vertex.26"" ""vertex.62""  ")
a.Writeline ("   edge create straight ""vertex.27"" ""vertex.63""  ")
a.Writeline ("   edge create straight ""vertex.30"" ""vertex.66""  ")
a.Writeline ("   edge create straight ""vertex.31"" ""vertex.67""  ")
a.Writeline ("   edge create straight ""vertex.35"" ""vertex.71""  ")
a.Writeline ("   edge create straight ""vertex.38"" ""vertex.74""  ")
a.Writeline ("   edge create straight ""vertex.42"" ""vertex.78""  ")
a.Writeline ("   edge create straight ""vertex.43"" ""vertex.79""  ")
a.Writeline ("   edge create straight ""vertex.46"" ""vertex.82""  ")
a.Writeline ("   edge create straight ""vertex.50"" ""vertex.86""  ")
```

Now faces are created by stitching the edges together.

```
a.Writeline ("   face create wireframe ""edge.1"" ""edge.25""
             ""edge.95"" ""edge.99"" real    ")
a.Writeline ("   face create wireframe ""edge.99"" ""edge.103""
             ""edge.2"" ""edge.29"" real  ")
a.Writeline ("   face create wireframe ""edge.3"" ""edge.33""
             ""edge.103"" ""edge.107"" real ")
a.Writeline ("   face create wireframe ""edge.107"" ""edge.108""
             ""edge.4"" ""edge.34"" real ")
a.Writeline ("   face create wireframe ""edge.108"" ""edge.109""
             ""edge.5"" ""edge.45"" real ")
a.Writeline ("   face create wireframe ""edge.6"" ""edge.57""
             ""edge.109"" ""edge.110"" real ")
a.Writeline ("   face create wireframe ""edge.106"" ""edge.110""
             ""edge.12"" ""edge.58"" real     ")
a.Writeline ("   face create wireframe ""edge.102"" ""edge.106""
             ""edge.11"" ""edge.53"" real     ")
a.Writeline ("   face create wireframe ""edge.98"" ""edge.102""
             ""edge.10"" ""edge.50"" real ")
a.Writeline ("   face create wireframe ""edge.97"" ""edge.98""
             ""edge.9"" ""edge.49"" real    ")
a.Writeline ("   face create wireframe ""edge.96"" ""edge.97""
             ""edge.8"" ""edge.37"" real    ")
a.Writeline ("   face create wireframe ""edge.95"" ""edge.96""
             ""edge.7"" ""edge.28"" real    ")
a.Writeline ("   face create wireframe ""edge.96"" ""edge.17""
             ""edge.100"" ""edge.27"" real ")
a.Writeline ("   face create wireframe ""edge.19"" ""edge.38""
             ""edge.97"" ""edge.101"" real ")
a.Writeline ("   face create wireframe ""edge.18"" ""edge.35""
             ""edge.104"" ""edge.108"" real     ")
a.Writeline ("   face create wireframe ""edge.20"" ""edge.46""
             ""edge.105"" ""edge.109"" real     ")
a.Writeline ("   face create wireframe ""edge.13"" ""edge.26""
             ""edge.99"" ""edge.100"" real ")
a.Writeline ("   face create wireframe ""edge.14"" ""edge.30""
             ""edge.103"" ""edge.104"" real     ")
a.Writeline ("   face create wireframe ""edge.15"" ""edge.51""
             ""edge.101"" ""edge.102"" real     ")
a.Writeline ("   face create wireframe ""edge.16"" ""edge.54""
             ""edge.105"" ""edge.106"" real     ")
a.Writeline ("   face create wireframe ""edge.21"" ""edge.39""
             ""edge.100"" ""edge.101"" real     ")
a.Writeline ("   face create wireframe ""edge.22"" ""edge.42""
             ""edge.101"" ""edge.105"" real     ")
a.Writeline ("   face create wireframe ""edge.23"" ""edge.43""
             ""edge.104"" ""edge.105"" real     ")
a.Writeline ("   face create wireframe ""edge.24"" ""edge.31""
             ""edge.100"" ""edge.104"" real     ")
a.Writeline ("   face create wireframe ""edge.28"" ""edge.64""
             ""edge.111"" ""edge.114"" real     ")
a.Writeline ("   face create wireframe ""edge.37"" ""edge.73""
             ""edge.114"" ""edge.119"" real     ")
a.Writeline ("   face create wireframe ""edge.49"" ""edge.85""
             ""edge.119"" ""edge.123"" real     ")
a.Writeline ("   face create wireframe ""edge.25"" ""edge.61""
             ""edge.111"" ""edge.112"" real     ")
a.Writeline ("   face create wireframe ""edge.112"" ""edge.29""
             ""edge.65"" ""edge.115"" real    ")
```

```
a.Writeline ("   face create wireframe ""edge.115"" ""edge.33""
             ""edge.69"" ""edge.117"" real    ")
a.Writeline ("   face create wireframe ""edge.34"" ""edge.70""
             ""edge.117"" ""edge.118"" real    ")
a.Writeline ("   face create wireframe ""edge.45"" ""edge.81""
             ""edge.118"" ""edge.122"" real    ")
a.Writeline ("   face create wireframe ""edge.57"" ""edge.93""
             ""edge.122"" ""edge.126"" real    ")
a.Writeline ("   face create wireframe ""edge.123"" ""edge.124""
             ""edge.50"" ""edge.86"" real    ")
a.Writeline ("   face create wireframe ""edge.124"" ""edge.125""
             ""edge.53"" ""edge.89"" real    ")
a.Writeline ("   face create wireframe ""edge.125"" ""edge.126""
             ""edge.58"" ""edge.94"" real    ")
a.Writeline ("   face create wireframe ""edge.27"" ""edge.63""
             ""edge.114"" ""edge.113"" real    ")
a.Writeline ("   face create wireframe ""edge.38"" ""edge.74""
             ""edge.119"" ""edge.120"" real    ")
a.Writeline ("   face create wireframe ""edge.35"" ""edge.71""
             ""edge.116"" ""edge.118"" real    ")
a.Writeline ("   face create wireframe ""edge.46"" ""edge.82""
             ""edge.121"" ""edge.122"" real    ")
a.Writeline ("   face create wireframe ""edge.26"" ""edge.62""
             ""edge.112"" ""edge.113"" real    ")
a.Writeline ("   face create wireframe ""edge.30"" ""edge.66""
             ""edge.115"" ""edge.116"" real    ")
a.Writeline ("   face create wireframe ""edge.51"" ""edge.87""
             ""edge.120"" ""edge.124"" real    ")
a.Writeline ("   face create wireframe ""edge.54"" ""edge.90""
             ""edge.121"" ""edge.125"" real    ")
```

After that all faces are selected and connected.

```
a.Writeline ("   face connect ""face.1"" ""face.2"" ""face.3""
             ""face.4"" ""face.5"" ""face.6"" ""face.7"" \ ")
a.Writeline ("     ""face.8"" ""face.9"" ""face.10"" ""face.19""
             ""face.11"" ""face.12"" ""face.13"" \   ")
a.Writeline ("     ""face.14"" ""face.15"" ""face.16"" ""face.17""
             ""face.18"" ""face.20"" ""face.21"" \ ")
a.Writeline ("     ""face.22"" ""face.23"" ""face.24"" ""face.25""
             ""face.26"" ""face.27"" ""face.28"" \ ")
a.Writeline ("     ""face.29"" ""face.30"" ""face.31"" ""face.32""
             ""face.33"" ""face.34"" ""face.35"" \ ")
a.Writeline ("     ""face.36"" ""face.37"" ""face.38"" ""face.39""
             ""face.40"" ""face.41"" ""face.42"" \ ")
a.Writeline ("     ""face.43"" ""face.44"" ""face.45"" ""face.46""
             ""face.47"" ""face.48"" ""face.49"" \ ")
a.Writeline ("     ""face.50"" ""face.51"" ""face.52"" ""face.53""
             ""face.54"" ""face.55"" ""face.56"" \ ")
a.Writeline ("     ""face.57"" ""face.58"" ""face.59"" ""face.60""
             ""face.61"" ""face.62"" ""face.63"" \ ")
a.Writeline ("     ""face.64"" ""face.65"" ""face.66"" ""face.67""
             ""face.68"" ""face.69"" ""face.70"" \ ")
a.Writeline ("     ""face.71"" real  ")
a.Writeline ("   face create wireframe ""edge.39"" ""edge.75""
             ""edge.113"" ""edge.120"" real    ")
a.Writeline ("   face create wireframe ""edge.42"" ""edge.78""
             ""edge.120"" ""edge.121"" real    ")
a.Writeline ("   face create wireframe ""edge.43"" ""edge.79""
             ""edge.116"" ""edge.121"" real    ")
```

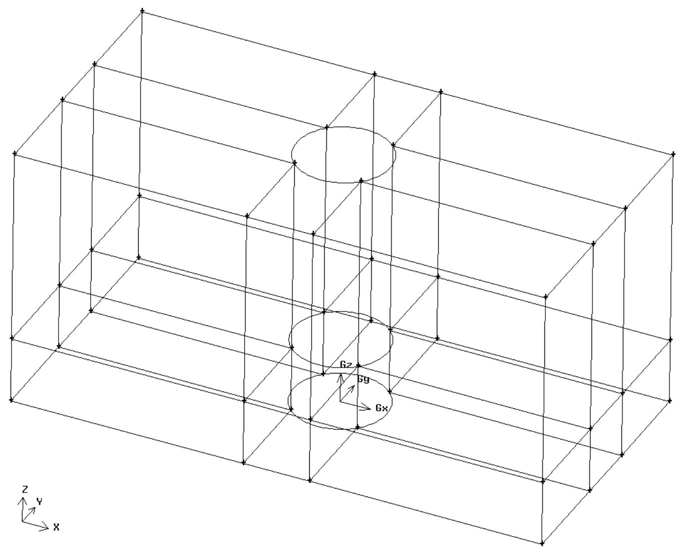
```
a.Writeline ("   face create wireframe ""edge.31"" ""edge.67""
             ""edge.113"" ""edge.116"" real    ")
a.Writeline ("   face connect ""face.1"" ""face.2"" ""face.3""
             ""face.4"" ""face.5"" ""face.6"" ""face.7"" \ ")
a.Writeline ("     ""face.8"" ""face.9"" ""face.10"" ""face.19""
             ""face.11"" ""face.12"" ""face.13"" \   ")
a.Writeline ("     ""face.14"" ""face.15"" ""face.16"" ""face.17""
             ""face.18"" ""face.20"" ""face.21"" \ ")
a.Writeline ("     ""face.22"" ""face.23"" ""face.24"" ""face.25""
             ""face.26"" ""face.27"" ""face.28"" \ ")
a.Writeline ("     ""face.29"" ""face.30"" ""face.31"" ""face.32""
             ""face.33"" ""face.34"" ""face.35"" \ ")
a.Writeline ("     ""face.36"" ""face.37"" ""face.38"" ""face.39""
             ""face.40"" ""face.41"" ""face.42"" \ ")
a.Writeline ("     ""face.43"" ""face.44"" ""face.45"" ""face.46""
             ""face.47"" ""face.48"" ""face.49"" \ ")
a.Writeline ("     ""face.50"" ""face.51"" ""face.52"" ""face.53""
             ""face.54"" ""face.55"" ""face.56"" \ ")
a.Writeline ("     ""face.57"" ""face.58"" ""face.59"" ""face.60""
             ""face.61"" ""face.62"" ""face.63"" \ ")
a.Writeline ("     ""face.64"" ""face.65"" ""face.66"" ""face.67""
             ""face.68"" ""face.69"" ""face.70"" \ ")
a.Writeline ("     ""face.71"" ""face.72"" ""face.73"" ""face.74""
             ""face.75"" real  ")
```

When all faces are connected, the volumes of the *wind environment* and the top cylinder are created by stitching the faces together.

```
a.Writeline ("   volume create stitch ""face.1"" ""face.10""
             ""face.28"" ""face.39"" ""face.40"" \   ")
a.Writeline ("     ""face.44"" real  ")
a.Writeline ("   volume create stitch ""face.4"" ""face.13""
             ""face.38"" ""face.40"" ""face.41"" \   ")
a.Writeline ("     ""face.48"" real  ")
a.Writeline ("   volume create stitch ""face.7"" ""face.16""
             ""face.36"" ""face.37"" ""face.41"" \   ")
a.Writeline ("     ""face.46"" real  ")
a.Writeline ("   volume create stitch ""face.2"" ""face.11""
             ""face.29"" ""face.44"" ""face.45"" \   ")
a.Writeline ("     ""face.51"" real  ")
a.Writeline ("   volume create stitch ""face.5"" ""face.48""
             ""face.49"" ""face.50"" ""face.51"" \   ")
a.Writeline ("     ""face.14"" real  ")
a.Writeline ("   volume create stitch ""face.8"" ""face.17""
             ""face.35"" ""face.46"" ""face.47"" \   ")
a.Writeline ("     ""face.49"" real  ")
a.Writeline ("   volume create stitch ""face.3"" ""face.12""
             ""face.30"" ""face.31"" ""face.42"" \   ")
a.Writeline ("     ""face.45"" real  ")
a.Writeline ("   volume create stitch ""face.6"" ""face.15""
             ""face.32"" ""face.42"" ""face.43"" \   ")
a.Writeline ("     ""face.50"" real  ")
a.Writeline ("   volume create stitch ""face.9"" ""face.18""
             ""face.33"" ""face.34"" ""face.43"" \   ")
a.Writeline ("     ""face.47"" real  ")
a.Writeline ("   volume create stitch ""face.10"" ""face.19""
             ""face.52"" ""face.55"" ""face.64"" \  ")
a.Writeline ("     ""face.68"" real  ")
a.Writeline ("   volume create stitch ""face.13"" ""face.22""
             ""face.53"" ""face.65"" ""face.64"" \  ")
```

```
a.Writeline ("     ""face.72"" real  ")
a.Writeline ("   volume create stitch ""face.16"" ""face.25""
             ""face.54"" ""face.61"" ""face.65"" \  ")
a.Writeline ("     ""face.70"" real  ")
a.Writeline ("   volume create stitch ""face.11"" ""face.20""
             ""face.56"" ""face.68"" ""face.69"" \  ")
a.Writeline ("     ""face.75"" real  ")
a.Writeline ("   volume create stitch ""face.14"" ""face.23""
             ""face.72"" ""face.73"" ""face.74"" \  ")
a.Writeline ("     ""face.75"" real  ")
a.Writeline ("   volume create stitch ""face.17"" ""face.26""
             ""face.62"" ""face.70"" ""face.71"" \  ")
a.Writeline ("     ""face.73"" real  ")
a.Writeline ("   volume create stitch ""face.12"" ""face.21""
             ""face.57"" ""face.58"" ""face.66"" \  ")
a.Writeline ("     ""face.69"" real  ")
a.Writeline ("   volume create stitch ""face.15"" ""face.24""
             ""face.59"" ""face.66"" ""face.67"" \  ")
a.Writeline ("     ""face.74"" real  ")
a.Writeline ("   volume create stitch ""face.18"" ""face.27""
             ""face.60"" ""face.63"" ""face.71"" \  ")
a.Writeline ("     ""face.67"" real  ")
```

This second phase of the script has created the frame of the computational domain. The result is shown in the following picture.



The next step is to mesh the domain. First the circles are meshed with grading 1.0 and interval count 36.

```
a.Writeline ("   edge picklink ""edge.21""   ")
a.Writeline ("   edge mesh ""edge.21"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge picklink ""edge.22""   ")
a.Writeline ("   edge mesh ""edge.22"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge picklink ""edge.23""   ")
a.Writeline ("   edge mesh ""edge.23"" successive ratio1 1 intervals
             36  ")
```

```
a.Writeline ("   edge picklink ""edge.24""   ")
a.Writeline ("   edge mesh ""edge.24"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.31"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.31""   ")
a.Writeline ("   edge mesh ""edge.31"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.39"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.39""   ")
a.Writeline ("   edge mesh ""edge.39"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.42"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.42""   ")
a.Writeline ("   edge mesh ""edge.42"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.43"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.43""   ")
a.Writeline ("   edge mesh ""edge.43"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.67"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.67""   ")
a.Writeline ("   edge mesh ""edge.67"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.75"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.75""   ")
a.Writeline ("   edge mesh ""edge.75"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.78"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.78""   ")
a.Writeline ("   edge mesh ""edge.78"" successive ratio1 1 intervals
             36  ")
a.Writeline ("   edge delete ""edge.79"" keepsettings onlymesh   ")
a.Writeline ("   edge picklink ""edge.79""   ")
a.Writeline ("   edge mesh ""edge.79"" successive ratio1 1 intervals
             36  ")
```

Then the short horizontal edges, except the 12 short edges in the middle of the outside edges, are meshed with first length (13/60)*h and interval count 16.

```
a.Writeline ("   edge modify ""edge.64"" ""edge.28"" ""edge.7""
             ""edge.13"" ""edge.26"" ""edge.30"" \    ")
a.Writeline ("      ""edge.14"" ""edge.62"" ""edge.66"" ""edge.70""
             ""edge.34"" ""edge.4"" backward   ")
a.Writeline ("   edge picklink ""edge.64"" ""edge.28"" ""edge.7""
             ""edge.13"" ""edge.26"" ""edge.30"" \  ")
a.Writeline ("      ""edge.14"" ""edge.62"" ""edge.66"" ""edge.70""
             ""edge.34"" ""edge.4""    ")
a.Writeline ("   edge mesh ""edge.64"" ""edge.28"" ""edge.7""
             ""edge.13"" ""edge.26"" ""edge.30"" \  ")
a.Writeline ("      ""edge.14"" ""edge.62"" ""edge.66"" ""edge.70""
             ""edge.34"" ""edge.4"" firstlength \  ")
a.Writeline ("     ratio1 " & (13 / 60) * h & " intervals 16")
a.Writeline ("   edge picklink ""edge.93"" ""edge.57"" ""edge.6""
             ""edge.16"" ""edge.54"" ""edge.90"" \  ")
a.Writeline ("      ""edge.87"" ""edge.51"" ""edge.15"" ""edge.9""
             ""edge.49"" ""edge.85""    ")
a.Writeline ("   edge mesh ""edge.85"" ""edge.49"" ""edge.9""
             ""edge.15"" ""edge.51"" ""edge.87"" \  ")
a.Writeline ("      ""edge.90"" ""edge.54"" ""edge.16"" ""edge.6""
             ""edge.57"" ""edge.93"" firstlength \  ")
```

```
a.Writeline ("     ratio1 " & (13 / 60) * h & " intervals 16")
```

The long horizontal edges are meshed with first length (13/60)*h and interval count 23.

```
a.Writeline ("   edge modify ""edge.1"" ""edge.25"" ""edge.61""
             backward ")
a.Writeline ("   edge picklink ""edge.1"" ""edge.25"" ""edge.61""  ")
a.Writeline ("   edge modify ""edge.17"" ""edge.27"" ""edge.63""
             backward    ")
a.Writeline ("   edge mesh ""edge.1"" ""edge.25"" ""edge.61""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge modify ""edge.19"" ""edge.38"" ""edge.74""
             backward    ")
a.Writeline ("   edge picklink ""edge.19"" ""edge.38"" ""edge.74"" ")
a.Writeline ("   edge mesh ""edge.19"" ""edge.38"" ""edge.74""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge modify ""edge.10"" ""edge.50"" ""edge.86""
             backward    ")
a.Writeline ("   edge picklink ""edge.10"" ""edge.50"" ""edge.86"" ")
a.Writeline ("   edge mesh ""edge.10"" ""edge.50"" ""edge.86""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge picklink ""edge.69"" ""edge.33"" ""edge.3""  ")
a.Writeline ("   edge mesh ""edge.3"" ""edge.33"" ""edge.69""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge picklink ""edge.71"" ""edge.35"" ""edge.18"" ")
a.Writeline ("   edge mesh ""edge.18"" ""edge.35"" ""edge.71""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge picklink ""edge.63"" ""edge.27"" ""edge.17"" ")
a.Writeline ("   edge mesh ""edge.17"" ""edge.27"" ""edge.63""
             firstlength ratio1 " & (13 / 60) * h & " intervals 23")
a.Writeline ("   edge picklink ""edge.94"" ""edge.58"" ""edge.12""
             ""edge.20"" ""edge.46"" ""edge.82""   ")
a.Writeline ("   edge mesh ""edge.82"" ""edge.46"" ""edge.20""
             ""edge.12"" ""edge.58"" ""edge.94"" \ ")
a.Writeline ("     firstlength ratio1 " & (13 / 60) * h & " intervals
             23")
```

The short vertical edges are meshed with interval size 0.1h and grading 1.0

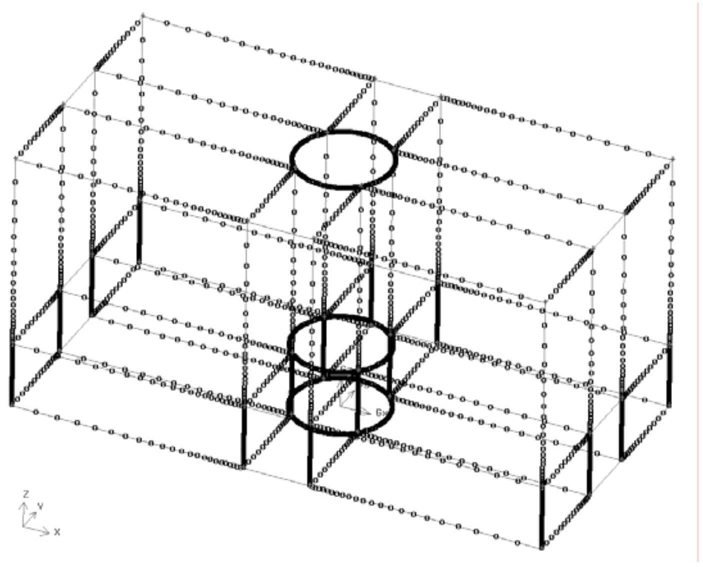
```
a.Writeline ("   edge picklink ""edge.95""   ")
a.Writeline ("   edge mesh ""edge.95"" successive ratio1 1 size " &
             (6 / 60) * h)
a.Writeline ("   edge picklink ""edge.110"" ""edge.109"" ""edge.108""
""edge.107"" ""edge.106"" \    ")
a.Writeline ("     ""edge.102"" ""edge.101"" ""edge.105""
""edge.104"" ""edge.100"" ""edge.103"" ""edge.99"" \   ")
a.Writeline ("     ""edge.98"" ""edge.97"" ""edge.96""   ")
a.Writeline ("   edge mesh ""edge.96"" ""edge.97"" ""edge.98""
""edge.99"" ""edge.103"" ""edge.100"" \   ")
a.Writeline ("     ""edge.104"" ""edge.105"" ""edge.101""
""edge.102"" ""edge.106"" ""edge.107"" \   ")
a.Writeline ("     ""edge.108"" ""edge.109"" ""edge.110"" successive
ratio1 1 size " & 6 * h / 60)
```

The long vertical edges are meshed with first length 0.1h and interval count 24

```
a.Writeline ("   edge picklink ""edge.111"" ""edge.114"" ""edge.119""
             ""edge.123"" ""edge.124"" \    ")
a.Writeline ("     ""edge.125"" ""edge.120"" ""edge.121""
             ""edge.113"" ""edge.116"" ""edge.112"" \   ")
```

```
a.Writeline ("     ""edge.115"" ""edge.117"" ""edge.118""
             ""edge.122"" ""edge.126""  ")
a.Writeline ("   edge mesh ""edge.126"" ""edge.122"" ""edge.118""
             ""edge.117"" ""edge.115"" ""edge.112"" \   ")
a.Writeline ("     ""edge.116"" ""edge.113"" ""edge.121""
             ""edge.120"" ""edge.125"" ""edge.124"" \   ")
a.Writeline ("     ""edge.123"" ""edge.119"" ""edge.114""
             ""edge.111"" firstlength ratio1 " & 6 * h / 60 & "
             intervals \  ")
a.Writeline ("   24")
```

All edges of the computational domain are now meshed. The result is shown below.



The next step is to mesh the volumes of the domain. Apart from the cylinders, the volumes of the computational domain are meshed with hexahedral elements with the map scheme.

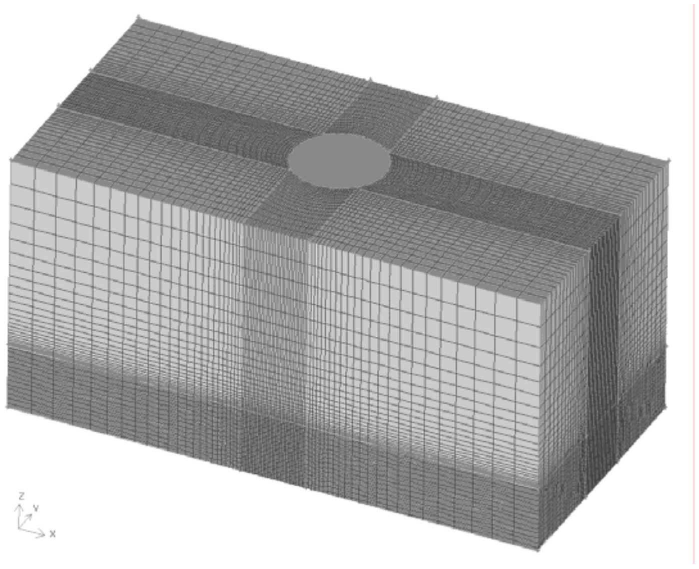
```
a.Writeline ("   volume mesh ""volume.1"" ""volume.2"" ""volume.3""
             ""volume.4"" ""volume.6"" ""volume.7"" \    ")
a.Writeline ("     ""volume.8"" ""volume.9"" ""volume.10""
             ""volume.11"" ""volume.12"" ""volume.13"" \    ")
a.Writeline ("      ""volume.15"" ""volume.16"" ""volume.17""
             ""volume.18"" map size 1")
```

Finally the boundary zones are assigned to the boundaries of the domain.

```
a.Writeline ("   physics create ""Inlet"" btype ""VELOCITY_INLET""
face ""face.54"" ""face.53"" \    ")
a.Writeline ("     ""face.52"" ""face.38"" ""face.39"" ""face.37"" ")
a.Writeline ("   physics create ""Outlet"" btype ""PRESSURE_OUTLET""
             face ""face.58"" ""face.31"" \  ")
a.Writeline ("     ""face.59"" ""face.32"" ""face.60"" ""face.33"" ")
a.Writeline ("   physics create ""Right"" btype ""SYMMETRY"" face
             ""face.28"" ""face.29"" ""face.30"" \  ")
a.Writeline ("     ""face.56"" ""face.57"" ""face.55""   ")
a.Writeline ("   physics create ""Left"" btype ""SYMMETRY"" face
             ""face.61"" ""face.62"" \   ")
a.Writeline ("     ""face.63"" ""face.36"" ""face.35"" ""face.34"" ")
a.Writeline ("   physics create ""Ground_far"" btype ""WALL"" face
             ""face.1"" ""face.4"" ""face.7"" \    ")
```

```
a.Writeline ("     ""face.8"" ""face.2"" ""face.3"" ""face.6""
             ""face.9""    ")
a.Writeline ("   physics create ""Sky"" btype ""VELOCITY_INLET"" face
             ""face.25"" ""face.26"" \  ")
a.Writeline ("     ""face.27"" ""face.24"" ""face.22"" ""face.19""
             ""face.21"" ""face.20"" ""face.23""   ")
a.Writeline ("   window modify volume invisible mesh ")

a.Close

End Sub
```

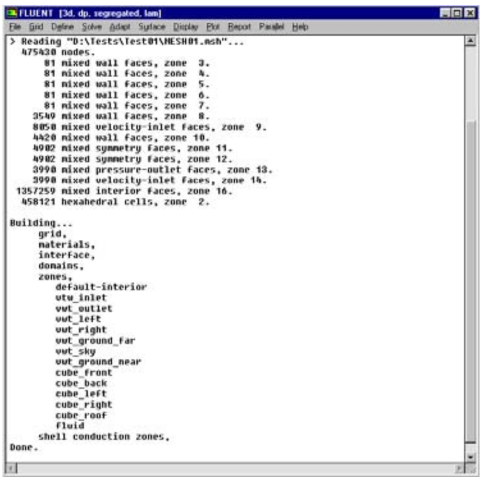
The result of the journal file is shown in the following figure. The environment is meshed; the two cylinders in the middle are not. In the bottom cylinder the research area will be placed, after which the two cylinders can be meshed.



Appendix M: Steady calculation procedure in Fluent

In this appendix the procedure to setup a steady calculation in Fluent is given. Text and pictures are derived from Van Nalta [16] and where necessary slightly adapted. At the end of the appendix the script of the user-defined function is given that adds the turbulence quantities and the logarithmic velocity profile to the calculation.

1. Read the mesh file from Gambit (File → Read → Case…).

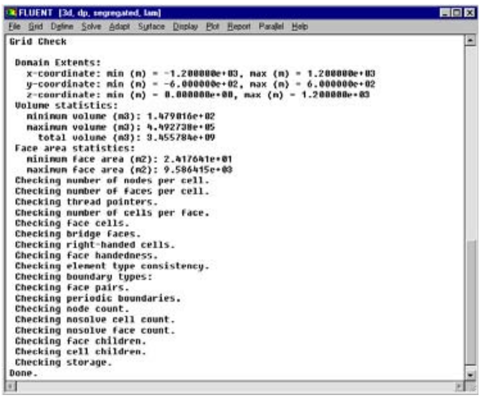

FLUENT [3d, dp, segregated, lam]
File Grid Define Solve Adapt Surface Display Plot Report Parallel Help

```
> Reading "D:\Tests\Test01\MESH01.msh"...
  475430 nodes.
      81 mixed wall faces, zone  3.
      81 mixed wall faces, zone  4.
      81 mixed wall faces, zone  5.
      81 mixed wall faces, zone  6.
      81 mixed wall faces, zone  7.
    3549 mixed wall faces, zone  8.
    8050 mixed velocity-inlet faces, zone  9.
    4420 mixed wall faces, zone 10.
    4902 mixed symmetry faces, zone 11.
    4902 mixed symmetry faces, zone 12.
    3990 mixed pressure-outlet faces, zone 13.
    3990 mixed velocity-inlet faces, zone 14.
 1357259 mixed interior faces, zone 16.
  458121 hexahedral cells, zone  2.

Building...
     grid,
     materials,
     interface,
     domains,
     zones,
        default-interior
        vtw_inlet
        vwt_outlet
        vwt_left
        vwt_right
        vwt_ground_far
        vwt_sky
        vwt_ground_near
        cube_front
        cube_back
        cube_left
        cube_right
        cube_roof
        fluid
     shell conduction zones,
Done.
```

Step 1. Reading the mesh file.

2. Check the grid (Grid → Check).

   Fluent automatically performs a number of checks in connection with the integrity of the grid. The user should check whether there are no negative volumes (minimum volume > 0) and faces (minimum face area > 0).

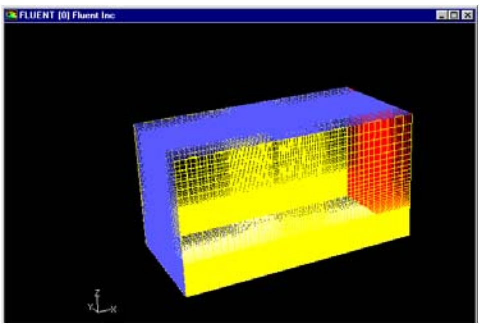

FLUENT [3d, dp, segregated, lam]
File Grid Define Solve Adapt Surface Display Plot Report Parallel Help

```
Grid Check

 Domain Extents:
   x-coordinate: min (m) = -1.200000e+03, max (m) = 1.200000e+03
   y-coordinate: min (m) = -6.000000e+02, max (m) = 6.000000e+02
   z-coordinate: min (m) = 0.000000e+00, max (m) = 1.200000e+03
 Volume statistics:
   minimum volume (m3): 1.479016e+02
   maximum volume (m3): 4.492730e+05
     total volume (m3): 3.455784e+09
 Face area statistics:
   minimum face area (m2): 2.417641e+01
   maximum face area (m2): 9.586415e+03
 Checking number of nodes per cell.
 Checking number of faces per cell.
 Checking thread pointers.
 Checking number of cells per face.
 Checking face cells.
 Checking bridge faces.
 Checking right-handed cells.
 Checking face handedness.
 Checking element type consistency.
 Checking boundary types:
 Checking face pairs.
 Checking periodic boundaries.
 Checking node count.
 Checking nosolve cell count.
 Checking nosolve face count.
 Checking face children.
 Checking cell children.
 Checking storage.
Done.
```

Step 2. Checking the gird.

3. Display the grid and visually inspect the grid (Display → Grid…).

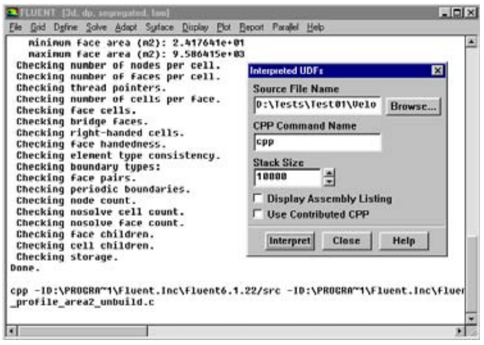
   Check whether all boundaries are present and of the appropriate type. Inspect the grid for gaps and strangely distorted cells.



Step 3. Visually checking the gird.

4. Interpret the User-defined function (Define → User-Defined → Functions → Interpreted…).

   Place the UDF preferably in the same folder as the mesh file.


FLUENT [3d, dp, segregated, lam]
File Grid Define Solve Adapt Surface Display Plot Report Parallel Help

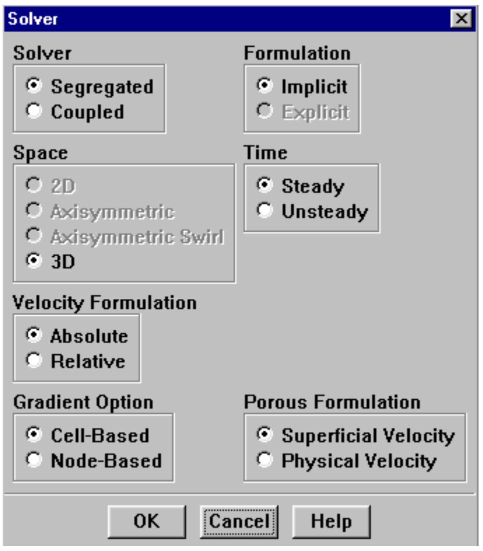
```
   minimum face area (m2): 2.417641e+01
   maximum face area (m2): 9.586415e+03
 Checking number of nodes per cell.
 Checking number of faces per cell.
 Checking thread pointers.
 Checking number of cells per face.
 Checking face cells.
 Checking bridge faces.
 Checking right-handed cells.
 Checking face handedness.
 Checking element type consistency.
 Checking boundary types:
 Checking face pairs.
 Checking periodic boundaries.
 Checking node count.
 Checking nosolve cell count.
 Checking nosolve face count.
 Checking face children.
 Checking cell children.
 Checking storage.
Done.

cpp -ID:\PROGRA~1\Fluent.Inc\fluent6.1.22/src -ID:\PROGRA~1\Fluent.Inc\fluer
_profile_area2_unbuild.c
```

Interpreted UDFs
Source File Name: D:\Tests\Test01\Velo
CPP Command Name: cpp
Stack Size: 10000
Display Assembly Listing [ ]
Use Contributed CPP [ ]

Step 4. The interpreted UDF panel.

5. Choose the solver (Define → Models → Solver…).

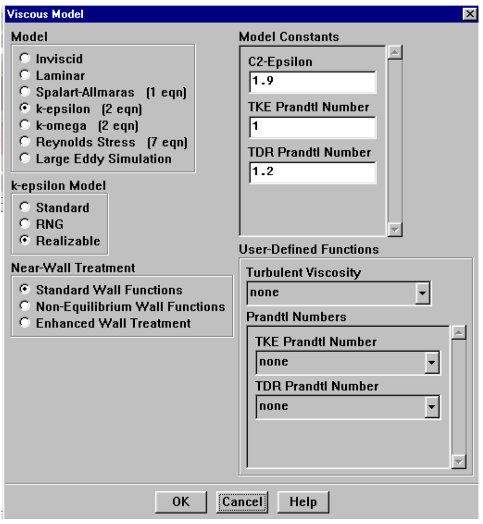
   Segregated, Implicit, 3D, Steady, Absolute, Cell-Based, Superficial Velocity. These are the default values.


Solver
Solver: [✓ Segregated] Coupled
Formulation: [✓ Implicit] Explicit
Space: 2D Axisymmetric Axisymmetric Swirl [✓ 3D]
Time: [✓ Steady] Unsteady
Velocity Formulation: [✓ Absolute] Relative
Gradient Option: [✓ Cell-Based] Node-Based
Porous Formulation: [✓ Superficial Velocity] Physical Velocity

Step 5. Setting the solver.

6. Define the turbulence model (Define → Models → Viscous…)

   k-epsilon (2 eqn), Realizable, Standard Wall Functions, Default Model Constants (1.9, 1, 1.2), User-Defined Functions: all none.

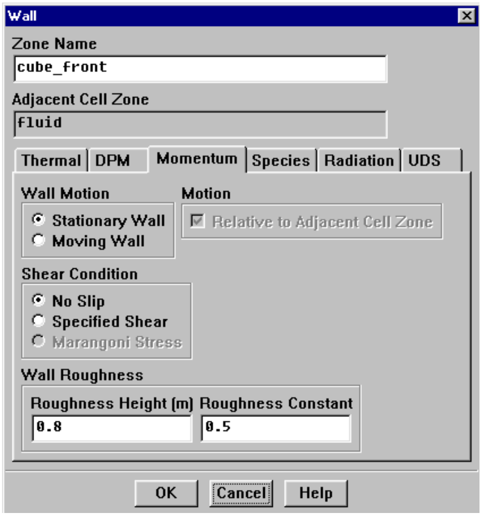

Viscous Model
Model: Inviscid Laminar Spalart-Allmaras (1 eqn) [✓ k-epsilon (2 eqn)] k-omega (2 eqn) Reynolds Stress (7 eqn) Large Eddy Simulation
k-epsilon Model: Standard RNG [✓ Realizable]
Near-Wall Treatment: [✓ Standard Wall Functions] Non-Equilibrium Wall Functions Enhanced Wall Treatment
Model Constants
C2-Epsilon: 1.9
TKE Prandtl Number: 1
TDR Prandtl Number: 1.2
User-Defined Functions
Turbulent Viscosity: none
Prandtl Numbers
TKE Prandtl Number: none
TDR Prandtl Number: none

Step 6. Defining the turbulence model.

7. Define the boundary conditions (Define → Boundary Conditions…)

   a. building: (Set… → Momentum) Stationary wall, No slip, 0.8, 0.5.

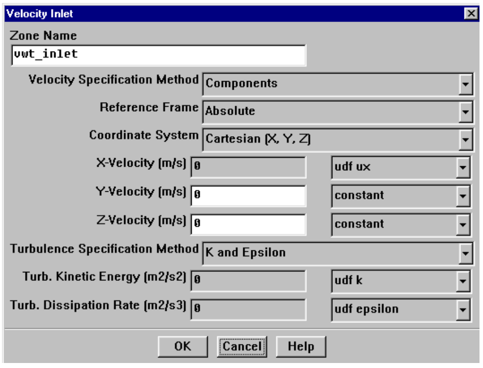
      For each building the roughness height and the roughness constant have to be defined. The roughness height depends on the friction factor of the façades.



Zone Name: cube_front
Adjacent Cell Zone: fluid
Wall Motion: [✓ Stationary Wall] Moving Wall
Motion: Relative to Adjacent Cell Zone
Shear Condition: [✓ No Slip] Specified Shear, Marangoni Stress
Wall Roughness: Roughness Height (m) 0.8, Roughness Constant 0.5

Step 7a. Defining the boundary condition at the front of the cube.

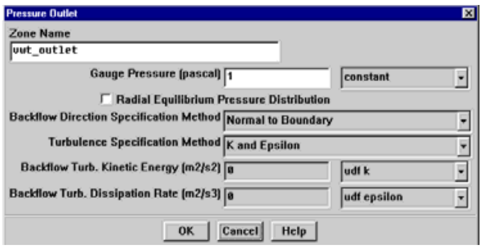
   b. inlet: (Set…) Components, Absolute, Cartesian (X, Y, Z), udf ux, constant 0, constant 0, K and Epsilon, udf k, udf epsilon.



Zone Name: vwt_inlet
Velocity Specification Method: Components
Reference Frame: Absolute
Coordinate System: Cartesian (X, Y, Z)
X-Velocity (m/s): 0, udf ux
Y-Velocity (m/s): 0, constant
Z-Velocity (m/s): 0, constant
Turbulence Specification Method: K and Epsilon
Turb. Kinetic Energy (m2/s2): 0, udf k
Turb. Dissipation Rate (m2/s3): 0, udf epsilon

Step 7b. Defining the boundary condition at the inlet of the virtual wind tunnel.

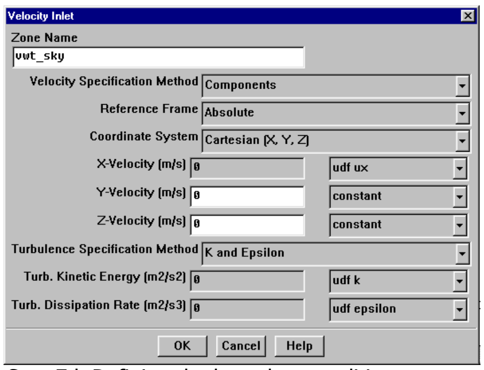
   c. outlet: (Set…) constant 1, Normal to Boundary, K and Epsilon, udf k, udf epsilon.



Zone Name: vwt_outlet
Gauge Pressure (pascal): 1, constant
Radial Equilibrium Pressure Distribution
Backflow Direction Specification Method: Normal to Boundary
Turbulence Specification Method: K and Epsilon
Backflow Turb. Kinetic Energy (m2/s2): 0, udf k
Backflow Turb. Dissipation Rate (m2/s3): 0, udf epsilon

Step 7c. Defining the boundary condition at the outlet of the virtual wind tunnel.

   d. sky: (Set…) Components, Absolute, Cartesian (X, Y, Z), udf ux, constant 0, constant 0, K and Epsilon, udf k, udf epsilon.

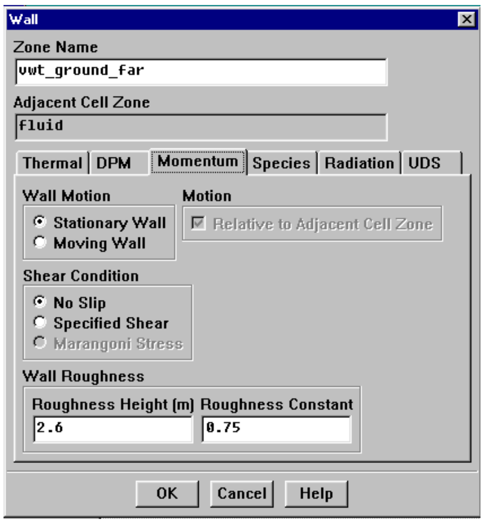


Zone Name: vwt_sky
Velocity Specification Method: Components
Reference Frame: Absolute
Coordinate System: Cartesian (X, Y, Z)
X-Velocity (m/s): 0, udf ux
Y-Velocity (m/s): 0, constant
Z-Velocity (m/s): 0, constant
Turbulence Specification Method: K and Epsilon
Turb. Kinetic Energy (m2/s2): 0, udf k
Turb. Dissipation Rate (m2/s3): 0, udf epsilon

Step 7d. Defining the boundary condition at the top of the virtual wind tunnel.

e. ground: (Set… → Momentum) Stationary wall, No slip, 2.6, 0.75.

For unbuilt areas the roughness length is 0.2, what gives a roughness height of 2.6. If a built area is placed into the domain, the roughness length and roughness height have to be defined again.


Wall

Zone Name: vwt_ground_far

Adjacent Cell Zone: fluid

Thermal | DPM | Momentum | Species | Radiation | UDS

Wall Motion: [✓ Stationary Wall] Moving Wall

Motion: [✓ Relative to Adjacent Cell Zone]

Shear Condition: [✓ No Slip] Specified Shear Marangoni Stress

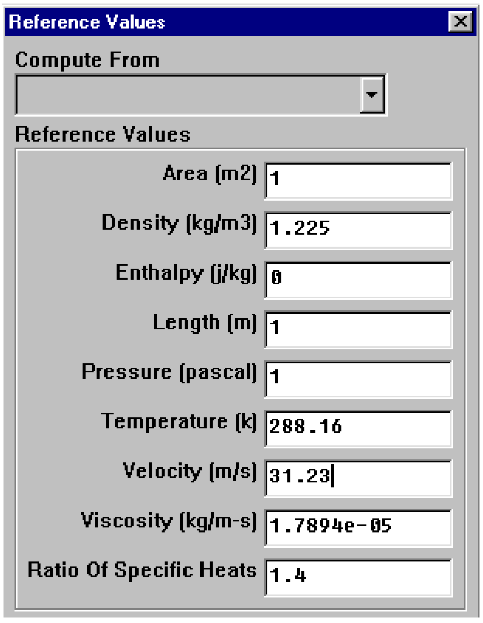
Wall Roughness

Roughness Height (m)	Roughness Constant
2.6	0.75

Step 7e. Defining the boundary condition at the ground of the virtual wind tunnel.

8. Set reference values (Report → Reference Values…)

Change the Pressure (pascal) to 1 and the Velocity (m/s) to 31.23. Keep the default values for the other parameters.


Reference Values

Compute From

Reference Values

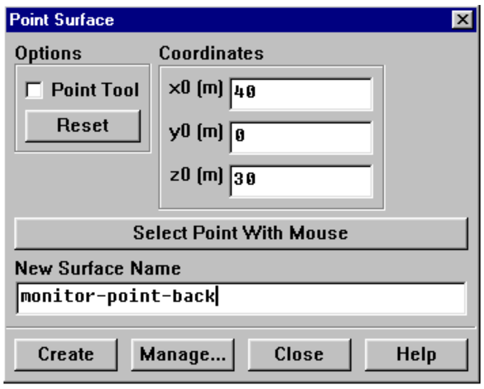
Area (m2)	1
Density (kg/m3)	1.225
Enthalpy (j/kg)	0
Length (m)	1
Pressure (pascal)	1
Temperature (k)	288.16
Velocity (m/s)	31.23
Viscosity (kg/m-s)	1.7894e-05
Ratio Of Specific Heats	1.4

Step 8. The Reference Values that have been used for the calculations.

9. Create points (Surface → Point…)

These are the points that are used to monitor convergence. Some stratetic points have to be chosen to monitor the convergence:

- point just above the building
- point just behind the building


Point Surface

Options: Point Tool (unchecked)

Coordinates

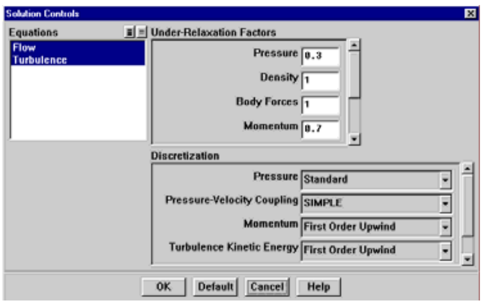
x0 (m)	40
y0 (m)	0
z0 (m)	30

New Surface Name: monitor-point-back

Step 9. Creating a point above the roof of the cube, to be used to monitor convergence.

10. Choose the discretization schemes (Solve → Controls → Solution…).

    Default Under-Relaxation Factors (0.3, 1, 1, 0.7, 0.8, 0.8, 1), Default discretization schemes (Standard, SIMPLE, First Order Upwind, First Order Upwind, First Order Upwind).


Solution Controls

Equations: Flow, Turbulence

Under-Relaxation Factors

Pressure	0.3
Density	1
Body Forces	1
Momentum	0.7

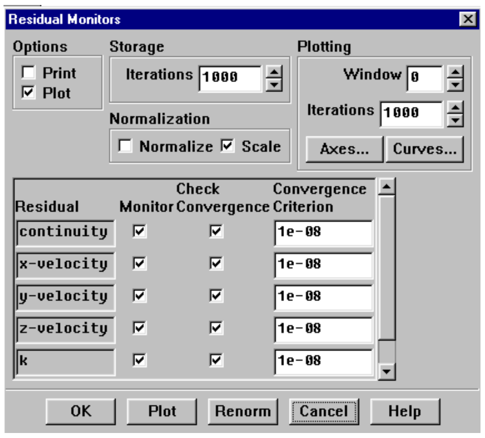
Discretization

Pressure	Standard
Pressure-Velocity Coupling	SIMPLE
Momentum	First Order Upwind
Turbulence Kinetic Energy	First Order Upwind

Step 10. The default settings for the solution control panel.

11. Set the convergence monitors (Solve → Monitors →)

    a. residuals (→ Residual…)
       plot all residuals, scaled, with convergence criterion 10e-9.


Residual Monitors

Options: ☐ Print, ☑ Plot

Storage: Iterations 1000

Normalization: ☐ Normalize ☑ Scale

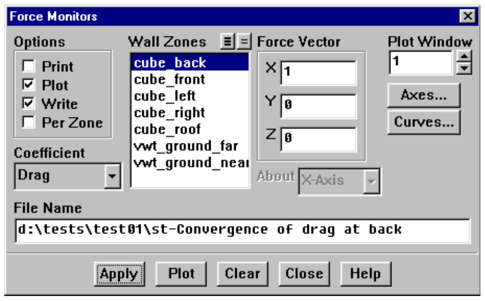
Plotting: Window 0, Iterations 1000

Residual	Monitor	Check Convergence	Convergence Criterion
continuity	☑	☑	1e-08
x-velocity	☑	☑	1e-08
y-velocity	☑	☑	1e-08
z-velocity	☑	☑	1e-08
k	☑	☑	1e-08

Step 11a. Settings for monitoring the convergence of the residuals. Convergence is checked and the calculation is stopped when the desired criterion is reached.

    b. drag at back (→ Force…)
       Plot, Write, Drag coefficient, cube back, Force Vector (1,0,0), File Name, Apply.

       An extra monitor is added to monitor the convergence of drag at the back of the building


Force Monitors

Options: ☐ Print, ☑ Plot, ☑ Write, ☐ Per Zone

Wall Zones: cube_back (selected), cube_front, cube_left, cube_right, cube_roof, vwt_ground_far, vwt_ground_near

Force Vector: X 1, Y 0, Z 0

About: X-Axis

Plot Window: 1

Coefficient: Drag

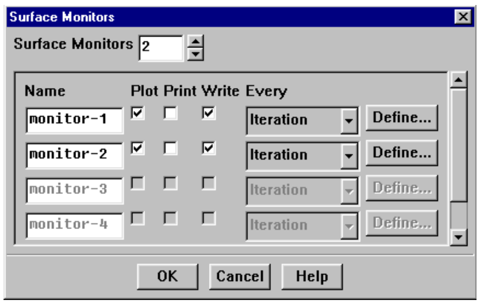
File Name: d:\tests\test01\st-Convergence of drag at back

Step 11b. Settings for monitoring the convergence of the drag at the back of the cube.

c. velocity magnitude above roof and at the back (→ Surface…)

2 Surface Monitors, Plot, Write, Every Iteration,

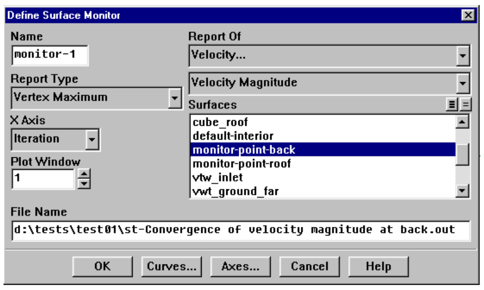
monitor-1: (Define…) Velocity Magnitude, Vertex Maximum, Iteration on X Axis, monitor-point-back, File Name.


Surface Monitors

Surface Monitors 2

Name	Plot	Print	Write	Every	
monitor-1	✓		✓	Iteration	Define...
monitor-2	✓		✓	Iteration	Define...
monitor-3				Iteration	Define...
monitor-4				Iteration	Define...

monitor-2: (Define…) Velocity Magnitude, Vertex Maximum, Iteration on X Axis, monitor-point-roof, File Name.


Define Surface Monitor

Name: monitor-1

Report Of: Velocity...

Velocity Magnitude

Report Type: Vertex Maximum

Surfaces:
cube_roof
default-interior
monitor-point-back
monitor-point-roof
vtw_inlet
vwt_ground_far

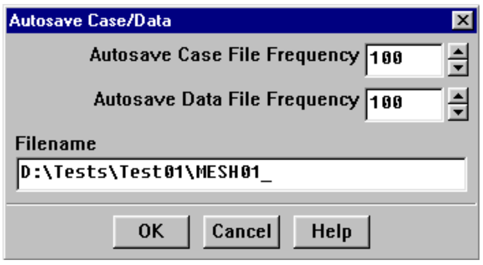
X Axis: Iteration

Plot Window: 1

File Name: d:\tests\test01\st-Convergence of velocity magnitude at back.out

Step 11b. Monitoring the convergence of the velocity magnitude at two characteristic points.

12. Set the autosave option (File → Write → Autosave…)


Autosave Case/Data

Autosave Case File Frequency 100

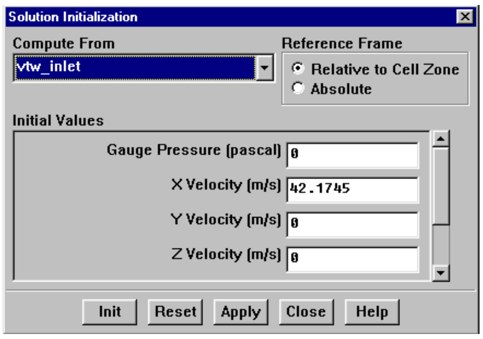
Autosave Data File Frequency 100

Filename: D:\Tests\Test01\MESH01_

Step 12. Enable the autosave option, to automatically save the case and data files at a certain frequency.

13. Initialise the solution (Solve → Initialize → Initialize…)

Compute From inlet, Init.


Solution Initialization

Compute From: vtw_inlet

Reference Frame: [✓ Relative to Cell Zone] Absolute

Initial Values

Gauge Pressure (pascal) 0

X Velocity (m/s) 42.1745

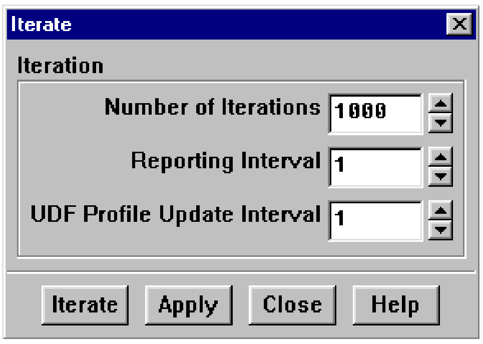
Y Velocity (m/s) 0

Z Velocity (m/s) 0

Step 13. Initializing the solution.

14. Iterate (Solve → Iterate…)

    1000 iterations, reporting interval 1, udf update interval 1.


Iterate

Iteration

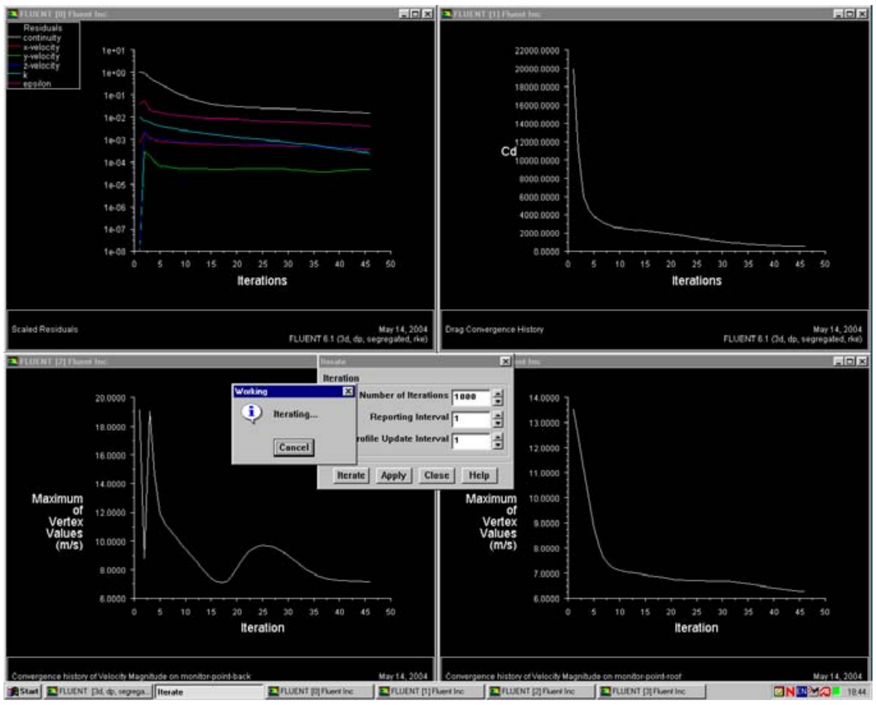
Number of Iterations 1000

Reporting Interval 1

UDF Profile Update Interval 1

Iterate Apply Close Help

Step 14. Starting the iteration procedure.


Residuals
continuity
x-velocity
y-velocity
z-velocity
k
epsilon
Iterations
Scaled Residuals
FLUENT 6.1 (3d, dp, segregated, rke)
May 14, 2004
Cd
Iterations
Drag Convergence History
Working
Iterating...
Cancel
Maximum of Vertex Values (m/s)
Iteration
Convergence history of Velocity Magnitude on monitor-point-back
Maximum of Vertex Values (m/s)
Iteration
Convergence history of Velocity Magnitude on monitor-point-roof
Start

Iteration

Number of Iterations 1000

Reporting Interval 1

Profile Update Interval 1

Iterate Apply Close Help

The screen during the iteration procedure.

The user-defined function that adds the turbulence quantities and the logarithmic velocity profile to the calculation is given by:

```
#include "udf.h"
#define z0 0.2
#define d 0
#define kappa 0.42
#define uster 2.3
#define cmu 0.09
#define visc 1.7894e-05
#define rho 1.225

DEFINE_PROFILE(ux,threadu,nvu)
{
        face_t f;
        real a[ND_ND];
        real x;
        real y;
        real z;

        begin_f_loop (f,threadu)
        {
                F_CENTROID(a,f,threadu);
                x=a[0];
                y=a[1];
                z=a[2];
                if (z<=d)
                {
                        F_PROFILE(f,threadu,nvu)=0;
                }
                else
                        F_PROFILE(f,threadu,nvu)=(uster/kappa)*log((z-d)/z0);
        }
        end_f_loop (f,threadu)
}

DEFINE_PROFILE(k,threadk,nvk)
{
        face_t f;
        real x[ND_ND];

        begin_f_loop (f,threadk)
        {
                F_CENTROID(x,f,threadk);
                F_PROFILE(f,threadk,nvk)=pow(uster,2)/sqrt(cmu);
        }
        end_f_loop (f,threadk)
}
```

```
DEFINE_PROFILE(epsilon,threade,nve)
{
        face_t f;
        real x[ND_ND];

        begin_f_loop (f,threade)
        {
                F_CENTROID(x,f,threade);
                if (x[2]<=d)
                        F_PROFILE(f,threade,nve)=10;
                else
                        F_PROFILE(f,threade,nve)=pow(uster,3)/(kappa*(x[2]-d));
        }
        end_f_loop (f,threade)
}
```

Appendix N: Step by step guide - grid generation methods

In this appendix a step by step guide is given to mesh the lower cylinder containing the model of interest with one of the two developed grid generation methods. The first method uses hexahedral elements and can be used to mesh simple, straight geometry with the same layout from the bottom to the top of the building model. The second grid generation method uses tetrahedral elements and can be used to mesh more complex geometry. Before one of the methods is applied, the computational domain has to be generated first. Because the dimensions of the domain depend on the height of the highest building in the domain and the radius of the research area, the fourth design tool can be used to generate the required domain. The domain is constructed around the origin, with the center of the cylinder's bottom at (0, 0, 0). After the model of interest is imported in the lower cylinder, it must be placed in the middle of the cylinder, at the origin. It is recommended to move the building model to the origin already in the CAD system where it is created. When it is exported to Gambit, the model is automatically placed at the right position then. When the environment also has to be taken into account, it can be generated with the developed design tools. The tools automatically place the center of the research area at the origin. Before the research area is imported into Gambit, the circular ground surface first has to be removed manually from the model, for example in Rhinoceros. The plane was generated for visualization reasons, but the Van Nalta domain uses its own ground surface. When the ground plane is removed and the model is imported in Gambit, one of the two grid generation methods can be applied to mesh the cylinders of the domain.

N.1 Hexahedral elements

1. **<u>Subtract the building model(s) from the lower cylinder</u>**

   Geometry -> Volume -> Boolean Operations -> Subtract
   Volume = volume lower cylinder (= volume 5)
   Subtract volume(s) = volume(s) building model(s) (= from volume 19)

2. **<u>Find the shortest edge of the research area</u>**

   Geometry -> Edge -> Connect/Disconnect -> Real and Virtual -> Highlight shortest edge

3. **<u>Attach size function to ground edges building model(s) and bottom of cylinder</u>**

   Tools -> Sizing function -> Create
   Type = fixed
   Sources: bottom edges of building model(s)
   Attachment: circular ground face of lower cylinder (= face 5)
   Parameters: Start size: length of the shortest edge at maximum
   Growth rate: 1.02
   Size limit: height of the building model (this is ignored because the circle edge is already meshed)

4. **<u>Mesh ground face of bottom cylinder and top of the building model(s)</u>**

   Mesh -> Face -> Mesh faces
   Mesh the ground face of the bottom cylinder (= face.5) and the top face(s) of the building(s) with Quad elements and the Pave scheme

5. **Link mesh distribution on ground edges of building model(s) with top edges**

   Mesh -> Edge -> Link/Unlink
   First select the ground edge; then select the top edge. Repeat the procedure for all edges

6. **Mesh the vertical edges of the building model(s)**

   Mesh -> Edge -> Mesh edges
   Mesh all vertical edges of the building model(s) with an interval size equal to the start size of the size function

7. **Mesh the volume of the lower cylinder**

   Mesh -> Volume -> Mesh volumes
   Mesh the lower cylinder (= volume 5) with Hex/Wedge elements and the Cooper scheme

8. **Mesh the volume of the upper cylinder**

   Mesh -> Volume -> Mesh volumes
   Mesh the upper cylinder (= volume 14) with Hex/Wedge elements and the Cooper scheme

9. **Assign boundary zones on the building(s) faces and the nearby ground face**

   Zones -> Specify boundary types -> Add

   Building of interest:
   a. Front of building model -> "Building_front" -> wall
   b. Back of building model -> "Building_back" -> wall
   c. Left side of building model -> "Building_left" -> wall
   d. Right side of building model -> "Building_right" -> wall
   e. Top of building model -> "Building_top" -> wall

   Other buildings in the research area:
   a. Front, back, left, right and top of building model -> "Building_name " -> wall

   Bottom of the lower cylinder
   a. Circular ground face -> "Ground_near" -> wall

10. **Export the mesh**

   File -> Export -> Mesh

N.2 Tetrahedral elements – Automatic meshing

1. **Subtract the building model(s) from the lower cylinder**

   Geometry -> Volume -> Boolean Operations -> Subtract
   Volume = volume lower cylinder (= volume 5)
   Subtract volume(s) = volume(s) building model(s) (= from volume 19)

2. **Find the shortest edge of the research area**

   Geometry -> Edge -> Connect/Disconnect -> Real and Virtual -> Highlight shortest edge

3. **Mesh all edges of the building model(s)**

   Mesh -> Edge -> Mesh edges
   Mesh all edges of the building model(s) with an interval size equal to the length of the shortest edge at maximum. That edge is meshed with at least one cell then. However, if possible, it is preferred to choose a smaller interval size.

4. **Mesh the volume of the lower cylinder**

   Mesh -> Volume -> Mesh volumes
   Mesh the lower cylinder (= volume 5) with Tet/Hybrid elements and the TGrid scheme

5. **Mesh the volume of the upper cylinder**

   Mesh -> Volume -> Mesh volumes
   Mesh the upper cylinder (= volume 14) with Tet/Hybrid elements and the TGrid scheme

6. **Assign boundary zones on the building(s) faces and the nearby ground face**

   Zones -> Specify boundary types -> Add

   Building of interest:
   a. Front of building model -> "Building_front" -> wall
   b. Back of building model -> "Building_back" -> wall
   c. Left side of building model -> "Building_left" -> wall
   d. Right side of building model -> "Building_right" -> wall
   e. Top of building model -> "Building_top" -> wall

   Other buildings in the research area:
   a. Front, back, left, right and top of building model -> "Building_name " -> wall

   Bottom of the lower cylinder
   a. Circular ground face -> "Ground_near" -> wall

7. **Export the mesh**

   File -> Export -> Mesh

Appendix O: Least Squares method

During this thesis two methods were proposed to simplify the enclosing curves that are generated from the outer points of a building model. The NURBS fitting method is already developed and seems to work properly. Another proposed method is the Least Squares method. In this appendix the method is discussed extensively and examples are given to fit a straight line and a parabola through a certain dataset. To simplify the enclosing curves with the Least Squares method, a third degree polynomial at minimal is required as the enclosing curves are closed curves. However, the method is not implemented yet for simplifying such curves and additional research is required to develop the method further.

Least Squares method

The Least Squares method in general is a method to determine the best-fit curve through a certain dataset. The Least Squares method is based on minimizing the squared error in vertical direction between a data point and a parametric curve. The smaller the difference between the original dataset and the proposed curve, the better the fit. Given a dataset with n points (x_1,y_1), (x_2,y_2), ..., (x_n,y_n), the polynomial that fits the dataset is given by:

$$y = f(x : c_1, c_2, ..., c_k) \qquad \text{(O.1)}$$

where the set of parameters c are the fitting parameters of the polynomial. By choosing the right values for these parameters, the error between the points and the curve can be minimized. The fitting polynomial can be expressed in the general linear form:

$$y(x) = c_1(x) + c_2(x) + c_3(x) + ... + c_k(x) \qquad \text{(O.2)}$$

In order to fit a straight line through the dataset, a first degree polynomial is required. Fitting a parabola through the dataset requires a second degree polynomial. In case of *n* data points, the errors between the curve and the data points are:

$$\begin{aligned}
\varepsilon_1 &= c_1(x_1) + c_2(x_1) + c_3(x_1) + ... + c_k(x_1) - y_1 \\
\varepsilon_2 &= c_1(x_2) + c_2(x_2) + c_3(x_2) + ... + c_k(x_2) - y_2 \\
&. \\
&. \\
\varepsilon_n &= c_1(x_n) + c_2(x_n) + c_3(x_n) + ... + c_k(x_n) - y_n
\end{aligned} \qquad \text{(O.3)}$$

The total squared error that has to be minimized for a set of n data points is:

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} [c_1(x_1) + c_2(x_2) + ... + c_k(x_i) - y_i]^2 \qquad \text{(O.4)}$$

The minimum value for S can be found by finding a set of parameters c that makes the errors at the data points as small as possible. This minimum value is derived when all partial derivatives of S are equal to zero. For a system of k parameters c it follows:

$$\begin{aligned}\frac{\partial S}{\partial c_1} &= 0\\ \frac{\partial S}{\partial c_2} &= 0\\ &.\\ \frac{\partial S}{\partial c_k} &= 0\end{aligned} \tag{O.5}$$

With the obtained values for the parameters c the fitting polynomial that minimizes the squared error for all data points can be formulated. On the following pages the least squares method is explained with two examples. The first example fits a straight line through a certain dataset; the second example fits a parabola through the dataset.

If a certain dataset contains the following eight points for example:

x	1	2	3	4	5	6	7	8
y	2	3	3	2	3	4	3	2

In order to fit a straight line through the dataset, a first degree polynomial y has to be found that minimizes S:

$$y = c_1 + c_2 x \tag{O.6}$$

$$S = \sum_{i=1}^{8} [y_i - c_1 - c_2 x_i]^2 \tag{O.7}$$

The partial derivatives of S with respect to c_1 and c_2 are:

$$\frac{\partial S}{\partial c_1} = -2\sum_{i=1}^{8} (y_i - c_1 - c_2 x_i) = 0 \tag{O.8}$$

$$\frac{\partial S}{\partial c_2} = -2\sum_{i=1}^{8} x_i (y_i - c_1 - c_2 x_i) = 0 \tag{O.9}$$

Some rearrangement results in:

$$\sum_{i=1}^{8} y_i = 8c_1 + \left(\sum_{i=1}^{8} x_i\right)\cdot c_2 \tag{O.10}$$

$$\sum_{i=1}^{8} x_i y_i = \left(\sum_{i=1}^{8} x_i\right)\cdot c_1 + \left(\sum_{i=1}^{8} x_i^2\right)\cdot c_2 \tag{O.11}$$

To obtain a solution for the parameters c_1 and c_2, the four sums Σx_i, Σy_i, $\Sigma(x_i)^2$ and $\Sigma x_i y_i$ have to be inserted into Equations (O.10) and (O.11). The following table gives the values of the different variables.

i	1	2	3	4	5	6	7	8	Σ
x_i	1	2	3	4	5	6	7	8	36
y_i	2	3	3	2	3	4	3	2	22
$(x_i)^2$	1	4	9	16	26	36	49	64	204
$x_i y_i$	2	6	9	8	15	24	21	16	101

Inserting these values into Equations (O.10) and (O.11) leads to the following equations:

$$22 = 8 \cdot c_1 + 36 \cdot c_2 \tag{O.12}$$

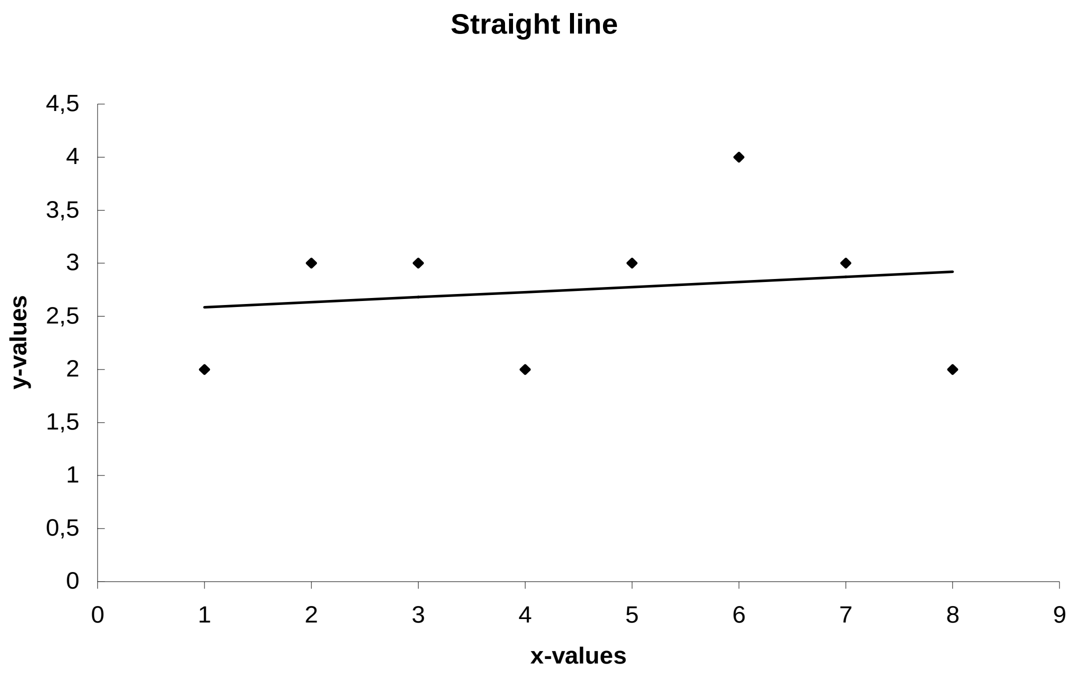
$$101 = 36 \cdot c_1 + 204 \cdot c_2 \tag{O.13}$$

From these equations the values of c_1 and c_2 can be determined: c_1 = 2,536
c_2 = 0,048

The equation for the fitting polynomial through the dataset now becomes:

$$y = c_1 + c_2 \cdot x = 2.536 + 0.048 \cdot x \tag{O.14}$$

The following figure gives the result of the straight line that is fitted through the data points.



In the next example a parabola will be fitted through the dataset. In order to fit a parabola, a second degree polynomial has to be found that minimizes S:

$$y = c_1 + c_2 x + c_3 x^2 \tag{O.15}$$

$$S = \sum_{i=1}^{8} [y_i - c_1 - c_2 x_i - c_3 x_i^2]^2 \tag{O.16}$$

The partial derivatives of S with respect to c_1, c_2 and c_3 are:

$$\frac{\partial S}{\partial c_1} = -2\sum_{i=1}^{8} (y_i - c_1 - c_2 x_i - c_3 x_i^2) = 0 \tag{O.17}$$

$$\frac{\partial S}{\partial c_2} = -2\sum_{i=1}^{8} x_i (y_i - c_1 - c_2 x_i - c_3 x_i^2) = 0 \tag{O.18}$$

$$\frac{\partial S}{\partial c_3} = -2\sum_{i=1}^{8} x_i^2 (y_i - c_1 - c_2 x_i - c_3 x_i^2) = 0 \tag{O.19}$$

Some rearrangement results in:

$$\sum_{i=1}^{8} y_i = 8c_1 + \left(\sum_{i=1}^{8} x_i\right)\cdot c_2 + \left(\sum_{i=1}^{8} x_i^2\right)\cdot c_3 \tag{O.20}$$

$$\sum_{i=1}^{8} x_i y_i = \left(\sum_{i=1}^{8} x_i\right)\cdot c_1 + \left(\sum_{i=1}^{8} x_i^2\right)\cdot c_2 + \left(\sum_{i=1}^{8} x_i^3\right)\cdot c_3 \tag{O.21}$$

$$\sum_{i=1}^{8} x_i^2 y_i = \left(\sum_{i=1}^{8} x_i^2\right)\cdot c_1 + \left(\sum_{i=1}^{8} x_i^3\right)\cdot c_2 + \left(\sum_{i=1}^{8} x_i^4\right)\cdot c_3 \tag{O.22}$$

The following table gives the values for Σx_i, Σy_i, $\Sigma(x_i)^2$, $\Sigma x_i y_i$, $\Sigma(x_i)^2 y_i$, $\Sigma(x_i)^3$ and $\Sigma(x_i)^4$.

i	1	2	3	4	5	6	7	8	Σ
x_i	1	2	3	4	5	6	7	8	36
y_i	2	3	3	2	3	4	3	2	22
$(x_i)^2$	1	4	9	16	26	36	49	64	204
$x_i y_i$	2	6	9	8	15	24	21	16	101
$(x_i)^2 y_i$	2	12	27	32	75	144	147	128	567
$(x_i)^3$	1	8	27	64	125	216	343	512	1296
$(x_i)^4$	1	16	81	256	625	1296	2401	4096	8772

To obtain a solution for the parameters c_1, c_2 and c_3, the various sums have to be inserted into Equations (O.20), (O.21) and (O.22). This leads to the following equations:

$$22 = 8 \cdot c_1 + 36 \cdot c_2 + 204 \cdot c_3 \tag{O.23}$$

$$101 = 36 \cdot c_1 + 204 \cdot c_2 + 1296 \cdot c_3 \tag{O.24}$$

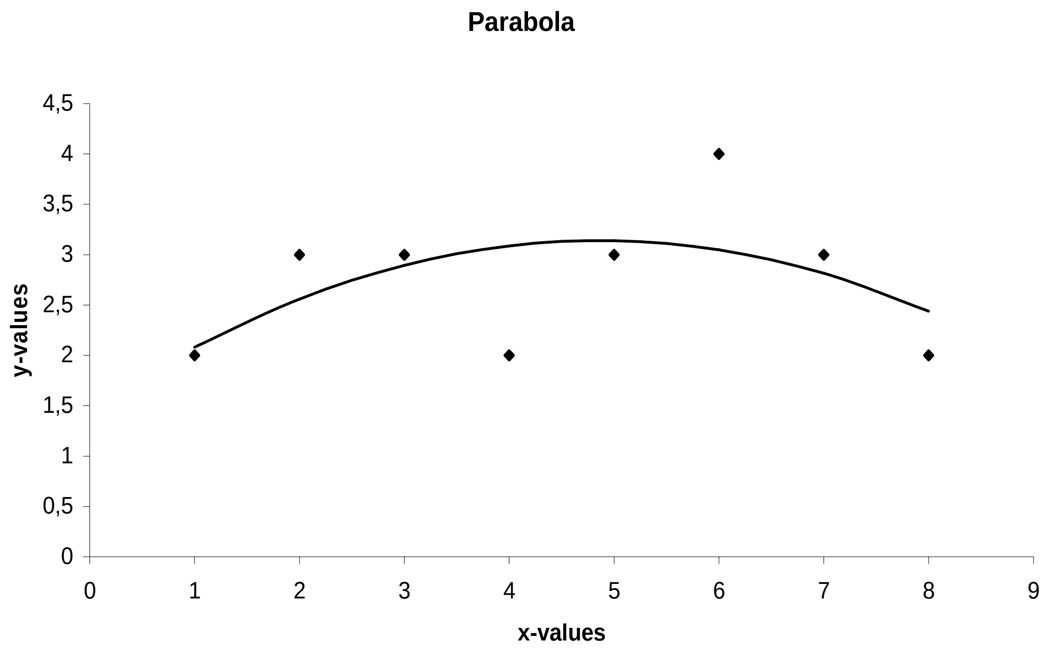
$$567 = 204 \cdot c_1 + 1296 \cdot c_2 + 8772 \cdot c_3 \tag{O.25}$$

From these equations the values of c_1, c_2 and c_3 can be determined: c_1 = 1,464
c_2 = 0,690
c_3 = -0,071

The equation for the fitting parabola through the dataset now becomes:

$$y = c_1 + c_2 \cdot x + c_3 \cdot x^2 = 1.464 + 0.690 \cdot x - 0.071 \cdot x^2 \tag{O.26}$$

The following picture gives the result of the parabola that is fitted through the data points.


Parabola
y-values
x-values