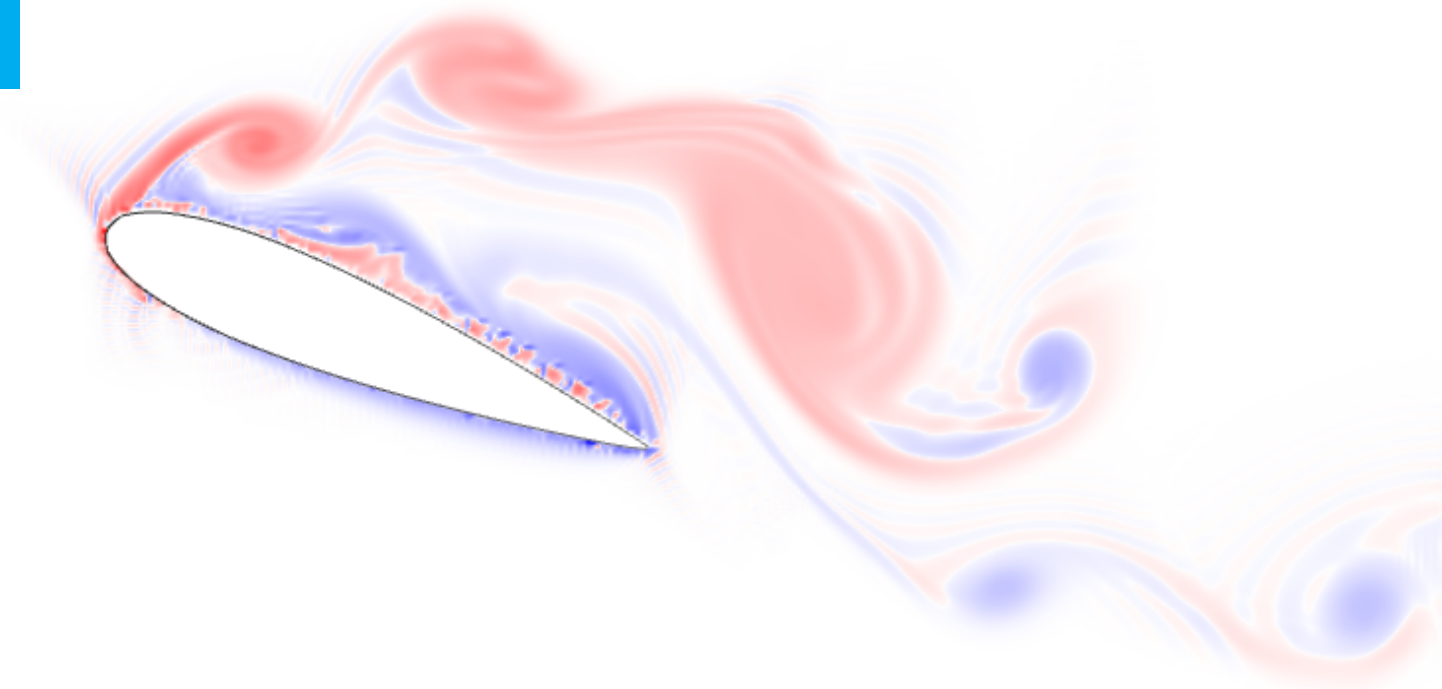
DSE 2 - Eddy

Design, development and building of an interactive wing design tool able of real-time simulation

J.G. Barnhoorn	4105257	I. van Leeuwen	4084667
S.G. Brust	4049195	K. Min	4096819
D.P. van Herwaarden	1524275	J. Ran	4086260
J. Huibers	4090594	S.F. van der Sandt	1514075
M. Kuijpers	4056434	R. Schilder	4107888

Final Report

Design Synthesis Excercise



TU Delft Delft University of Technology

Aerospace Engineering

January 28, 2014

This page was intentionally left blank.

Preface

To finalize their bachelor studies at the Faculty of Aerospace Engineering at Delft University of Technology, a group of ten students was assigned the task to develop an interactive airflow simulation tool. This report is the final status report of this group. It provides information about the entire design process, starting from the design concept up to the final prototype. The most important design choices are explained and ideas behind the program code are written down. Since the project is slightly different than standard design synthesis exercises (DSE), this report might deviate from standard DSE reports in content and lay-out. The conceptual design phase for example, was relatively short and some of the standard deliverable topics are not listed in this report, because they were not in line with the project. All the other information however, is provided.

We would like to express our gratitude to our tutors Richard Dwight and Marios Kotsonis for the opportunity to work on this assignment and for their great inspiration and guidance during the development process of the prototype tool. Their help ranged from providing a first version solver, advice on aerodynamic calculations to ideas on GUI development and computer architecture. We would also like to thank our coach Jurij Sodja, who was of great help in starting up the graphical user interface (GUI) development and whose comments on development ideas were always of great value. Additionally, we would like to thank Ir. C.W.J. Lemmens for his help, insight, and guidance in understanding parallel computing and implementing such techniques in our project. Furthermore, we would like to thank Dr. M. Holländer, Dr.ir. W.P. Breugem, L. van Midden, R. Gupta, MSc, and all the other TU Delft staff members that helped out and provided the facilities to work on this project.

Summary

When designing a wing, designers are faced with the challenge of combining creativity with aerodynamic performance and structural integrity. Although computational tools can be of great help in design processes, they are usually very time-consuming and far from intuitive. A rapid interactive flow simulation tool could be a solution that gives designers the opportunity to input an arbitrary shape in an airflow and obtain real-time feedback on the flow behavior and forces and moments that result. Such a tool could be a communication aid between designers or used for educational purposes in classrooms or science museums. Although some interactive flow simulation tools already exist, they lack accuracy and are therefore not to be relied upon when it comes to computational flow data.

As a capstone project, a group of ten aerospace engineering students is given the task of designing and developing a tool that satisfies the wishes for the tool that is aimed for. The key feature of this simulation tool is interactivity. This requires the tool to allow for user-drawn input and to simulate the airflow around this input in real-time. To enhance the interactivity a minimum of 30 frames per second was set while maintaining accuracy on a grid size of 128x128 units. This should all be realized within a budget of €2,500.

A final concept was found in an environment where the user could input an arbitrary shape in the general user interface by drawing, followed by immediate flow visualization around this shape on the drawing surface and display of requested airflow variables. Based on these functions a program architecture was developed that visualized the different components of the program, namely a graphical user interface, refinement of user input, a CFD solver, post-processing of CFD data and a graphical output part. All parts were developed in C++.

Development started with creating a working CFD solver, modeled after work produced by Richard Dwight and several research papers regarding vortex-in-cell methods. In the solver, Dirichlet boundary conditions are applied to the domain edges and no through-flow and no-slip boundary conditions are applied to the immersed boundary of the input object. In order to achieve necessary speed requirements, a custom PC was built with a large graphics processing unit that would allow for multi-threading of large matrix operations. The computation speed of the code itself was increased by making the code more efficient and using effective programming libraries such as Paralution.

Concurrently, a graphical user interface was developed that allows for functionality and drawn user input. Refine input functions were realized, such as translating, rotating the drawn input and adding multiple objects in the computational domain. Furthermore, robustness to random input was provided by an automated figure-closing function.

The post-processing part of the program was developed to compute a pressure field based on the obtained velocity field, as well as aerodynamic forces, moments and coefficients requested by the user. For the pressure solver the same computational techniques are used as for the velocity computations, but for this purpose based on the pressure Poisson equation. Forces and moments are calculated based on conventional aerodynamic theory.

A program to graphically visualize the velocity and pressure field of the airflow was written, and also to show pressure coefficient distribution plots. In addition a particle engine was built that helps visualizing streamlines.

The general user interface and graphical output are hosted by an interactive beamer that can project and recognize input provided by a pen that functions as a mouse pointer.

The work done in all the separate parts of the program resulted in a tool that is able to handle user-drawn input or preset shapes which can be reshaped and allows for the parameters V, c, T, ρ, μ

and Re to be defined by user input. The prototype can visualize incompressible, viscous flow, within two to three seconds of initialization time, by means of a velocity vector field, a vorticity field and particle streamlines. Furthermore it can provide the user with aerodynamic forces and moments M', L', D', the corresponding non-dimensional coefficients C_m, C_l and C_d, the aerodynamic center, the center of pressure and a C_p-distribution graph. For this tool a frame rate of at least 30 frames per second was achieved.

The result is a tool that can help understanding airflow around an object and can aid designers to communicate their ideas to others in an easy, fast and interactive way. Furthermore, the tool could be used for educational purposes and be mounted in classrooms or science museums. The Eddy simulation tool will cost €10,000. Future marketing campaigns will target on aerospace companies specifically, by emphasizing the communicative and interactive nature of the tool.

Further development opportunities lie in increasing the computational speed of the program by improving the pressure solver and the graphical output, since these are the current bottlenecks of the program. Next to speeding up the program, more features and functionalities could be added, e.g. a more accurate determination of the aerodynamic center, simulation of flow around 3D objects or simulation of compressible flow.

List of symbols

c	Chord length	$[m]$
C_d	Drag coefficient	[-]
C_l	Lift coefficient	[-]
C_m	Moment coefficient	[-]
C_N	Normal coefficient	[-]
C_p	Pressure coefficient	[-]
C_R	Resultant coefficient	[-]
C_T	Tangential coefficient	[-]
$\boldsymbol{D'}$	Drag per unit span	$\left[\frac{N}{m}\right]$
ds	Arc length	$[m]$
$\boldsymbol{F}$	Force	$[N]$
g	Boundary condition forcing term	[-]
$\boldsymbol{L'}$	Lift per unit span	$\left[\frac{N}{m}\right]$
L/D	Lift to drag ratio	[-]
M'_{ac}	Moment per unit span around aerodynamic center	$[N]$
M'_{LE}	Moment per unit span around leading edge	$[N]$
$\boldsymbol{N'}$	Normal force per unit span	$\left[\frac{N}{m}\right]$
$Normals_{exact}$	Normal unit vector at exact boundary	[-]
$Normals_{grid}$	Normal unit vector at boundary grid points	[-]
N_x	Number of x grid points	[-]
N_y	Number of y grid points	[-]
p	Static pressure	$\left[\frac{N}{m^2}\right]$
p_∞	Free-stream pressure	[Pa]
q_∞	Dynamic pressure	$\left[\frac{N}{m^2}\right]$
R'	Resultant force per unit span	$\left[\frac{N}{m}\right]$
Re	Reynolds Number	[-]
T	Temperature	[K]
T'	Tangential force per unit span	$\left[\frac{N}{m}\right]$
$\boldsymbol{u}$	Velocity x-direction	$\left[\frac{m}{s}\right]$
$\boldsymbol{u_\infty}$	Far field velocity	$\left[\frac{m}{s}\right]$
$\boldsymbol{v}$	Velocity	$\left[\frac{m}{s}\right]$

x_{ac}	Location of aerodynamic center	$[m]$
x_{cp}	Location of center of pressure	$[m]$
α	Angle of attack	[deg]
$\boldsymbol{\Gamma_{exact}}$	Exact boundary points	[-]
$\boldsymbol{\Gamma_{grid}}$	Grid points at shape boundary	[-]
μ	Dynamic viscosity	$\left[\frac{kg}{m \cdot s}\right]$
ν	Kinematic viscosity	$\left[\frac{m^2}{s}\right]$
$\boldsymbol{\Omega}$	Grid points inside shape	[-]
$\boldsymbol{\omega}$	Vorticity	$\left[\frac{1}{s}\right]$
ϕ	Velocity Potential	$\left[\frac{m^3}{s}\right]$
$\boldsymbol{\psi}$	Stream function	$\left[\frac{m^3}{s}\right]$
ρ	Density	$\left[\frac{kg}{m^3}\right]$
ρ_∞	Free-stream density	$\left[\frac{kg}{m^3}\right]$

List of abbreviations

ac	Aerodynamic center
AMG	Algebraic Multi-Grid
ATX	Advanced Technology Extended
BC	Boundary Condition
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CSR	Compressed Sparse Row
DDR3	Double Data Rate 3
DSE	Design synthesis exercise
FBD	Functional Breakdown Diagram
FFD	Functional Flow Diagram
GCC	GNU C Compiler
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HDD	Hard Disk Drive
IBM	Immersed Boundary Method
IDE	Integrated Development Environment
MAC	Mean Aerodynamic Chord
MMD	Multidimensional Market Definition
MSVC	Microsoft Visual C++
PP	Post-Processing
PPE	Pressure Poisson Equation
PREP	Pre-processing
PSU	Power Supply Unit
px	Pixels
RAM	Random Access Memory
RHS	Right-Hand Side
ROI	Return on investment
SE	Systems Engineering
SSD	Solid State Drive
SWOT	Strengths, Weaknesses, Opportunities, and Threats
VIC	Vortex-In-Cell

Contents

1. Introduction

As capstone to the bachelor curriculum of aerospace engineering, students have the opportunity to work through the design experience, working every facet of the design process starting from specifications given by the client up to the final conceptual design. Knowledge gained in different courses throughout the bachelor curriculum is applicable to the various design problems the team faces.

In this design synthesis exercise [7] an interactive wing design tool will be developed by ten students in ten weeks. The complete design of the tool includes the conceptual design, in which a design concept is chosen by way of a careful trade-off. Construction of a prototype (for which €2,500 is available) and design and programming of the tool will happen simultaneously, the results of which will be presented at the closing symposium.

The tool shall at least be able to simulate the airflow behaviour around two dimensional objects. The demands of the clients are to make an interactive and innovative tool with a natural interface, by using non-conventional concepts such that flow phenomena can be explained intuitively. This requires real-time computing, while providing a suitable level of accuracy and functionality for engineers.

This final report describes the complete design process towards a working end product that meets the requirements. Emphasis is placed on the latter design phases, because the initial phases are detailed in the baseline report [2] and project plan [3]. First, the activities performed in the conceptual phases are explained. Chapter 2 discusses the design goals, chosen concept and the priority scheme. In chapter 3 an overview of the program structure is given using comprehensive block diagrams. The following five chapters explain the functions of each block of the program architecture. Herein, theoretical background is explained and the way the functions are programmed. Chapter 9 gives and overview of the hardware components of the tool and the performance that the complete system can deliver. An analysis of the end product is given in chapter 10. The last chapter (11) discusses options and improvements for further development for post-DSE activities. In extension of this report, a separate manual/documentation will be made for the tool.

2. Start of tool development

This chapter describes the starting phase of the tool development. First the goal of the design and development project is set out. Secondly, the concept that was found to comply with the requirements best is explained, followed by the steps in development listed by priority. Finally the sustainable development strategy is discussed.

2.1 Design and development goal

On request of the client an interactive environment for simulating flows in real-time is to be designed and developed. Such a tool can aid design engineers in getting rapid understanding of the flow behavior around arbitrary objects or shapes, but can also be used as a teaching aid, in classrooms or other educational environments such as classrooms or science museums. From these objectives the following three main challenges arise for the tool:

- Interactivity
- Real-time computation
- Rapid visualization

This all should be realized within a budget of €2,500.

2.2 Chosen concept

A concept that could realize the afore mentioned requirements was found in an environment where the user can draw an arbitrary shape in the user interface and receive immediate feedback on the flow behavior around that object.

Different hardware solutions were found for this concept. A trade-off between these solutions was done based on criteria such as feasibility, costs, required computing power, ergonomics, appeal and applicability. This resulted in the decision for integrating the tool with an interactive beamer and a PC.

One of the main advantages of using an interactive beamer is that image recognition is already included in the beamer to establish the drawing feature. For the design team, this excludes the task of having to program a similar function. Furthermore, the beamer can project and recognize the position of the pen on any arbitrary flat surface. This increases the applicability of the tool in many different environments. The user can use this pen to draw a shape or object on the surface where the GUI is projected. When having a good designed user interface, the image recognition of the beamer also increases the intuitiveness of the tool.

The tool will perform the required computations on the PC and the airflow and requested variables will be shown in real-time to enhance the interactivity of the tool. The advantage of a PC is that it can be custom build, such that hardware components can be chosen with specific qualities suited for the tool.

2.3 Priorities for prototype tool

The request of the client for an interactive wing design tool is of high ambition, just as the ideas and wishes of the design team are. In order to be able to deliver a working prototype at the end of the time frame available, priorities for the different prototype stages are set. This list of priorities is given in figure 2.1. The priorities are categorized, first by the part of the program which they correspond to, being GUI, calculations or CFD, and then input, output and lay out.
The prototype priority stages are:

- **Minimum**. This is a first working prototype, that can be fine-tuned and improved during the second half of the development time frame.
- **Symposium**. This is the prototype that is planned to be delivered and presented on the day of the DSE Symposium. It will be the final product of DSE group 2.
- **Useful**. This inherits the features that are needed for the tool to be actually useful as an airflow simulation tool. The aim is to implement these features in the symposium tool if possible.
- **Extra**. This category contains extra features that are beyond the useful prototype, but could be implemented in future versions of the tool.
- **3D**. This contains the features and variables that are required for flow simulation on 3D geometries. Again these features could be implemented in development stages beyond the DSE time frame.

2.4 Sustainable development strategy

The product designed during this project basically consists of a software program and therefore it will not directly be very harmful to the environment (e.g. in terms of greenhouse gases). This makes it difficult to develop an extensive sustainable development strategy. However, this does not mean that sustainable aspects should not be considered at all.

First of all, a possible objective of the software tool indirectly contributes to sustainability. An advanced version of the tool could for example replace windtunnel experiments on preliminary prototypes. Manufacturing of prototypes will be redundant, saving costs and material. Furthermore, since a smaller number of windtunnel tests will suffice for development of a (wing) profile, resources required for running the windtunnel are saved.

Secondly, the tool itself can be made sustainable by incorporating an energy saving mode. Furthermore, efficient programming of the tool requires less computing power. The energy consumption and recyclability of subsystems will be trade-off criteria for choosing hardware components.

Thirdly, it is likely that the amount of energy consumed during the development phase will be larger than during the operation of the tool itself. Therefore an energy conscious atmosphere is created among the design team during development. Turning off the laptops during breaks is one example of the sustainable consciousness.

Figure 2.1: Prototype priority scheme

3. Program structure

This chapter clarifies the structure of the program. First a functional flow diagram (FFD) and functional breakdown diagram (FBD) is made to identify the functions that the tool should perform. The next section shows the program architecture in which an overview of the program blocks is given and the flow of data through the program is explained.

3.1 Functional flow diagram

This section presents the functions that the design tool must perform. These functions are shown in figure 3.2. The sequence of functions is indicated by arrows. In the diagram four main functions are presented, being *Insert Input*, *Process Input*, *Compute Output* and *Show Output*.

3.2 Functional breakdown structure

The functions presented in the FFD are represented hierarchically in the functional breakdown diagram. The FBD is given by figure 3.3.

3.3 Program architecture

A program architecture is made to illustrate the flow of data through the system and to and from its environment. This architecture is shown in figure 3.1. The main program blocks in chronological order are the *Graphical user interface (GUI)*, *Pre-proccessing*, *CFD*, *Post-processing* and *Graphical output*.

GUI
The interaction between the user and the program is created via the graphical user interface. Herein, the user could draw an object, set values for certain parameters and change settings (*GUI IN*). The output (e.g. vector fields, pressure graphs and warnings) is also visualized via the graphical user interface to the user (*GUI OUT*). The link for the drawing feature is directly connected from the *GUI IN* to the *GUI OUT*.

Pre-processing
The pre-processing (PREP) block receives the input (shape coordinates, parameters and settings) from the GUI and further processes this such that the information required for the CFD solver and post-processing block are given in the correct format. This information includes the grid- and exact boundary, interior points, arc lengths and unit normal vectors.

CFD
Boundary information and input parameters are given to the CFD solver. Here the velocity and vorticity at each grid point are computed. These are given to the graphical output block and post-processing block.

Post-processing
In the post-processing (PP) block, the pressure and other aerodynamic parameters are computed using the velocities from the CFD solver and boundary data from the pre-processing block. The output is then send to the grapical ouput.

Graphical output

For visualization, the graphical output converts the raw data into graphs, pictures and animations. This block requires the velocity, vorticity, shape coordinates and aerodynamic parameters such that a vector field, vorticity field, particle plot and pressure graph can be made. The results are then send to the GUI, where the flow phenomena around the inserted shape is displayed to the user.

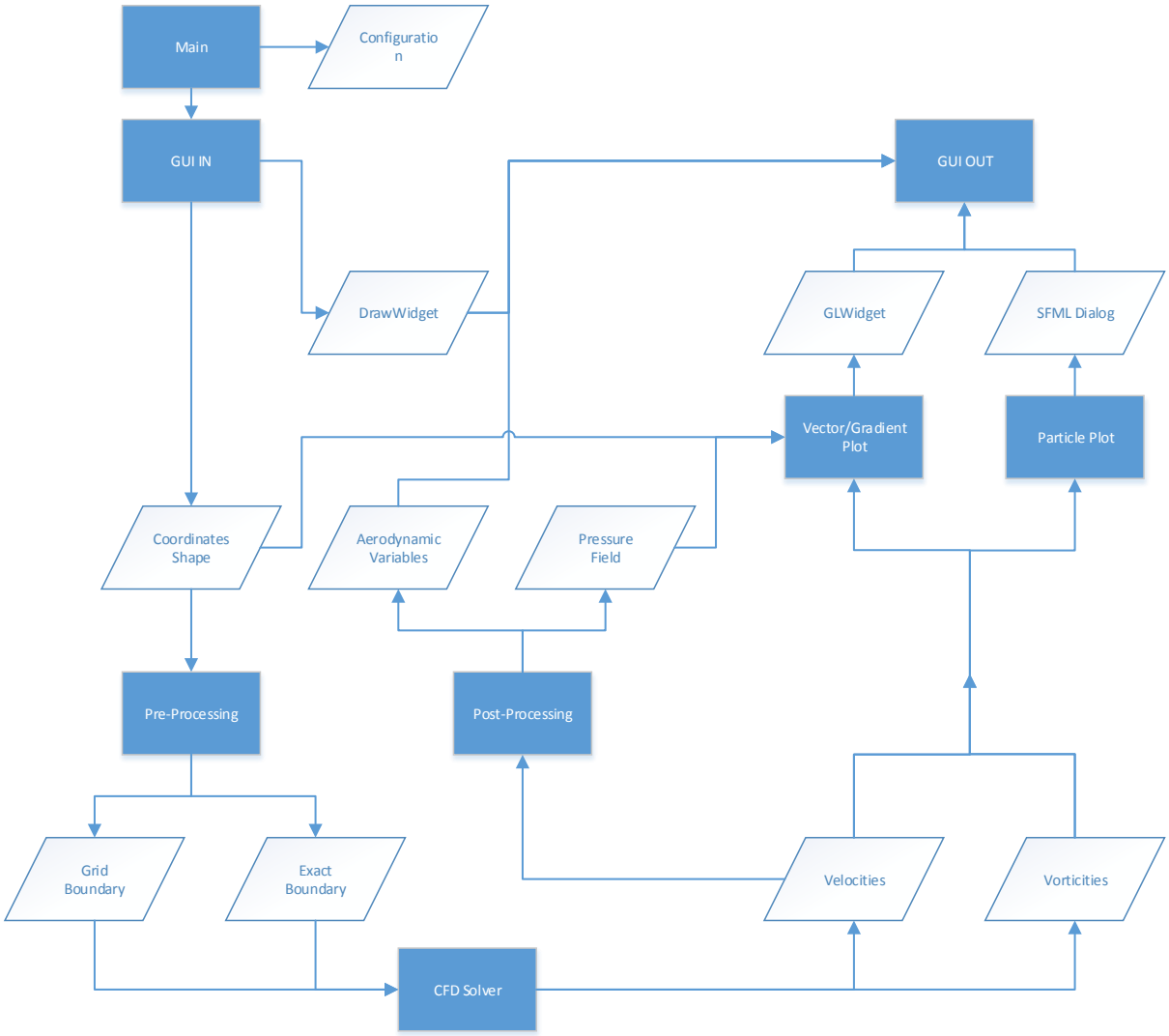


Figure 3.1: Program architecture

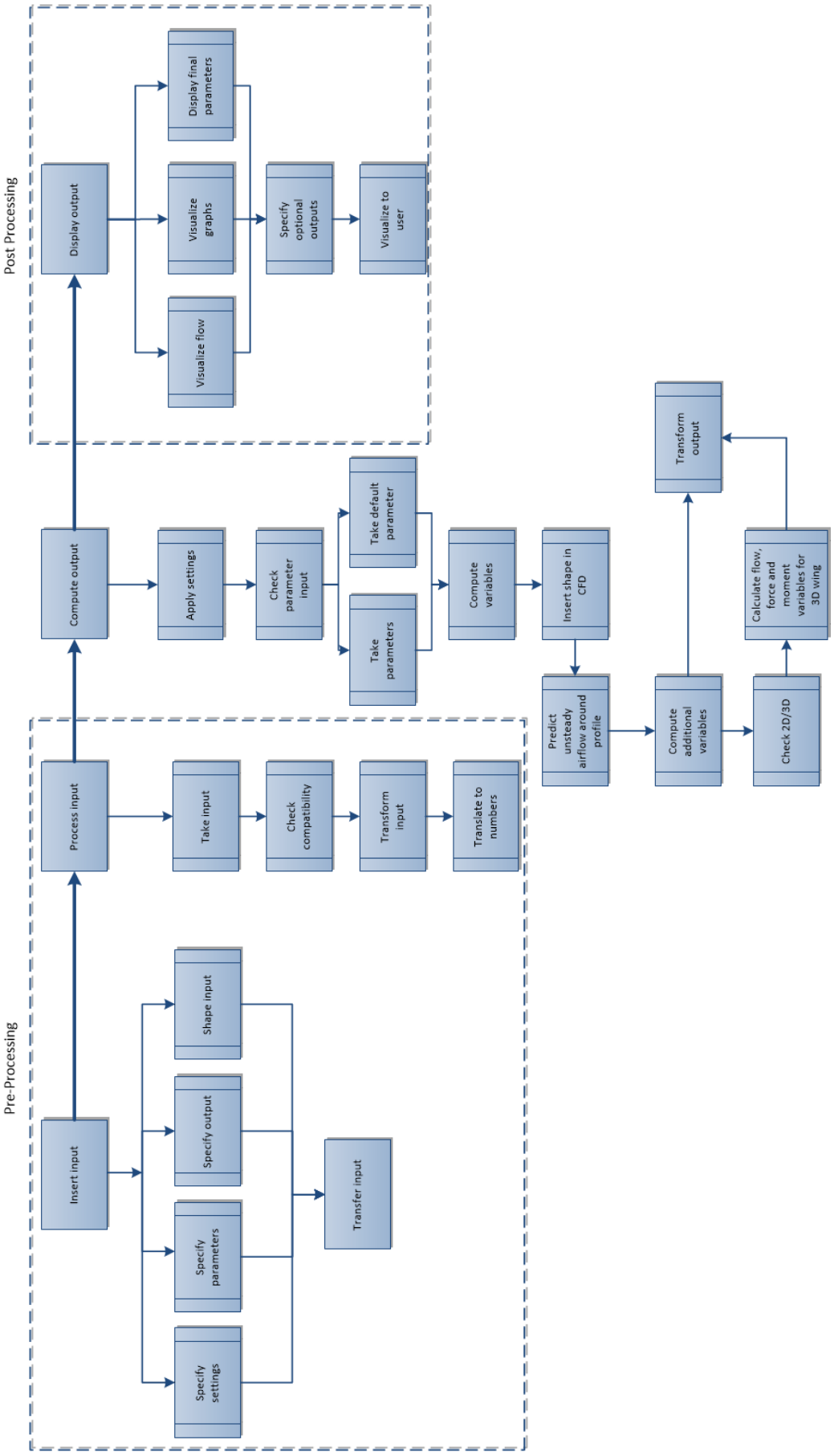


Figure 3.2: Functional flow diagram

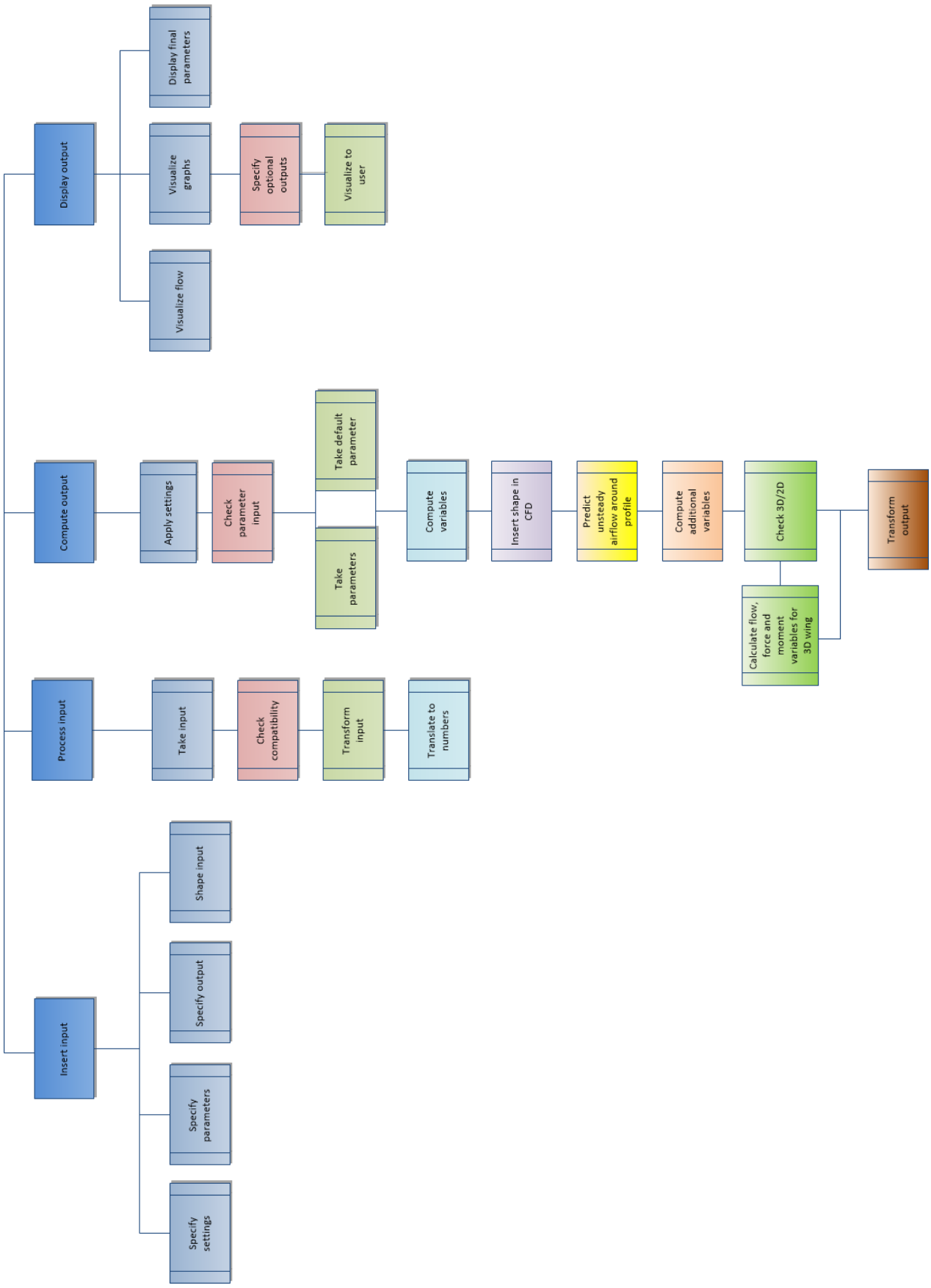


Figure 3.3: Functional breakdown diagram

4. Graphical user interface

This chapter describes the design of the GUI. The GUI combines all aspects of the tool and is the interface between the user and the tool. In the GUI, the user can insert the desired drawings, shapes, parameters and settings and can interact with these and activate certain parts of the tool. The GUI also shows the output of the system, like plots with vectors, gradients and particles.

The first part of this chapter describes the design process of the GUI. It lists the goals that need to be achieved and then goes into more detail how the environment was selected and finally the internal framework. The second part describes the final layout of the GUI. It describes the internal framework, important functions and features. It gives a clear overview of what it can do, and how these functions interact with themselves and the rest of the tool.

4.1 Design process of the GUI

This section discusses the design process of the GUI. Starting with the goals set for the tool. It then further elaborates on the chosen programming environment. This includes the IDE and the compiler. Thirdly, further research is shown into alternatives of C++ that might have been better suited for GUI or graphical output programming. Lastly it focusses on the design process of the drawing system.

4.1.1 Design goals

The minimum working prototype must be able to perform CFD calculations with predefined airfoil shapes. The airfoils and calculated vector field must be visible on the screen. The user interface must consist of a start/stop button and a small input menu.

With the minimum accomplished, extra options are added until the requirements for the symposium are met. These requirements are that it must be possible to draw a shape with the pointer of the interactive beamer. It also must allow for adaptations in the predefined airfoil shapes. Multiple objects (e.g. high lift devices) must give reliable flow behavior and several parameters need to be adjustable. The most important aerodynamic loads must be displayed and the menu must be attractive.

4.1.2 Research for development environment

First of all, an integrated development environment (IDE) is necessary. This is the interface which one works with, integrating all necessary elements to produce a program (e.g. code editor, user). The options considered were wxDev, Code::Blocks, NetBeans, QT Creator, and Microsoft Visual C++ (MSVC).

The choice of IDE in the first place heavily depends on the ease of integration with the necessary libraries and compilers, and secondly on the option for cross-platform work. The first dependency is due to the time penalty, and the chance of it not working, which comes with difficult integration. The second makes it workable as a group, to not all have to move to one platform, and be able to code during exploration.

The IDE needs a compiler, to be able to compile the code, debugging and releasing it, to be able to build a project in order to run it. Some IDEs come with their own compiler(s). wxDev and NetBeans do not, but are easily integrated with MinGW (port of GCC) and (for wxDev) Microsoft compilers.

For graphical output such as a vector field, GUI, and of course for other applications (e.g. graphs, mathematics), libraries are necessary. In Table 4.1 the libraries considered are listed.

Table 4.1: Graphical Output Libraries Overview

Type	Library	Comments
GUI	wxWidgets	Combination with vector field not simple
	QT	Very active community; intuitive editor
	FLTK	Also active community, less intuitive
Vector Field	QwtPlot3D	Outdated
	Gnu Octave	Maybe still an option, if GnuPlot embedding in GUI works
	wxMathplot	Integration with wxWidgets works well
	OpenGL	Fastest, low-level
	QT (Painter)	Intuitive
	AGG	Difficult, Windows difficult/ badly supported
	MathGL	Made for quality, not for speed
	GPC-QT	Like MathGL, also slow
Plotting	qCustomplot	Easy option for additional graphs
	GnuPLot	see Elaboration

The decision has been made to use QT Creator as an IDE, with its own integrated GCC (MinGW for windows) compiler, using its OpenGL or QPainter applications, with optional libraries such as qCustomplot. The setup and the use of this combination are simple, the application is broad, and there is strong support for it. Libraries not chosen, or not included in the table were usually omitted due to poor documentation. Logically, the integration with the non-graphical part of the program has been taken into account.

4.1.3 Alternatives to C++

Most of the libraries were difficult to install or implement. The installations were poorly documented. This gave many problems so Hans Geers, teacher of C++ at EWI, was contacted to help get libraries working. After explaining the project and project goal, he concluded that C++ would probably be too difficult and JAVA would be easier to use. It also has graphical libraries itself, so the problem with libraries would also be solved. JAVA was not taken in the trade off since nobody in the team had ever worked with it. Not knowing if JAVA would be a better option, it was decided to both continue with C++ and to give a look at JAVA. Within an hour it was already possible to give graphical output, without any previous knowledge in JAVA. To compare: three days were spent on C++ already without visual output. Getting outputs in JAVA was relatively easy. However, meanwhile also output was generated with Qt, which uses C++ and has graphical libraries included. A choice had to be made on which program would be used for the final version. The option on JAVA was discontinued due the complexity of the integration of JAVA and C++.

Another alternative which was considered is the C# language in combination with the XNA engine. This could potentially provide for a real time graphics window and an easy to use syntax. The C# language is not fast enough when dealing with the computationally intensive tasks needed for the prototype. Therefore C++ would have to provide for this. Again this would require a rather complex integration. And these are known to be less efficient than a C++ only program.

The use of gnuplot for graphical output generation was investigated because it seemed a feasible option for the plotting part of the graphical output. Gnuplot is relatively easy to use, it has very simple and intuitive syntax and a users guide, which is written in clear language. For plotting purposes only, gnuplot was an interesting option. It was also possible to run it from C++ code. Furthermore it gives fast output. The only disadvantage was that another program would be needed for graphical input and output, which allowed for user interface. With all the difficulties to get graphical libraries working, integrating many different programs seemed to be a challenge, unsure on whether the time to get them to work would pay off. Preference was given to using one program for all the graphical interfaces, to avoid difficulties with implementing multiple libraries.

4.2 Design process of the drawing system

This section describes the process of creating the draw system. It started with visualizing basic inputs in OpenGl and QPainter. The first goal was achieved when an airfoil could be drawn, and its coordinates read out, as described in the second subsection. It then elaborates on the first features involving mouse movements. It then became clear that the QGraphicsView would allow for better interaction and professional outlook. The final integration of the complete tool is then described in section 4.2.5. Finally the design of the graphical design is elaborated upon.

4.2.1 Visualizing input with OpenGL and QPainter

The input given to the prototype must also be visualised as output. The first goal was to achieve a drawing of an airfoil based on given coordinates. This was first done using OpenGL. The basic idea was to put the coordinates one by one in different vectors and then draw lines between the different coordinates, coloring all pixels enclosed by the airfoil. The next step was then to couple input data to the Qt program so that all coordinates were loaded automatically, so it would no longer be necessary to put in all coordinates manually. It became quickly clear that using only Qt libraries for drawing was more straightforward. Therefore, OpenGL was traded for QPainter. This required finding out how drawing coordinates with QPainter works. It appeared that QPainter was simpler to use than OpenGL. There is a downside. Seperating input and output does mean that these have to be shown on different windows. At the time, that seemed the best way to go to get the program working.

4.2.2 Implementing airfoils

To include airfoils, a feature of Qt is used, the ComboBox. This gives a dropdown menu from which the user can select any airfoil he wants. A database is used of airfoils with their variations. [11]. It features using the keyboard to search the first letter of the airfoil, as the list is quite extensive. When one airfoil is selected, its x and y coordinates are read out and shown on the screen.

Also a first clearing function was implemented, to prevent multiple airfoils from being loaded. At the time, it cleared only the airfoil previously drawn, allowing to also scroll through the list.

4.2.3 Drawing with mouse and first interactive features

The next step was to allow inputs from mouse movements. The reason for this is that the interactive beamer works with a pointer, which is a pen with the uses of a mouse. The idea here is to track the mouse movement and put the coordinates along the track in an array. The array is then used for drawing points at the different coordinates, but also in a modified way to feed to the CFD solver. To make it useful and interactive, this function is activated when the user holds the left mouse button pressed. The pointer of the interactive beamer is activated when pressing it against a surface (this is comparable to clicking). The reaction to this input can be set as any of the different mouse inputs.

To allow for multiple objects to be drawn, each was stored in a list. A button must be pressed to create a new object. The shape to be modified could be selected in a dropdown menu. And the first function to be able to read out these coordinates again for the solver was implemented.

The first interactive features included translating, rotating and scaling the airfoil. As transformation point, they used the first drawn point. Thus when in translating mode, the mouse would not draw, but the selected shape would be translated to the mouse, where its first coordinate would correspond with the mouse location. The rotation and scale are implemented as a slider and an input box. More features in the first stages were a clearing the selected shape and clear all the shapes.

4.2.4 Switch to QGraphicsView framework

The graphical user interface and the menu at first were one and the same screen. This meant that everywhere in the widget, where the mouse button was pressed, a drawing was made. To avoid this and also to give the program a more professional look, it was decided that the menu and the graphical output part had to be two clearly different outputs, shown in the same window. Qt has a solution for this, QGraphicsView. Switching to this gave new problems but also some new options.

QGraphicsView works as a widget, that takes a QGraphiscScene. The scene again, can include QGraphicsItems, which are shown on the screen. These items store the information about each shape. The originally drawn coordinates are stored, and also values for its rotation, scaling and translation. As more was learned about this framework, it also became possible to modify these items and make all features a lot more intuitive.

A custom scene class was created to allow the mouse events to be recognized. When the mouse is pressed, automatically a new item is created and when it moves, points are added to the item. Making these items also movable and selectable allowed for very intuitive editing of these items. Items can be selected by the mouse. The selected shapes can then be modified, deleted, and dragged and dropped at will.

Also collision detection was added, to prevent the drawing, when moving, from going outside of the screen. If on the border, it moves back. If outside, the item is deleted. QGraphicsView, makes these operations easy and accessible. Clearing, translating, rotating, scaling, selecting, finding selected items, collision detection are all implemented in the framework and work very fast.

4.2.5 Complete integration of the tool

Now that drawing of airfoils and shapes become more mature, as well als the other aspects of the tool, it was time for integrating everything. The GUI works with widgets. In dock-widgets on the left and right side, the buttons could be placed. In the "centralwidget" is then shown the drawing widget, or the plotting widget, or other output visuals.

When the plot widgets are accessed, the CFD solver starts running and takes the coordinates from the drawing. The Pre-Processing block is implemented in the GUI and activated when the coordinates are accessed. It loops thought the coordinates of each drawn item and converts them into the Grid and Exact Boundary input required by the solver. This is explained in more detail in section 5.1. To the GUI a tabwidget was added, where buttons could be stored for each widget.

4.2.6 Adding a graphical design

To make the the graphical user interface attractive, adjustments were made to the layout of the user interface. Qt allows to personalize the complete user interface. The background of the widget, graphics view and also the buttons can be changed. Changes for these backgrounds can be single colors, gradient colors or imported images. The borders of each of these can also be changed. Different radii for the corners create a different look. Large radii make rounder buttons and vice versa. The user in general wants feedback when pressing a button. This can be done by creating 3D effects or by changing the location of the text in the button downward. Using 3D effects to accomplish the feedback is done by first selecting a different color for the border of the button and give the button an outward padding. When clicking the button, the padding is changed to inward and the colors of the button and border are interchanged. For tabs this is done slightly different. This is firstly because the selected tab must not only be different than the other tabs when it is clicked but it must stay selected until a new tab is selected. Also, user intuition tells that when a tab is selected it lays more forward than the other tabs and often has a lighter color.

For the final look, many different color combinations were tried. Also many different background patterns were tried. However, the simplest lay out, using only three colors and no special background, seemed the best looking. The user interface can be further enhanced by changing the factors named above.

4.3 Layout and workings of the GUI

To explain the workings of the GUI in detail, this sections starts with explaining the overal layout of the GUI. Where is everything located? And describes how widgets are managed in the GUI. Secondly, the signals and slots of the GUI is explained. How are all the buttons connected with the code? Then the inclusion of the Pre-Processing block is elaborated upon. It describes how it is included, how coordinates are read out and given to the CFD solver. The last section is a extensive description of how the drawing widget works. It lists the framework, important program structures and all the features and their workings.

4.3.1 Overal Layout

In figure 4.1 the overal layout of the GUI is given. It gives an overview of where the different widgets are allocated. Widgets are the primary elements for creating a user interface. They can be embedded or used as a separate window. They display data and status information, receive the input and can also contain more widgets.

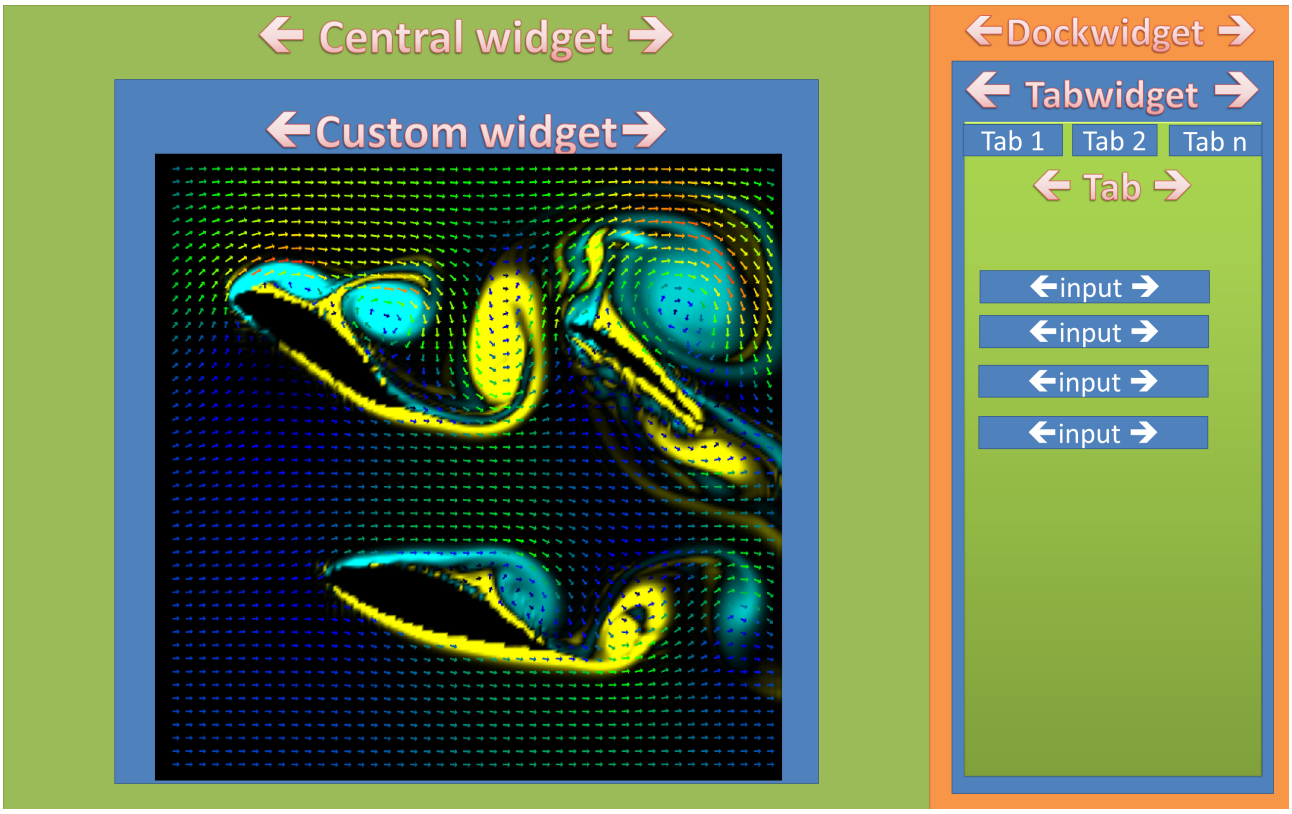


Figure 4.1: An overview of the GUI

The GUI can be set to full screen, and then has a resolution of 1280 x 800 px. The most important feature is drawing and plotting, and therefore most of the screen space (760x760px) is allocated for the central widget containing these widgets. On the side on the right, buttons and input widgets can be found. These are contained in a tabwidget, which can also be used to select the visible central-widget. In this way, the right buttons for the right widget are always visible. On the left, important parameters can be inserted, that are general to all the widgets.

The tool consists of four widgets. The first one is drawing. The second is the vectorfield and gradient plot. The third is the particle system. The last is the pressure plot and showing aerodynamic values. These widgets are created at the startup of the program. When another widget is selected, the previous widget will not be deleted, but is explicitly stored. To prevent them from making unnecessary calculations, their times is paused, and possibly the solver is reset, depending on the case.

4.3.2 Signals and slots

To change the behaviour of the program, or to change parameters and settings, the GUI needs buttons and other input widgets to take input from the user, and send that to activate the program. Qt uses a signals and slots system. Any class derived from a QObject class or a subclass hence off can send and receive signals.

This mechanism allows for a continuous running GUI, that only activates part of the code when needed. For example, sending information to the CFD solver can be triggered when pleased, or can be paused/stopped by the user.

This signals and slots system is also the connection between all the input widgets with the program. Buttons, dropdown menu's, sliders and such are input widgets available from Qt, and able to send different kinds of signals, or receive them. For example, a button can send a signal for being pressed, released, clicked, toggled or being destroyed entirely.

A signal can also contain information, such as values, states or entire data structures. They can therefore be used to transfer information from one part of the code to another, no matter their hierarchy in the code.

To implement this system, the class of an object must have a signal or a slot function. Two objects of different classes can then be connected by setting up a connection between a signal and slot. The signals from the input widgets are automatically available, and only need to be connected to a slot. This slot can be in the mainwindow widget directly, or be created in subwidgets or any other class.

4.3.3 Connection to the CFD solver

The CFD solver is setup in such a way that an object of it is created at the startup of the program. It has public functions available to take input data and allow the cfd solver to be reset, as well as other functions. The GUI includes the drawing widget, and needs to send the data from that widget, to the CFD solver by means of its input functions.

The CFD solver requires the input in a specific format. The pre-processing block is implemented to reformat the data from the drawing into input for the CFD solver. Because the CFD solver is only used when outputting information, the pre-processing block is only called upon when the plot widget is requested by the user, and only if he accessed the drawing widget before that. The exact flow of information is illustrated more clearly in figure 4.2

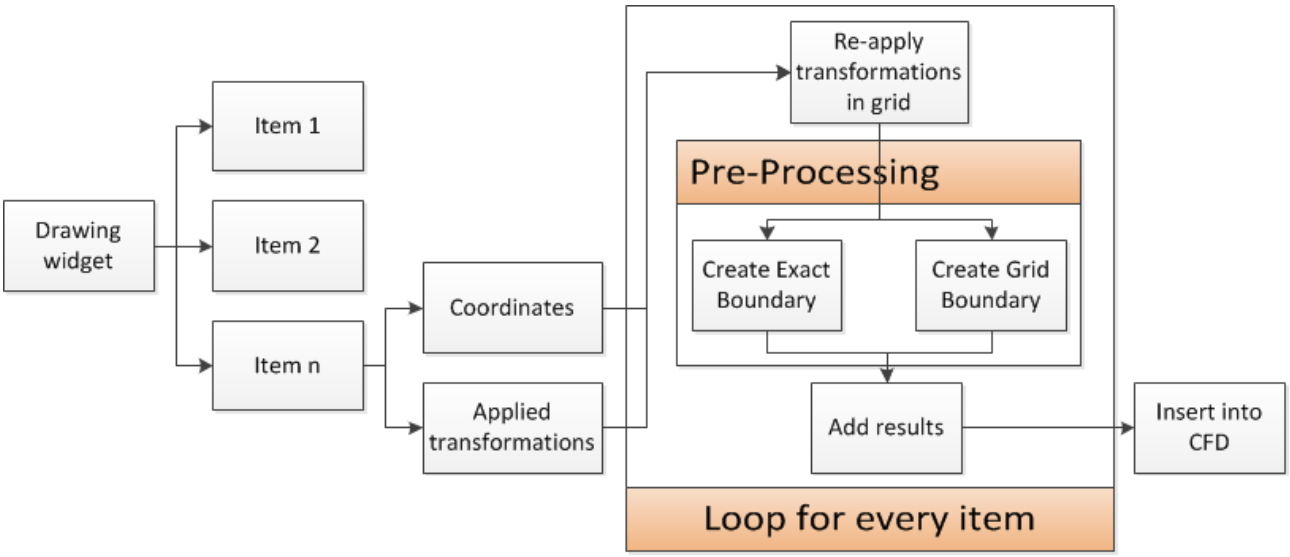


Figure 4.2: The flow of information from the drawing widget to the CFD solver.

In the drawing widget, the drawn coordinates are stored in items, as well as any applied transformations, such as rotating and scaling. The data is read out from the items, and the transformations

are applied again. Then the information for each item is send to the pre-processing block, where it is reformatted to boundary information for the CFD sovler. The result is added up and then inserted into the CFD solver object. The drawing widget will be explained in more detail in the following section.

4.3.4 Drawing

The drawing widget is the central widget where the user can draw any closed, non self-intersecting shape. The user can interact with these shapes, load predefined shapes and sent them to the CFD solver and other parts of the program. This section describes the framework behind the widget. It explains each of the features in more detail.

QGraphicsView framework

The drawing widget is visualized and interacted with, by means of a QGraphicsView. QGraphicsView is a QWidget that takes as input a scene of class QGraphicsScene. This scene contains objects of class QGraphicsItem, which are the items that are visualized on screen. The advantages are described below.

- Items are stored automatically in the QGraphicsScene, and can easily be accessed, based on different properties. These properties include selection, visible, location and name.
- It provides a fast interface for managing a large number of items.
- Individual items can be managed individually through selection. This is intuitive and works well with the mouse.
- It provides collision detection between items
- The view is easily implemented in the GUI and looks professional.
- Items can be transformed in multiple ways and these are automatically recorded. This is especially important for retrieving the coordinates for the solver.
- Items have tooltips and can be given style properties for a better outlook.

In order to be able to work with all of these functions, a custom scene class was written, named DrawScene. This class inherits the properties of QGrahicsScene. Certain functions can then be re-implemented to adjust their behavior. This way drawing with the mouse is also implemented.

To receive input from the buttons of the user interface, DrawScene contains slots that are connected to the signals of the GUI buttons. There are slots for changing the rotation and scaling factors, clear the current selection, clear all the items and for loading the airfoil. The last one is implemented twice, one for when a airfoil or preset is selected, and one for when it is only highlighted. This allows for easy scrolling and selecting presets.

Interaction with the mouse

Drawing with the mouse is by re-implementing the mouse event functions of the QGraphicsScene class. This is done for press, release, move and double-click mouse event functions. When the user presses the left mouse button, a new item of class QGraphicsPathItem is created at the clicked position. This convenience class plots the path on the view. When the user then moves the mouse, these coordinates are added to the path of the item. If however the mouse was not moved, this is recognized during mouse release, and the created item is again removed. When moving the mouse, the code checks if the coordinates are within the view. If not, they are not drawn, but the user can continue if he moves the mouse back into view.

A number of properties are added to the items. They are given a pen, giving the lines a color and a thickness. A brush colors the area beneath the shape. Also their flags are set to movable and selectable, allowing for selecting and then moving the items with the mouse.

When selecting shapes, the default behavior is that no multiple shapes can be selected, unless the ctrl key is pressed. By changing the control modifiers of the original mouse event functions, this is undone. If a single shape must be selected, this can now be done by double-clicking that shape. The double-click event function is changed to select only the shape that is under the mouse.

Modify shapes

As every created item can be selected, they can then also be modified using the options of the GUI. At this moment these include rotating, scaling and clearing the shape.

The transformation center for these modifications is set to the center of the rectangle that would enclose the entire shape. The rotation and scaling is saved in the items transformation matrix. This matrix describes all transformations made to the item, including the translation from its original point. Using this information when the original coordinates are read out for the CFD solver, allows to read out that data at the position desired by the user.

Clearing an item removes the item from the scene.

Reading out airfoils and presets

When airfoils or presets are read out, a signal is send from the GUI's combobox to read out the file with the selected preset's name. This function is implemented such that when a new preset is loaded, a new item is created. But when scrolling or highlighting shapes, they are read out, shown, and deleted when another is selected.

A preset file needs to simply have a list of coordinates seperated by a comma. If the read out file is not in the right format, no shape will be read in. It can have a first line to describe the contents of the datafile.

Applied filters and detection of forbidden combinations

A number of filters and detection of undesired situations has been implemented. This sections lists all the filters applied.

The first one is that if a shape is moved outside of the screen it is deleted. This is checked during the mouse release event for every selected. If a shape is on the border of the view, it is set to its previous position. This filter uses the built-in collision detection of the QGraphicsView framework. It checks for collisions with manually added items representing the borders of the view. This filter also implies that shapes can be removed by moving them outside of the view.

A second filter is that if a shape intersects itself, drawing is stopped. This is done by checking for every point added to the path of the item, if the line to it intersects with any section of the path of the item. If so, drawing is stopped, and any sections drawn before the intersection are removed. This results in a neatly closed shape.

5. Pre-processing

From the drawn input, the data needs to be formatted to be compatible with the CFD solver and the other aspects of the tool. Users can draw many difficult or incorrect shapes, and the format of the data would not be compatible with the solver.

The PREP block focuses on filtering the undesirable shapes and format the data such that the CFD can apply the boundary conditions to it. It also makes the data available for use in any other part of the tool.

This chapter elaborates on the PREP block. It describes firstly what types of data it takes as input and gives as output. Secondly, the steps taken in formatting the data are described. The structure of this block is shown and each function will be described in detail. Lastly, the applied filters are explained, describing how the robustness of the program is increased, to handle with unexpected user input.

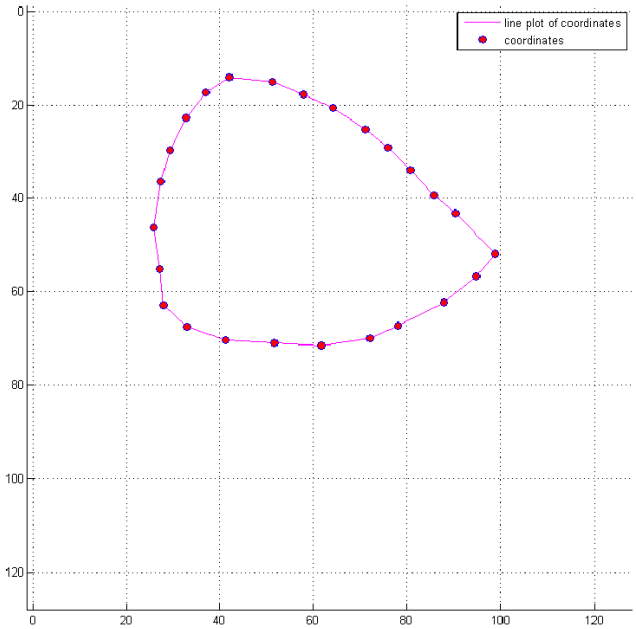
5.1 Input and output of pre-processing

In this section the input and output of the PREP block is described. It elaborates on the format and type of input and output.

5.1.1 Input from the drawing

The input from the interface exists of a list of coordinates for a selected shape on the screen. The PREP block is looped through these shapes and the GUI adds the results together. The entries of the list have to be sorted in the order that they were drawn. This is because the exact and grid boundary calculations loop through this list. This list is scaled to the grid used by the CFD.

The standard format is shown in equation (5.1). It lists the x- and y-coordinates in the 2D space, which are visually represented in figure 5.1. The space between the coordinates can be reduced manually.



$$\begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \tag{5.1}$$

Figure 5.1: The drawn coordinates visualized on a 128x128 grid.

It must be noted that in order to be more accurate, the user needs to take into account the limitations and assumptions of the tool.

Grid size Shapes that are smaller than the grid cannot be handled accurately.

Sharp edges Sharp edges can sometimes not be handled accurately, if the grid size is not high enough. Normal vectors on the grid points close to the sharp edge can point in random directions, because they will then be the sum of normal vectors of the line to and from the sharp edge.

5.1.2 Output to the CFD solver

The CFD solver requires two types of input. The first describes the shape on the gridpoints, further referred to as the grid boundary. The second describes the shape exactly, i.e. not on a grid. This is further referred to as the exact boundary.

The grid boundary describes the shape in three ways. Firstly, it lists the grid points closest to the shape. These points are further referred to as Gamma ($\mathbf{\Gamma}$). Secondly, it requires a normal unit vector for each gridpoint. This vector points normal to the surface on that point, defined in x and y direction. Lastly, the interior boundary points are required. These are all the points inside the shape, and are further referred to as Omega ($\mathbf{\Omega}$). The grid boundary is visualized in matrix notation in equations 5.2-5.4. The results for a NACA 0024 airfoil on a 64x64 grid is shown in figure 5.2.

$$\mathbf{\Gamma_{grid}} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \quad (5.2) \qquad Normals_{grid} = \begin{bmatrix} Nx_1 & Ny_1 \\ \vdots & \vdots \\ Nx_i & Ny_i \\ \vdots & \vdots \\ Nx_n & Ny_n \end{bmatrix} \quad (5.3) \qquad \mathbf{\Omega} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \\ \vdots & \vdots \\ x_m & y_m \end{bmatrix} \quad (5.4)$$

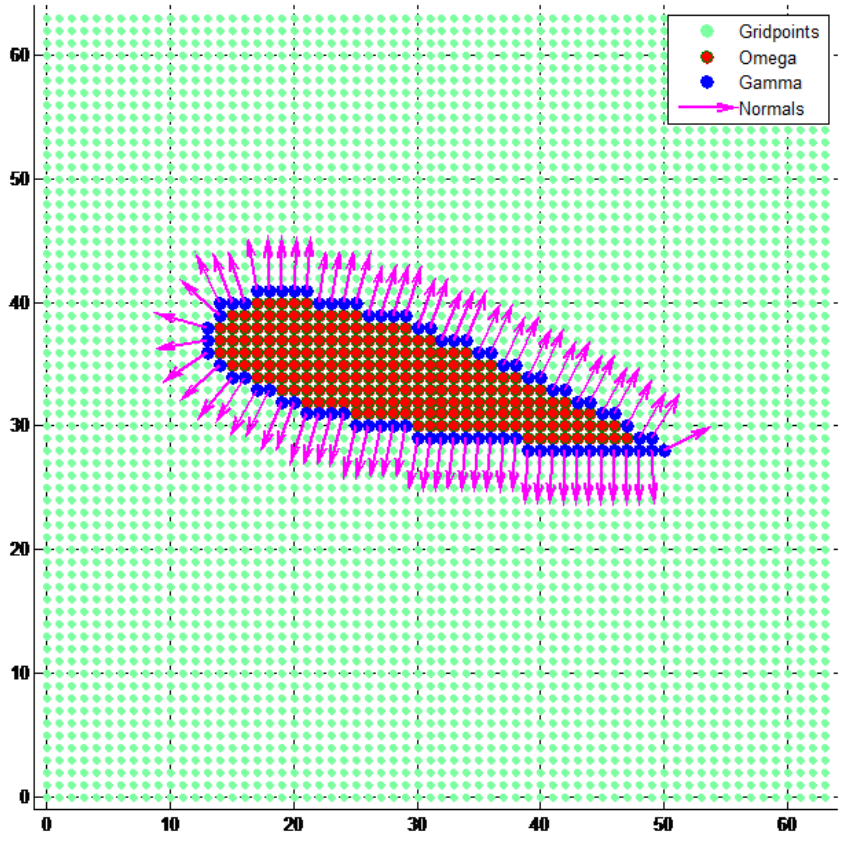


Figure 5.2: The grid boundary visualized on a 64x64 grid

The exact boundary also requires three types of information. It uses the lines between each coordinate that is inserted. It gives the x and y coordinate of the center of that line, the normal unit vector of that line, on that point, also in x and y direction. Lastly it requires for each line its length, the arc length ds. Those three types of information are visualized in matrix notation in equations 5.5-5.7. In figure 5.3 this is visualized again on the 64x64 grid.

$$\boldsymbol{\Gamma_{exact}} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \quad (5.5) \qquad Normals_{Exact} = \begin{bmatrix} Nx_1 & Ny_1 \\ \vdots & \vdots \\ Nx_i & Ny_i \\ \vdots & \vdots \\ Nx_n & Ny_n \end{bmatrix} \quad (5.6) \qquad arcs = \begin{bmatrix} ds_1 \\ \vdots \\ ds_i \\ \vdots \\ ds_n \end{bmatrix} \quad (5.7)$$

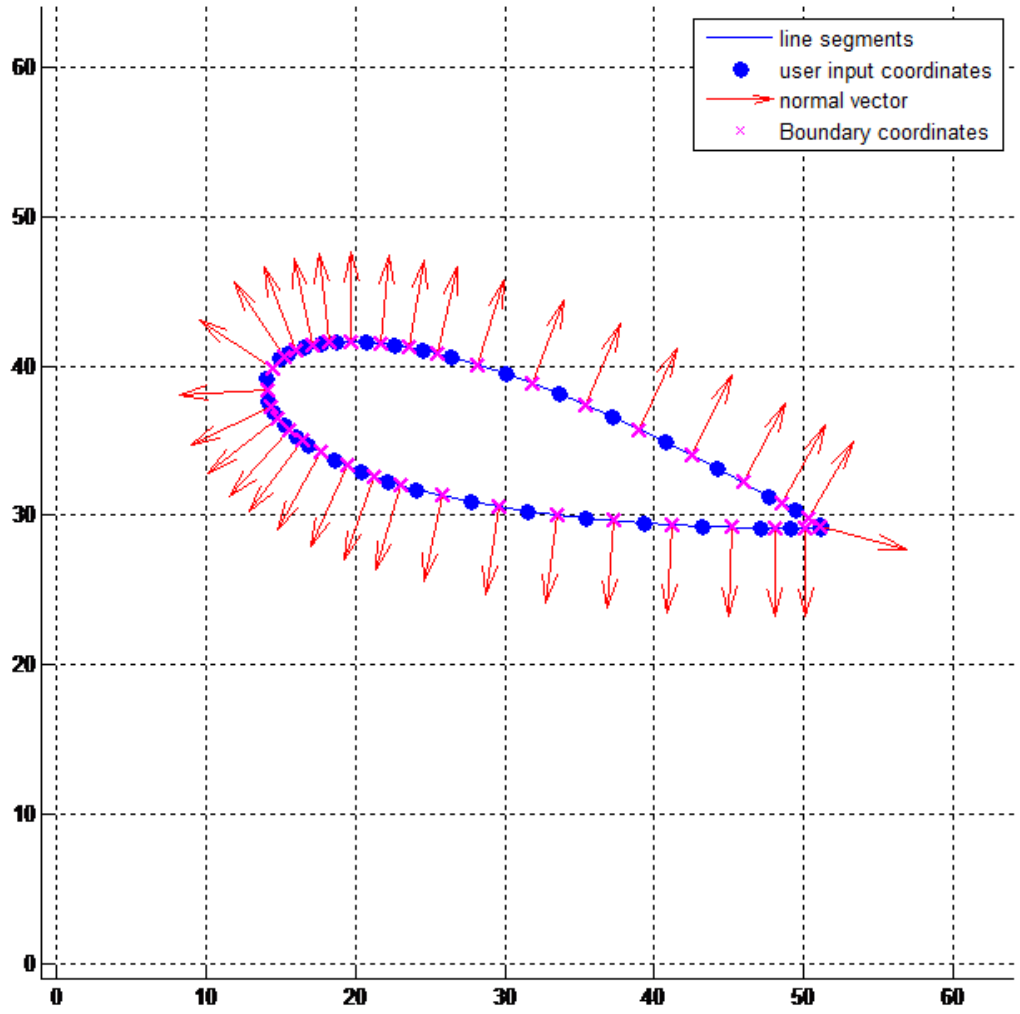


Figure 5.3: The exact boundary visualized on a 64x64 grid

5.2 Dataflow of pre-processing

To give an overview of the main functions, the flow of information is shown in figure 5.4. This is the step from taking the data from the user, reformatting and filtering that data and save it for the CFD solver. The different steps are explained in section 5.3.

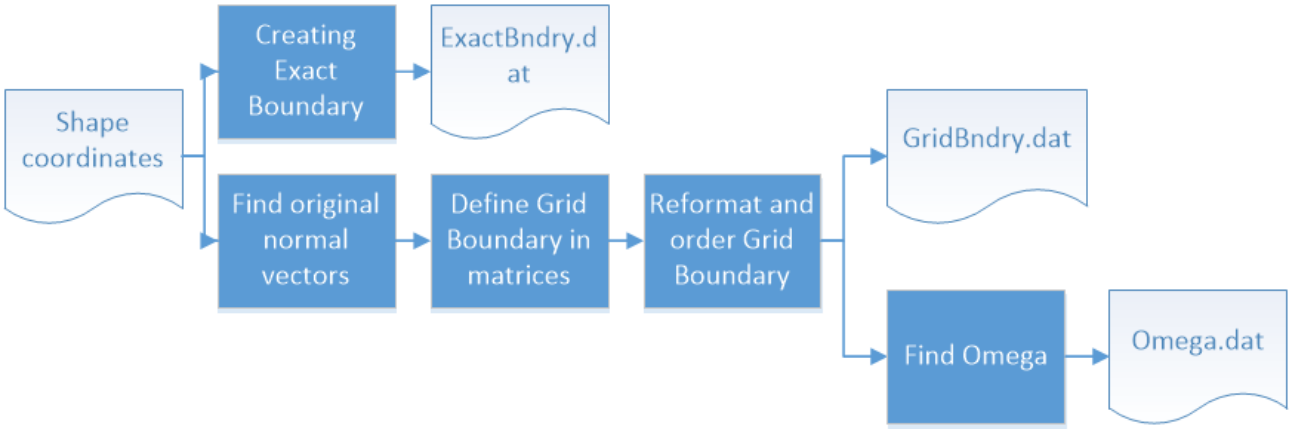


Figure 5.4: The flow of information of the refine input block

5.3 Theory and functions

In this section the calculations for the exact and grid boundary are discussed. For the exact boundary, these are quite straight forward. The grid boundary however takes more work. There, it is described how boundary Γ, the normal grid vectors N and the internal boundary points Ω are determined.

5.3.1 Exact boundary

The exact boundary is used for the viscosity part of the CFD solver. The solver needs the x,y-coordinates, the normal vector components and corresponding arc lengths ds as input. This is defined in equation 5.5 and visualized in figure 5.3. The function only requires the shape coordinates as input and then returns the matrix ExactBndry where each column corresponds to the following components: [x y ds Nx Ny].

The coordinates x and y are the average of the neighboring points, ds is the length between these points. Further, $N_x = (y_{i+1} - y_i)/ds$ and $N_y = -(x_{i+1} - x_i)/ds$. The (-) sign is in the N_y term, because it inverts the normal vector, as airfoils are usually drawn counter clockwise. If drawn clockwise, the (-) sign should be in the N_x term.

If the loop reaches the end, the values are computed between the last and first point of the user shape coordinates, i.e. it has to be a closed shape. If these points are the same, they are not computed, since ds would otherwise be zero, and have no effect on any calculation.

To enhance the accuracy of the solver, interpolation can be used to create a more accurate exact boundary. This will increase the smoothness of the vortices added to the flow to simulate the no-slip condition. In this program, only linear interpolation has been used. Cubic spline interpolation would offer a better accuracy, but has not been implented due to time constraints.

5.3.2 Grid boundary: Finding original normal vectors

To find the normal vectors of the grid boundary, first the normal vectors on the original airfoil shape have to be found, for each defined point. This is an important step, since using an incorrect method can lead to inaccurate and erroneous normal vectors.

The chosen method uses the two normal vectors closest to the point. For the normal vector on point j, the program takes the mean of the normal vectors of the neighboring line segments, which are known. In effect this is equal to the normal vector of the line between the neighboring points, which is how it is implemented in the code.

This method works well for smooth surfaces with evenly divided line segments. In future development, this can be replaced by more accurate models, using weighted factor, polynomials or other methods.

The last step is making each vector of unit size. Each of these vectors is divided by its magnitude.

5.3.3 Grid boundary: Find gridpoints and corresponding normal vectors

To create the grid boundary points and normal vectors, linear weighting is used. This means that between each drawn point a linear line is drawn and the nearest gridpoints to that line are then determined. To determine the normal vectors, a similar method is implemented. Here, the gridpoints closest to the original shape coordinates are assigned the normal vector as determined in the previous function. The gridpoints in between are then assigned a weighted average of the neighboring normal vectors.

If the space between the shape coordinates is smaller than the grid, the normal vectors are summed up. This gives a normal vector that is the average of the nearby original normal vectors.

The results are stored in sparse square matrices, equal to the size of the grid. The gridpoints are stored by placing an one in the matrix at their position. Normal vector components are stored in a similar way. The order in which the points are determined, is stored in a fourth matrix. This is needed to revert the square matrices back to vectors, such that the format can be as given in equation 5.2. The result of this function is a cubical matrix, where the slices contain: [Γ order Nx Ny].

5.3.4 Grid boundary: Reformatting and ordering boundary

In the function CreateGridBndry the cube received from the previous function is reformatted to arrays containing the information in ordered vectors. This can only be done now, because C++ does not handle varying size arrays very well and the total amount of gridpoints is not known before. Lastly, the normal vectors are converted to unit size. The ordering of the array is needed to find Ω. The result of the CreateGridBndry function is an array containing: [x y Nx Ny].

5.3.5 Grid boundary: Finding Omega

The last important step is to find Ω, an array containing the location of all points inside the boundary. To find these points, a function to check whether a point is inside or outside a polygon is used [8]. This method is very robust and can handle any closed shape implemented so far.

The pnpoly function applies raycasting to see whether a given point is inside or outside a polygon. This is then checked for every point on the grid. Although computationally the most expensive of refining the input, the time required is still very small. Also, these calculations only have to run once and the flow can still be visualized smoothly.

The result is again first stored in a matrix. The gridpoints are removed, as the pnpoly function cannot determine accurately whether a point on the boundary is in or outside. Finally the matrix is converted to an array containing the x and y coordinates in its columns.

6. CFD solver

In this chapter the CFD part of the tool will be explained. First a short introductory paragraph is given. In the next section, the basic principles and theory of the code will be explained. Section 6.3 shows the technique for solving the Poisson matrix. Then it is explained how speed-ups of the solver were achieved through GPU acceleration and in the last paragraph improvements and recommendations will be discussed.

6.1 Introduction

The CFD solver is the heart of the tool. All the computations to find the flow velocity, pressure and vorticity on which the output is based, are performed here. The code is based on a Python script, which was written by Dr. Richard P. Dwight of the Aerodynamics lab of the Aerospace Engineering Faculty of the TU Delft. The script again, was based on the work of Cottet and Poncet [6, 10], who use vortex-in-cell methods (VIC). The code still had its limitations, as it did not achieve satisfactory computation speed and therefore limited the frame rate of the tool. Another problem was integration with other program blocks such as the Output and the GUI, since these parts would be written with QT Creator in C++. Also for performance reasons, it was decided to rewrite the Python script into C++. This makes sure that integrating each part of the program would be a lot easier and would, through the use of GPU acceleration libraries (like Paralution), allow for more possibilities to make the CFD run faster.

6.2 Code explanations

The goal of this project was to develop a tool that can visualize incompressible, viscous flows around arbitrary objects. In order to accomplish this, mathematical equations are needed to calculate those flows. This is done by a CFD solver. The CFD solver gets its input as a list of coordinates. These coordinates form the boundary of a certain shape, for example an airfoil. The output of the solver will be a continuous flow of velocities and vorticities for every grid point at minimal 30 FPS. The CFD solver that is used in the development of the tool is a so-called particle method. This means that every grid point can be seen as a particle with a velocity and a vorticity. Every time step these values are re-calculated.

6.2.1 Initialization

The solver has an initialization part and a main loop. The first thing that is done in this part is the allocation of parameters. In this part the grid can be set to for example 256x256 units. The number of grid points is referred to as (N,N). Arrays for among others, velocity (v) and vorticity (ω) are allocated here. The vorticity array is of type (N,N), the velocity array is of type (N,N,2). The velocity thus consists of an x- and y-component for every grid point. A visualization of the grid is made in figure 6.1.

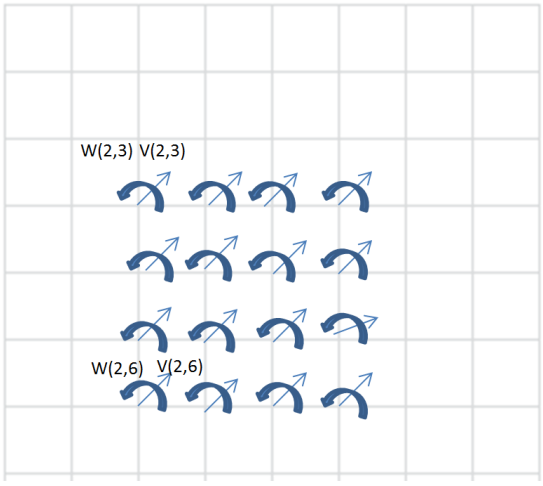


Figure 6.1: Vorticity and velocity on each grid point

Secondly, the Poisson matrix is created. A sparse format is used, because the matrix would get too large when a dense format is used. Paralution supports CSR (Compressed sparse row) matrices, therefore this format was chosen. The CSR matrix consists of three row vectors: one with values, one with column numbers and one with row indexes.

Thirdly, Paralution is initialized with specific settings. First the tolerance of the solver is set, this determines the precision of the calculation and has a substantial influence on the speed of the AMG solver.

Lastly, a so-called L-matrix is created. This matrix is later used to calculate ϕ, which in turn is needed to solve for the velocity. The matrix is not interesting in terms of physics, but the important aspect is that this takes quite some initialization time. This is because in the construction of the matrix the program has to go through the AMG-solver for M (the number of boundary points).

6.2.2 Main loop

In this section the computations that are performed in the main loop are explained.

Calculation of ψ

The first thing that is done in the main loop is the calculation of ψ. This result is obtained by solving equation 6.1.

$$\nabla^2 \psi = -\omega \tag{6.1}$$

The ω in this equation represents the vorticity. The vorticity has been calculated in the previous time step. In the first time step the matrix is filled with zeros, or, in the case a vortex is added, its filled with initial values. This step is a real bottleneck since Poisson equations are solved in a way that requires very large matrices. This leads to very long computation times. Section 6.3 will further elaborate on solving the Poisson equation.

Calculation of ϕ

The calculation of ϕ begins with setting up the so called L-matrix in the initialization phase. From this L-matrix the term g can be obtained. This g is then needed to solve for ϕ. This is done with equation 6.2.

$$\nabla^2 \phi = g \tag{6.2}$$

This is again a Poisson equation. The details of solving the equation can be found in section 6.3.

Interpolate vorticity

Another important function is the interpolation of the vorticity. As mentioned in the beginning of this chapter every grid point can be thought of as a particle. These particles move by a distance equal to the time step multiplied by its particular velocity. Then, for every grid point a weighted average of the surrounding vorticities is taken to re-calculate the new vorticity for the particle from that grid point.

Calculation of velocity

According to the Helmholtz decomposition the velocity consists of three parts [6, 10]:

- free stream velocity (v_∞)
- gradient of phi ($\nabla\phi$)
- curl of psi ($\nabla \times \psi$)

The free stream velocity is just a user input. The gradient and curl are mathematical operators. An algorithm to accomplish this is also included in the code.

Adding viscosity effects

In nature, the flow at the surface tangential to the surface is zero. Also the zone in the neighborhood of the object is affected. The layer in which this happens is called the boundary layer. So the challenge is to set the tangential velocity equal to zero. This is done by adding vorticity in the neighborhood of the boundary. The normal boundary which is typically between 128x128 to 256x256 is not sufficient to deal with boundary layers in an accurate way. That is why a more precise boundary is needed, in the solver this boundary is called the exact boundary. The exact boundary consists of a set of points that lie exactly on the boundary which the user has drawn or selected. A visualization of the exact boundary is shown in figure 6.2.

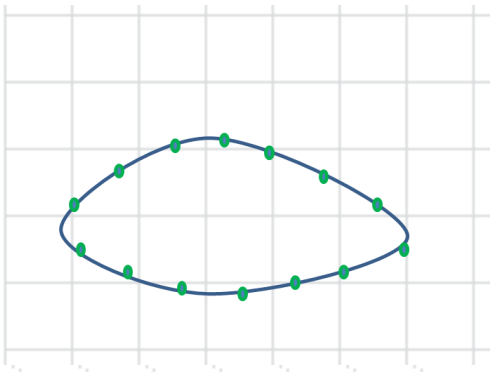


Figure 6.2: Exact boundary

The exact boundary is exactly on the boundary of the input object. The CFD solver gets this input from the pre-processing module, see chapter 5. At these exact boundary points the tangential velocity is calculated. This is then compensated by additional vorticities. This is shown in figure 6.3. The arrows on the boundary Γ represent the velocity tangential to this boundary. These velocities have to be canceled out, which is done by adding extra vorticities at the neighboring grid points.

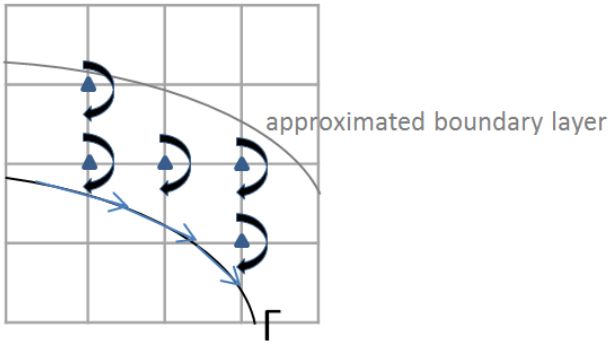


Figure 6.3: Vorticity added to satisfy no-slip condition

Set v and ω to zero inside boundaries

Finally the velocity and vorticity inside the boundary and on the boundary have to be manually set to zero.

6.2.3 Result

The final result of the CFD solver is an array of velocities and vorticities for every grid point. These results are then passed on to the post-processing group and to the graphical output group.

6.3 Solve Poisson equation

As said in section 6.2.2 the matrices in the Poisson equations get very large and therefore require very large computation times. For this reason fast solving methods where needed. This chapter will explain how Poisson equations can be solved in an efficient way. The upcoming part will deal with solving the following equation.

$$\nabla^2\psi = -\omega \tag{6.3}$$

The nabla operator is a differential operator. This means that ψ is differentiated two times in both the x-direction and the y-direction, resulting in $-\omega$.

By applying a Taylor series expansion to a random point with its neighbors, the second derivative at point (x,y) can be solved. The corresponding Taylor series expansion is given by 6.3.

$$\psi(x,y) = \psi(x,y) \tag{6.4}$$

$$\psi(x+1,y) = \psi(x,y) + \frac{d\psi}{dx}\Delta x + \frac{1}{2}\frac{d^2\psi}{dx^2}\Delta x^2 + O() \tag{6.5}$$

$$\psi(x-1,y) = \psi(x,y) - \frac{d\psi}{dx}\Delta x + \frac{1}{2}\frac{d^2\psi}{dx^2}\Delta x^2 + O() \tag{6.6}$$

$$\psi(x,y+1) = \psi(x,y) + \frac{d\psi}{dy}\Delta y + \frac{1}{2}\frac{d^2\psi}{dy^2}\Delta y^2 + O() \tag{6.7}$$

$$\psi(x,y-1) = \psi(x,y) - \frac{d\psi}{dy}\Delta y + \frac{1}{2}\frac{d^2\psi}{dy^2}\Delta y^2 + O() \tag{6.8}$$

For the CFD solver, a cartesian grid is used where $dx = dy = n$. When equation 6.4 is multiplied by four and is added to equation 6.5, 6.6, 6.7 and 6.8 the relationship in equation 6.9 is found.

$$\nabla^2\psi = \frac{4\psi(x,y) + \psi(x+1,y) + \psi(x-1,y) + \psi(x,y+1) + \psi(x,y-1)}{2 \cdot n^2} \tag{6.9}$$

When for example ψ needs to be computed at a 3x2 grid, as shown in figure 6.4, the Poisson solver has to solve matrix equation 6.10.

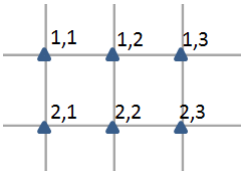


Figure 6.4: Example grid

$$A \cdot \boldsymbol{\psi} = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} \psi_{1,1} \\ \psi_{2,1} \\ \psi_{3,1} \\ \psi_{1,2} \\ \psi_{2,2} \\ \psi_{3,2} \end{bmatrix} = \begin{bmatrix} -\omega_{1,1} \\ -\omega_{2,1} \\ -\omega_{3,1} \\ -\omega_{1,2} \\ -\omega_{2,2} \\ -\omega_{3,2} \end{bmatrix} \tag{6.10}$$

One can imagine that for a higher amount of grid points, the Poisson matrix in equation 6.10 gets very large. The solver will run in a range between 128x128 to 256x256 boundary points. In the case of a 128x128 grid, there are 16.384 grid points. That means that the matrix A in equation 6.10 will be of size 16.384x16.384. This takes a very long time to solve when using direct numerical solving. For this reason it is necessary to turn to more sophisticated solvers. In the CFD solver an Algebraic Multi-Grid (AMG) solver is used. The way this AMG solver operates is beyond the scope of this report. The important thing is that it solves the Poisson matrices a lot faster than direct numerical solving methods could do.

6.4 GPU acceleration

One of the major challenges for the computational part of the program is that it needs to operate at a very high rate. To achieve a smooth image of 30 FPS the computational part has to perform calculations at a rate at least as high.

The most time consuming part of the CFD was solving three very large linear systems of $Ax = b$, to solve for ψ, ϕ and the pressure. In this system A is a poisson matrix with $(N_x \cdot N_y)^2$ numbers. This matrix size grows exponentially when using a finer grid. One advantage of the A-matrix is that most of the values are zeros, which allows for rewriting the matrix in a sparse matrix format such as a CSR. This greatly reduces the amount of matrix entries and therefore the required memory and computation time.

To solve the linear systems in a fast way, optimal use has to be made of the available computing resources in the system. One of the most powerful resources for solving large matrices is the GPU. This a very powerful computing part since it allows for a high level of parallelization due to the high amount of streaming processors.

With the NVIDIA GPU in the system, it is possible to use CUDA. CUDA enables integrating GPU acceleration into C-Code. Writing a smart algorithm for a solver is a difficult and time-consuming task. Therefore it was decided to use Paralution, a library based on CUDA with various solvers and pre-conditioners included. This library works both on computers with and without CUDA supported GPUs. This makes it very suitable for the project, where the tool is tested on different computers with different hardware specifications.

6.5 Improvements and recommendations

The most time consuming part at the moment is solving the Poisson equation. To achieve higher speed this step has to be further optimized. This could be done by a more efficient implementation of CUDA into the code. At the moment a standard Paralution library is used, but probably a higher level of optimization will be achieved when the solver was designed specifically for the GPU in the system. Running a profiler, making use of CUDA streams and reaching a higher occupancy could help to further speed up the solver. A second use of the parallelization technique could be achieved within the main loop, where the Poisson-solve step is currently run sequentially, first for ψ and then for ϕ. ψ is an input for the calculation to solve for ϕ. A good accuracy might still be achieved when using the ψ from one timestep earlier to parallelize the two poisson solves. When the Poisson solve step is not the bottleneck anymore, more speed could be achieved with more multithreading within the code. Also running the output on a separate thread could boost either framerate or gridsize.

7. Post-processing

In the Post-processing (PP) block of the airflow simulation program, calculations that are not part of the CFD block are carried out. This section provides an overview of all computations carried out in the PP block and their interdependent relations.

7.1 Overview PP block

In the PP block, a set of parameters that are of interest for the user is computed. These parameters are the pressure p, the thereon dependent forces and moments R', L', D' and M' and corresponding coefficients C_p, C_r, C_l, C_d and C_m that apply to the input object. Furthermore, locations of interesting parameters like the aerodynamic center (x_{ac}) and the center of pressure (x_{cp}) are computed.

An overview of all the computations and their dependencies is shown in figure 7.1. The required input for the PP block is obtained from two sources. First the PREP block provides information about the boundary grid points, like the location Γ, normal vectors n and arc length ds. Furthermore the input parameters of the user are given (e.g. angle of attack, free stream properties, viscosity, etc.). Secondly, the CFD program provides a velocity and vorticity field.

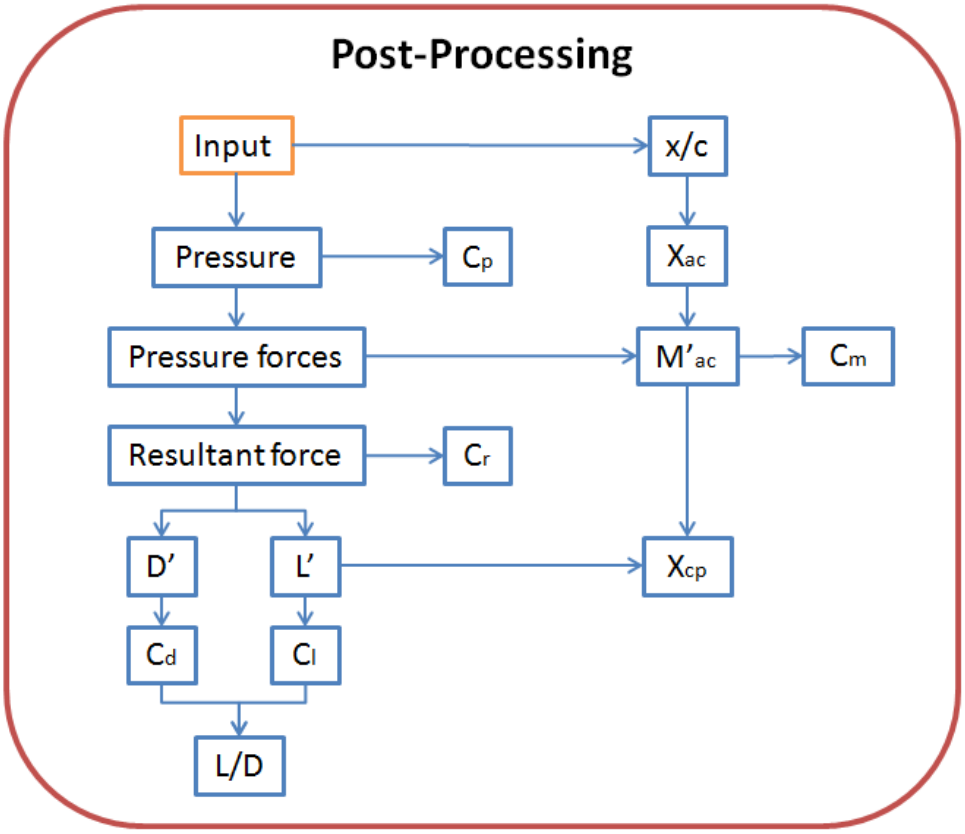


Figure 7.1: Overview of computations in PP block for symposium tool

In figure 7.1 can be seen that no boundary layer calculations are performed. In the program the boundary layer is not resolved, partly because of the coarse grid resolution. To simulate the presence of a boundary layer, vorticity is added at the interface boundary. This vorticity is a good model for the vorticity that would be added by a boundary layer. It can therefore be assumed that even without taking the viscous forces into account, the results of the velocity computations are modeling

the physics of having a physical boundary in a flow field. It should be noted that for the pressure computations viscosity is not neglected.

In table 7.1 the required input and output for all computation blocks of figure 7.1 are listed. The table is made in chronological order and the definition of the parameters can be found in the list of symbols. The subscript i, j indicates that the parameter is computed for every grid point.

Table 7.1: Required input and output for all computations

Process	Input	Output
Compute P	$u_{i,j}$, BC	$p_{i,j}$
Compute C_p	$p_{i,j}, p_\infty, u_\infty, \rho_\infty$	C_p
Compute F	$p_{i,j}, ds_{i,j}$, $n_{i,j}$	$F_{i,j}$
Compute R	$F_{i,j}$	$R, (N, T)$
Compute C_r	$R, p_\infty, u_\infty, \rho_\infty$	C_r
Compute L'	R, α	L'
Compute C_l	$L', p_\infty, u_\infty, \rho_\infty, c$	C_l
Compute D'	R, α	D'
Compute C_d	$D', p_\infty, u_\infty, \rho_\infty, c$	C_d
Compute L/D	L', D'	L/D
Estimate x_{ac}	c	x_{ac}
Compute M'_{ac}	$F_{i,j}, \Gamma_{i,j}, x_{ac}$	M'_{ac}
Compute $C_{m_{ac}}$	$M'_{ac}, p_\infty, u_\infty, \rho_\infty, c$	$C_{m_{ac}}$
Compute x_{cp}	$L', M'_{ac_{c/4}}$	x_{cp}

7.2 Pressure computation

From figure 7.1 and table 7.1 it is clear that obtaining the pressure has the highest priority, since almost all other computations depend on this variable. It is estimated that computing the pressure is also the most time consuming part, while the dependent forces and coefficients are rather easy to obtain. A first pressure solver, that was provided by Richard Dwight, was translated to C++ and integrated in the program. An explanation on how the pressure is computed is given in this section.

7.2.1 Pressure Poisson equation

The pressure at each grid point can be computed using the pressure Poisson equation (PPE) 7.3. The PPE is derived by taking the divergence of the momentum equation 7.1 (where the body forces are neglected) and using the incompressibility constraint 7.2.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} \tag{7.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{7.2}$$

$$\nabla^2 p = -f \tag{7.3}$$

Where the term f simply represents the right-hand side (RHS) of the equation. In a two dimensional situation, having $\mathbf{u}(u, v)$, the PPE can be written as equation 7.4.

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\rho \left[\left(\frac{\partial u}{\partial x} \right)^2 + 2 \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y} \right)^2 \right] + \rho \cdot \nu \left[\frac{\partial}{\partial x} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + \frac{\partial}{\partial y} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \right] \tag{7.4}$$

The complete RHS can be computed using the output velocities of the CFD block. In order to solve equation 7.4, the second derivatives of the pressure are discretized using a central difference scheme 7.5. It can be rewritten to 7.6 when assuming a unit mesh spacing. The result is a linear set of equations that can be written as $A\mathbf{p} = -\mathbf{f}$.

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \frac{p_{i-1,j} - 2p_{i,j} + p_{i+1,j}}{\Delta x^2} + \frac{p_{i,j-1} - 2p_{i,j} + p_{i,j+1}}{\Delta y^2} = RHS \tag{7.5}$$

$$4p_{i,j} - p_{i-1,j} - p_{i+1,j} - p_{i,j-1} - p_{i,j+1} = RHS \tag{7.6}$$

7.2.2 Boundary conditions

Equation 7.3 can be solved in the same way as equation 6.1 and 6.2 were solved for ψ and ϕ in the CFD block. However, before solving the correct boundary conditions need to be incorporated to have a physically meaningful set of equations.

Outer grid boundaries

At the outer boundary of the domain, Neumann boundary condition (BC) for the pressure will be used for the horizontal sides. This Neumann BCs set the pressure gradient at the upper and lower boundary equal to zero, implying the far field pressure is constant. At the inflow and outflow a Dirichlet boundary condition will be used, setting the pressure on these boundaries equal to zero such that the computed pressure in the domain can be visualized as relatively positive or negative to the far field pressure. The use of BCs is clarified in figure 7.2. The applied Neumann and Dirichlet BCs are shown in the figure as well.

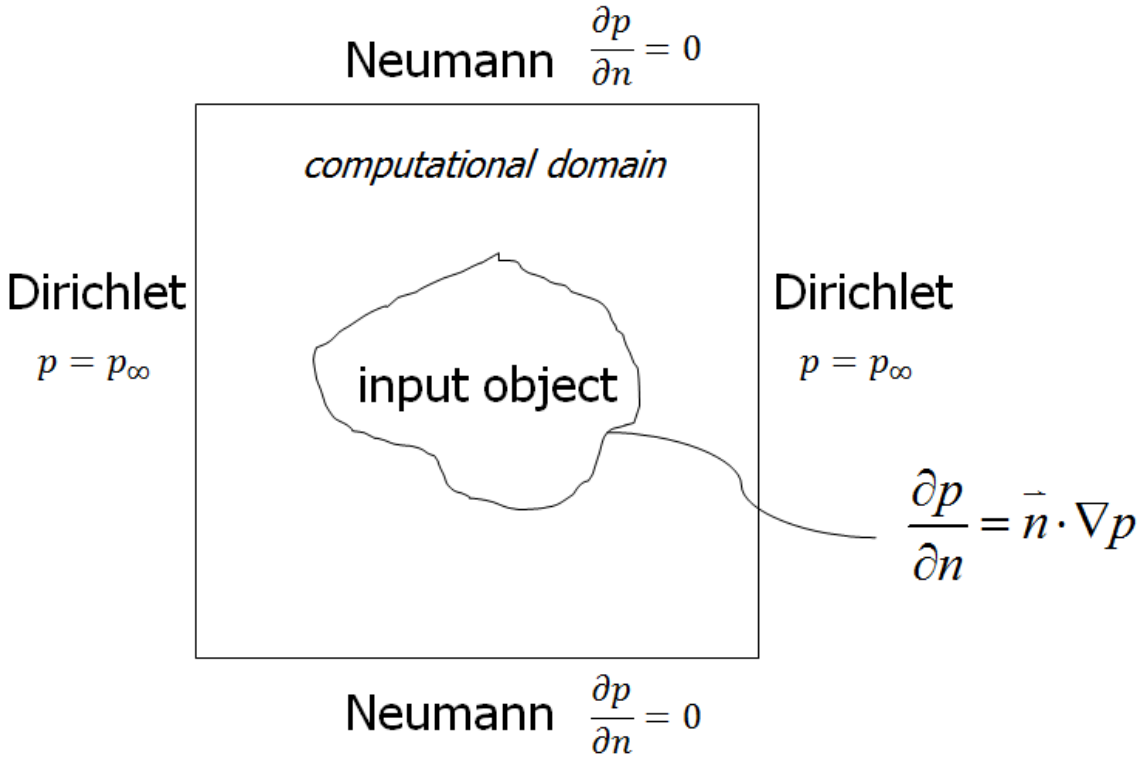


Figure 7.2: Boundary conditions for computation of pressure

To incorporate these boundary conditions on the outside boundaries, the system of linear equations should be adapted [5]. In case of a Dirichlet boundary condition, the complete column of grid points on the boundary is sliced off. In case of a Neumann boundary condition, the condition $\frac{\partial p}{\partial n} = 0$ is

discretized according to equation 7.7 (for the lower boundary the third term will be $p_{i,j+1}$) and can be substituted in equation 7.6.

$$3p_{i,j} - p_{i+1,j} - p_{i,j-1} - p_{i-1,j} = RHS \tag{7.7}$$

Internal object interface boundaries

Since the velocities inside the interface of the input object are set to zero in the CFD part of the program, additional boundary conditions are required at the interface with the immersed boundary of the input object to ensure no through-flow. To achieve this, a Neumann BC with the term $\frac{\partial p}{\partial n}$ is applied to the immersed boundary of the inner object, as indicated in figure 7.2. This BC however, cannot be implemented by using discretization schemes similar to those that are applied to the outer boundaries. This is because the interface with the object is not defined by the grid exactly, but is immersed in the grid. To apply the BC, an immersed boundary method (IBM) has to be implemented. This IBM concerns forcing terms g that are added to the RHS of equation 7.4, resulting in equation 7.8.

$$\nabla^2 p = -f + g \tag{7.8}$$

Solving this equation directly would require solving for a nonlinear relation between g and the BC $\frac{\partial p}{\partial n}$. Instead the pressure is defined as a sum of p_f and p_g (equation 7.9), which are pressure Poisson terms based on respectively the RHS as defined in equation 7.10 and a forcing term g. Equation 7.10 is just a different notation of equation 7.4. Next the term $\frac{\partial p}{\partial n}$ is defined by equation 7.12. Subsequently, the forcing term g is evaluated such that equation 7.12 holds on the immersed boundary Γ, using the L-matrix method that is used in the CFD block. Then g is used to solve for p_g, according to equation 7.11. Finally, the pressure is computed by summing up p_f and p_g.

$$p = p_f + p_g \tag{7.9}$$

$$\nabla^2 p_f = \nabla \left(\nu \nabla^2 \mathbf{u} - \mathbf{u} \cdot \nabla \mathbf{u} \right) \tag{7.10}$$

$$\nabla^2 p_g = g \tag{7.11}$$

$$\frac{\partial p}{\partial n} = \mathbf{n} \cdot \nabla p = \nu \nabla^2 (u \cdot n) - \mathbf{u} \nabla (\mathbf{u} \cdot \mathbf{n}) \tag{7.12}$$

7.2.3 Pressure coefficient

When the pressure is known, the pressure coefficient can be calculated by using the conventional equation 7.13.

$$C_p = \frac{p - p_\infty}{q_\infty} \tag{7.13}$$

7.3 Calculation of forces

Once the pressure distribution is known, the pressure forces on the object can be calculated. As was shown in figure 7.1, all remaining variables depend on these forces which in turn depend on the pressure distribution. The pressure p in each interface boundary point $\Gamma_{i,j}$ will be multiplied with the corresponding arc length ds and unit normal vector to obtain both the magnitude and direction of the force working on that point, see figure 7.3.

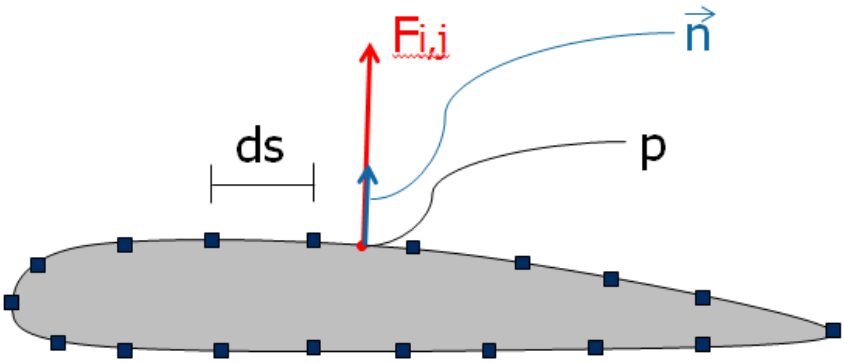


Figure 7.3: Pressure force segments

7.4 Resultant force, lift and drag

The resultant force is composed from an x- and y-component, each calculated by summing up the x- and y- components of the separate forces. The 2D lift force L' and drag force D' are then obtained with trigonometry from the normal and tangential component of this resultant force, being C_N and C_T. The orientation of these forces are depicted in figure 7.4. The 2D lift and drag coefficient, respectively C_d and C_l, are then computed by the standard equations, using dynamic pressure q_∞ and chord length c. The lift-drag ratio can be computed by either using L' and D', or C_l and C_d. The non-dimensional resultant force C_R can be calculated in the same fashion as C_d and C_l.

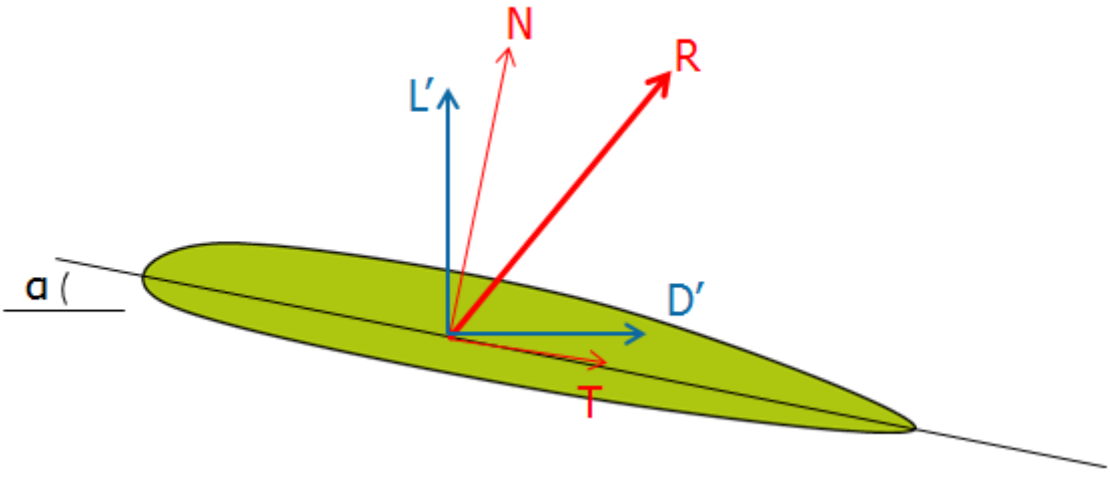


Figure 7.4: Computation of 2D lift L' and drag D'

7.5 Aerodynamic center

Before M'_{ac} can be calculated, the location of the aerodynamic center (ac) needs to be computed. This computation would require to run the CFD program for at least two angles of attack. Although this could take place in the background of the program, it would cost significant computation time. To save this time it was decided to estimate the position of the ac to be at $\frac{c}{4}$ on the mean aerodynamic chord (MAC). This is a valid assumption for symmetric airfoils in subsonic flight, as well as for thin airfoils. For cambered airfoils the quarter-chord position is only an approximation. It is important to

note that since the input to the tool can be any arbitrary shape, the assumption for the location of the aerodynamic center will not always be valid.

7.6 Moment about the aerodynamic center

The moment about the aerodynamic center M'_{ac} can be calculated according to equation 7.14 using the forces working on the boundary points Γ, as calculated in section 7.3, and the distance between the corresponding grid points and x_{ac}.

$$M_{ac} = \Sigma \left[F_{\Gamma(i,j)_x} \cdot y_{i,j} + F_{\Gamma(i,j)_y} \cdot (x_{\Gamma(i,j)} - x_{ac} \right] \tag{7.14}$$

Afterwards, the moment coefficient C_m can be computed according to equation 7.15.

$$C_m = \frac{M'_{ac}}{q_\infty \cdot c^2} \tag{7.15}$$

7.7 Center of pressure

Using the assumption for the location of the aerodynamic center, the center of pressure can be computed according to equation 7.16 [9].

$$x_{cp} = \frac{c}{4} - \frac{M_{ac_{c/4}}}{L'} \tag{7.16}$$

8. Graphical output

In this chapter the graphical output of the program is discussed, underlying theories and the visualization algorithms are explained. First the plotting is dealt with, after which the pressure plot and generation of particles are discussed.

8.1 Plotting

The plotting is done in a glWidget, which is a widget setup by QT and constructs a pipeline to OpenGL (drives the graphics card). Inside this widget, the language of OpenGL is used to display objects graphically.

8.1.1 Vector field

To visualize the velocity field that is computed by the CFD solver, a vector field is constructed, which is a grid of either vectors or lines. The main advantage of a vector field over a gradient field is that it can indicate orientation. This vector field is based on the velocity grid produced by the CFD solver. In the case of the arrows, each arrow is defined by six points, as depicted in figure 8.1. In the case of a line, it consists of two points, corresponding with the start and end of the arrow. The orientation of the arrows is defined by an angle, which depends on the orientation of the velocity θ. The length of the rectangular part of the arrow depends on the magnitude of the velocity. The size of the object depends on the amount of arrows set to be displayed. The color of the arrow depends on either the magnitude of the velocity, the vorticity, or the pressure.

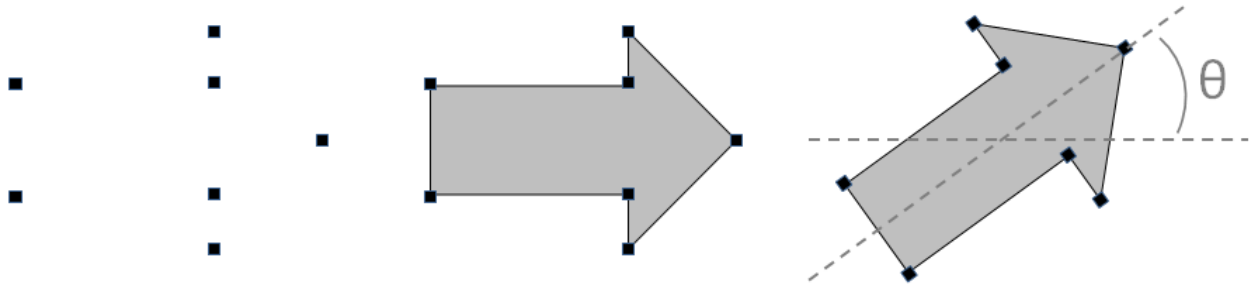


Figure 8.1: Construction of velocity field arrows

Displaying arrows has the advantage that the direction is directly clear. However, a large amount of arrows (to see on a finer grid what happens), results in a blurry vector field and therefore gives less insight. For this reason, lines are introduced, such that more is visible on a finer grid.

8.1.2 Gradient field

To visualize any field, such as pressure, velocity or vorticity, a gradient field is used. Basically every gridpoint is assigned a color based on its value in the grid. Then the color is interpolated by OpenGL between all the points, giving a clear, accurate, and fluent field of colors.

8.1.3 Clearing Values

Inside of the object, all values must be zero. In order to display this correctly, for both the gradient field and the vector field, a matrix is set up to indicate which arrows or gradients must not be displayed. The enclosed points (taken from the pre-processing block) is used to do this.

8.1.4 Overlay Object

To be able to see where exactly the object is, it is good to display it on top of the vector field and gradient field. A line is drawn in the way the user has, using the exact boundary.

8.1.5 Resize grid

The number of grid points for the vectors is smaller than the amount of computational grid points. Also, for a smoother picture, the grid of the gradient field is finer than the computational grid. For these reasons, it is necessary to be able to resize a grid, while maintaining the information. This section explains the algorithms that are used for resizing the grid.

Increase grid size
To increase the grid size (from coarse to fine), the bilinear interpolation technique is used, see equation 8.1. This technique is used to find the value at a new point P, based on the weighted values of the surrounding points, say $Q_{i,j}$. Each surrounding coarser gridpoint conributes to point P, based on the area across from the point. The idea is visualised in figure 8.2.

$$f(P) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \cdot (f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1)) \tag{8.1}$$

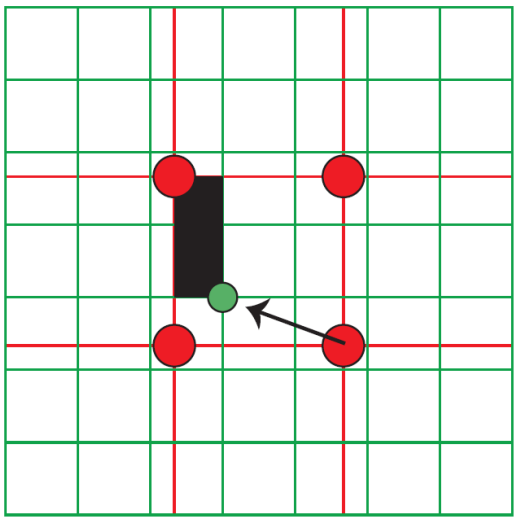


Figure 8.2: Calculation grid values for increased resolution

Decrease grid size
When decreasing the grid size (from fine to coarse), the information in the 'old' grid must be processed to create the 'new' grid. Simply deleting points would not give accurate results. The surrounding points give information for the new grid. The amount of points relevant to create the new point is determined by comparing the distance of the old matrix grid points to the different surrounding new grid points. The location of the new grid points is known relative to the old points. This means that the maximum distance a relevant old point has to one of the new points is the square root of the

ratios in x and y direction. This limits the amount of relevant points and therefore the amount of calculations. The surrounding points are weighted based on their distance to the new point. A closer located point has a smaller distance and the contribution is therefore higher than a point at a greater distance having the same value.

The points located at the corners of the old and new grid always overlap. Overlap can also occur at other locations in the grid. The overlapping points would have zero distance between the old and new points. This would give erroneous results and to take this overlap into account, the assumption is made that all surrounding points together have the same contribution to the new point as the overlap point. The idea is visualised in figure 8.3.

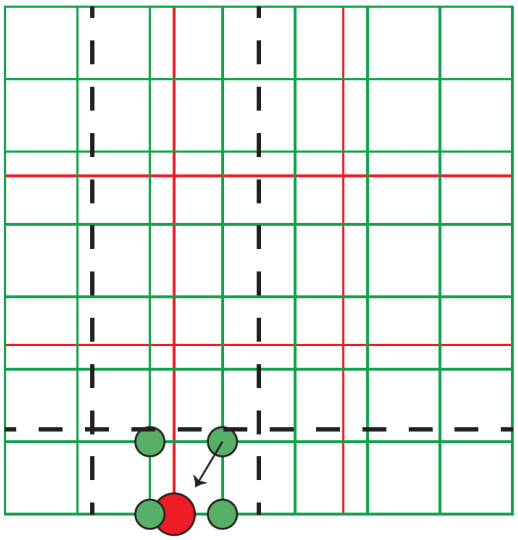


Figure 8.3: Calculation grid values for decreasing resolution

8.1.6 Architecture

The architecture of the plotting part of the program is represented in figure 8.4. On the right hand side the profile is imported from the pre-processing block. This is used to make the clearing matrices and to make the overlay object. This proces is only performed once, every time a new item is drawn.

On the left hand side, the loop is represented that the program goes through at a set time rate (e.g. 33ms). First, the solver is run once, from which the information is extracted. The velocity (which is in the form of x- and y- directional components) is converted into theta and magnitude format. These matrices, together with the vorticity and the pressure, are resized according to the vector grid size or gradient grid size. Then these matrices, together with the clearing matrices, and information about the maxima and minima of the matrices, are used to make the vectors and the gradient field. Finally eveything is painted by OpenGL and the loop is restarted.

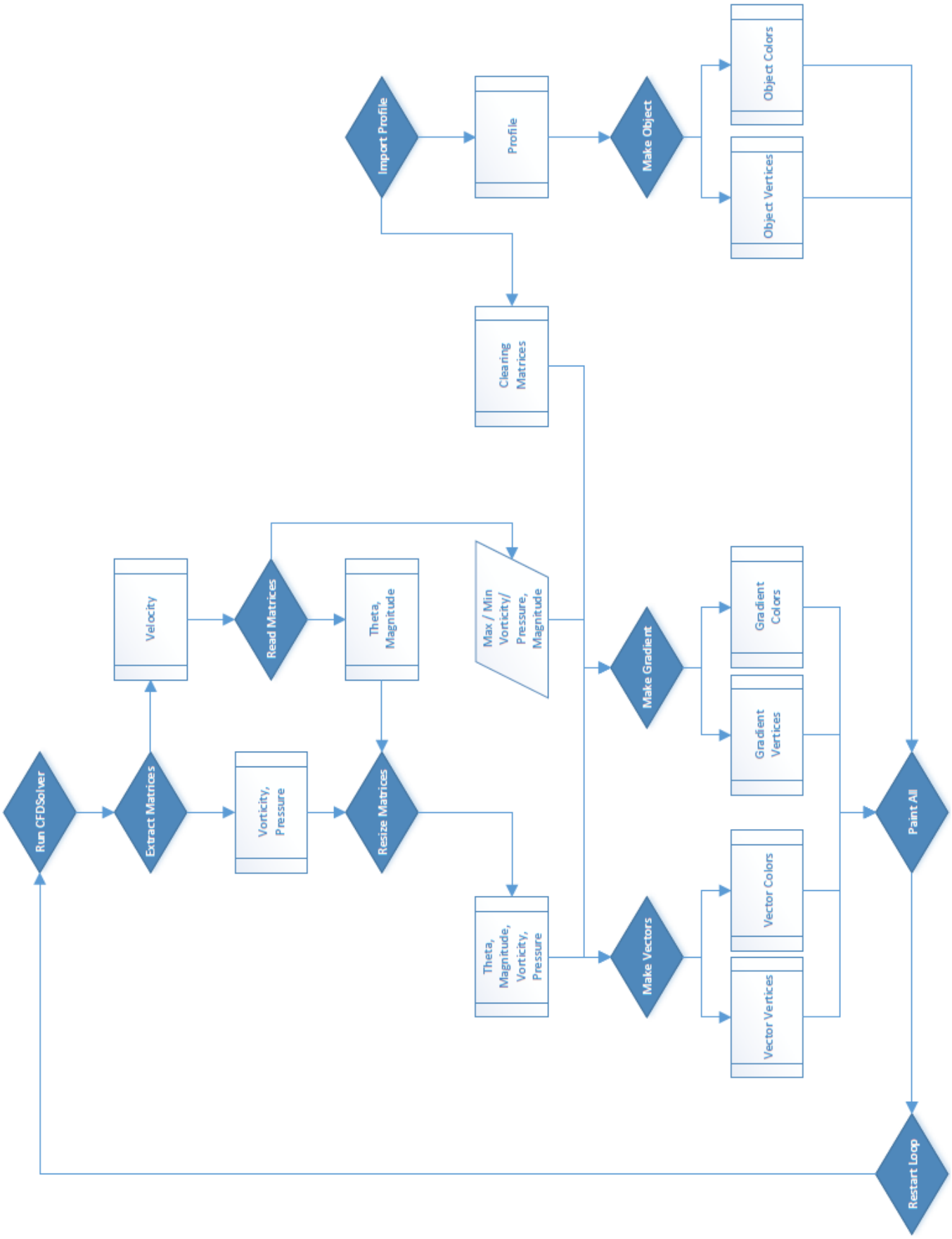


Figure 8.4: Plotting architecture

8.2 Pressure plot

The pressure distribution over the airfoil chord length is presented with a C_p-x/c plot. The plot must be visible in the graphical user interface, therefore a plotting library of Qt is used. This library is called QCustomPlot. The choice for this library was straight forward, since it is well documented and relatively easy to use. The pressure coefficients calculated in the post-processing block need to be converted to coordinates that are recognised by QCustomPlot. QCustomPlot automatically orders coordinates with increasing value. This gives bad results for the C_p-x/c plots, since most x/c coordinates have multiple C_p-coordinates. This was solved by using QCPCurve, which allows to use arbitrary ordered coordinates to be plotted in the order they occur in. It is also possible to plot multiple C_p-coordinates for the same x/c coordinate without getting a strange connection such as happened when using the regular plotting code. To keep the plots readable in the user interface, the axes are scaled automatically based on the coordinates given. Another option, which was relatively simple to implement is the possibility to move the ranges of the axes of the graph. While including this, the options to select the graph and to zoom were also included. These options are useful when comparing several plots.

8.3 Particles

Visualizing the translating motion of a particle in the simulated flow can give good insight in the flow around an object. When a flow is unstable the streamlines are different for each timestep. Therefore a particle which is dropped in the flow field will not follow a complete streamline but a pathline, which is the result of the unstable flow.

The movement of a particle is based on the underlying vector field. Once a particle is dropped in the flow it has a certain position with respect to the velocity vector field. The velocity is determined by using bilinear interpolation. This is necessary because the resolution of the available positions is higher than the resolution of the available velocity vector field. At timestep t+dt, the new position equals the previous position plus the change in position based on the velocity at time t. Performing this procedure repeatedly will make a particle move on the screen.

To visualize this for more than one particle an emitter type particle system was designed. This is particularly interesting for visualizing streaklines. A particle system is a collection of particles that are controlled as a group based on the same underlying physics. An emitter is a type of particle system that emits particles from a centralized position. The choice was made to emit at a fixed rate, i.e. particles are dropped at a fixed interval in the flow. Since the location of the emitter can be chosen arbitrarily, it can be positioned anywhere in the flow field. Another beautiful aspect of such a system is that it can be created multiple times in the same flow field, because they do not interfere with one another.

The structure of the system was designed in such a way that it runs for an indefinite period of time. Internal management of the emitted particles is designed such that it allows for this. One of the key ways of doing this is to create a predefined number of particles per system and keep reusing these. If particles are emitted at a fixed rate, then the number of available particles would run out. This issue is solved by assigning particles a lifetime. When the lifetime of a particle is up it will be reset and available to be emitted again. As long as the lifetime is shorter than the time it takes to reach the final element of a system, it can keep emitting particles indefinitely.

Due to this, a particle system consists of a row of particles that are either emitted or reset (they are reset by default). Only the emitted particles will need to be updated for their positions and velocities. A simple loop would suffice to update them. The loop would run from the first emitted element of the row to the last element of the row that has been emitted.

There is however another issue. Since the particles base their position and velocity on bilinear interpolation, these values are not defined outside the grid positions. Therefore a particle is also reset when it is out of the screen. Note that a particle can be reset due to it being out of bounds before its lifetime is up.

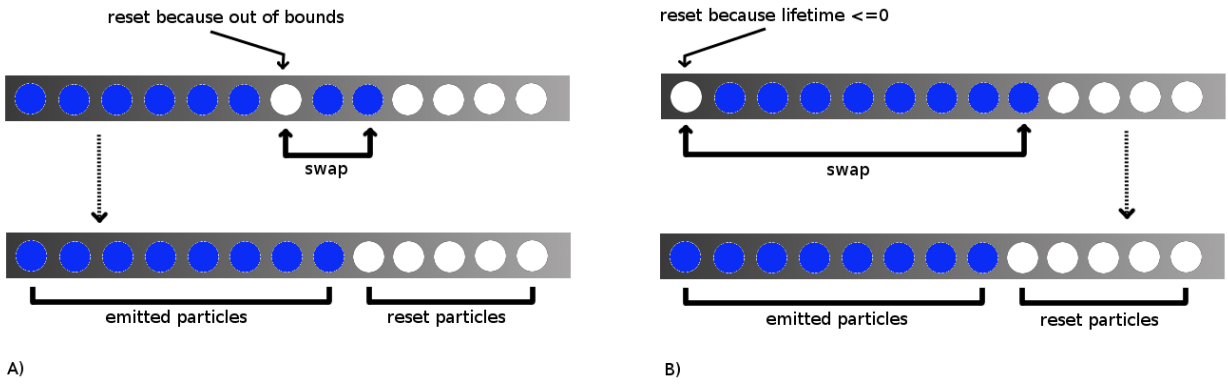


Figure 8.5: Efficient ordering of emitted and reset particles

Since only the emitted particles need to be updated, the loop should only run through the emitted particles. By swapping the rightmost emitted particle with the particle which has been reset before its lifetime is up, the emitted and reset particles are kept separated and easy to loop through, see figure 8.5A. Swapping the rightmost emitted particle with a particle which has zero lifetime left assures that the active particles are kept on the left side of the row, see figure 8.5B. This is beneficial for the loop that runs through the emitted particles.

9. Hardware

This chapter clarifies the physical appearance of the tool. First the acquired hardware components are listed, from which a cost analysis is made. Then an overview of the relations between the hardware components is given in a hardware block diagram. Furthermore the required electrical equipment is explained. The physical external appearance of the tool is discussed in section 9.5. At last, an analysis of the hardware performance is given.

9.1 Hardware components

In order for the tool to be fast, it requires strong hardware. This however should be within the limitations of the given budget. It should also be easily reproduced, thus demanding the use of off-the-shelf components. Because no real estimates of the required computing power are available, the chosen amount of computing power for CPU and GPU are based on what was achievable within the budget. Any overestimations in computing power can always be used to achieve a higher gridsize and therefore a higher accuracy. In future versions, this can be scaled to the wishes of the client, depending on the preference for an inexpensive or accurate system.

CPU [Intel i7 4770K]
The Intel i7-4770K was the fastest CPU available within the budget.

GPU [Palit GeForce GTX 770 Jetstream 4GB]
For the GPU, an important requirement was that it was an NVIDIA GPU because only NVIDIA cards support CUDA. Other important parameters of the card were the available memory and the speed. Since the solver mainly uses floating point numbers, gaming cards offer the best value. Like with the CPU, the maximum amount of performance within the budget was chosen.

RAM [Crucial Ballistix Tactical 2 x 4 GB 1600 MHz DDR3]
For the memory, 8 GB was thought to be sufficient. The cheapest ram that runs on 1600 MHz was chosen.

Motherboard [Gigabyte Z87-HD3]
For the motherboard the requirements were that it supported the chosen CPU and that it offered room for expansion in case an additional GPU was needed. The Gigabyte Z87-HD3 supports MultiGPU capabilities and can be used for overclocking. The motherboard also has 4 slots for DDR3 and thus allows for expansion.

SSD [Samsung 840 Evo 120GB]
A smaller and faster SSD was chosen, compared to equally priced HDD, the cost increase was marginal. The tool does not need a lot of storage, but having high write and read speeds could help to add speed to some parts of the program in the future.

PSU [XFX Pro 550W]
The PSU had to be able to deliver enough power; power consumption of the parts was checked using a power supply calculator. The computer system power requirement was found to be 383W. Although the 550W PSU is larger than needed, the added safety of a more capable unit was deemed acceptable in exchange for a marginal cost increase.

Case [Coolermaster Force 500]
As simple a case possible to fit in all the components was sufficient. The case supports the ATX motherboard format and has space for the big GPU. It also allows for extra fans, in the event that the case overheats.

Interactive projector [Epson EB-485Wi]
The projector was paramount to the interactivity of the tool and of the few that offer such capability, the Epson offered the best performance given the budget restrictions. Performance specifications such as a 3100 lumen bulb and 1280x800 resolution, the projector was powerful enough to achieve the intended purpose.

9.2 Budget breakdown

In this section the expenses for acquiring the hardware components required to build a prototype of Eddy are analyzed. First a decomposition of the PC costs is given and afterwards the relative cost of the interactive projector is shown.

9.2.1 PC Acquisition

PC costs were kept slightly under budget to allow for the acquisition of the interactive projector. A proportional breakdown of the costs can be seen in figure 9.1. Note that the largest cost allocation is in the GPU which plays a significant role in the high-speeds achieved by the solver.

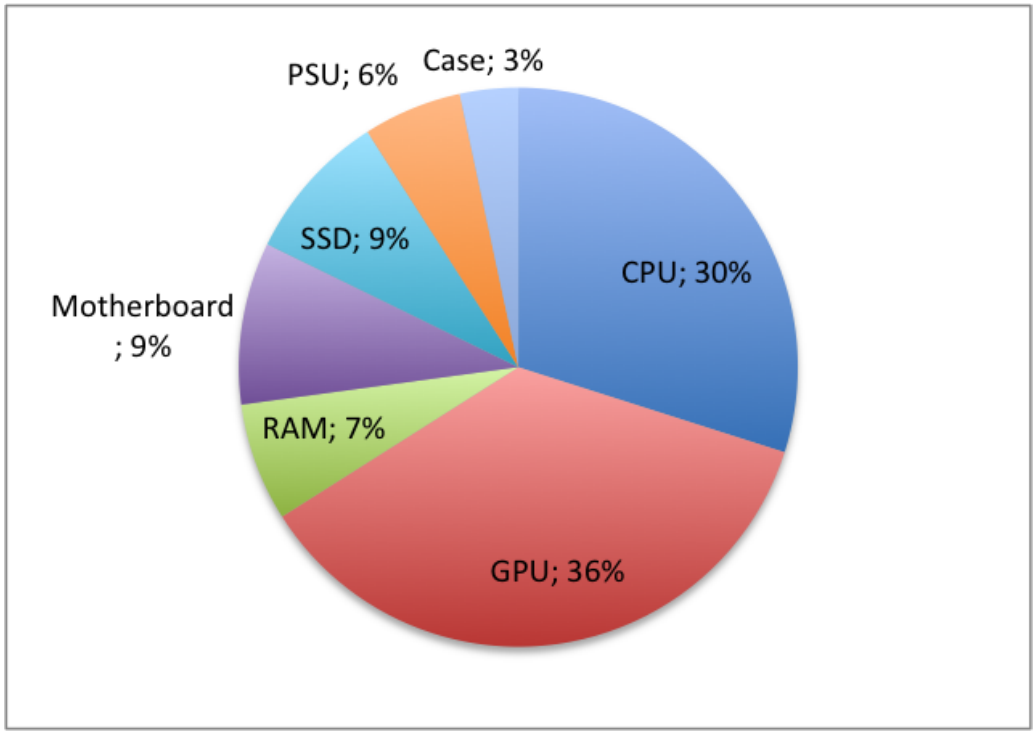


Figure 9.1: A cost breakdown of the PC hardware componentry

9.2.2 Interactive Projector Acquisition

The Epson EB485-Wi interactive projector constituted the largest cost for the project; costing more the PC in its entirety. Interactivity of the tool relies heavily upon the capabilities and feature set of the projector and therefore this cost was justified. This cost breakdown can be seen in figure 9.2.

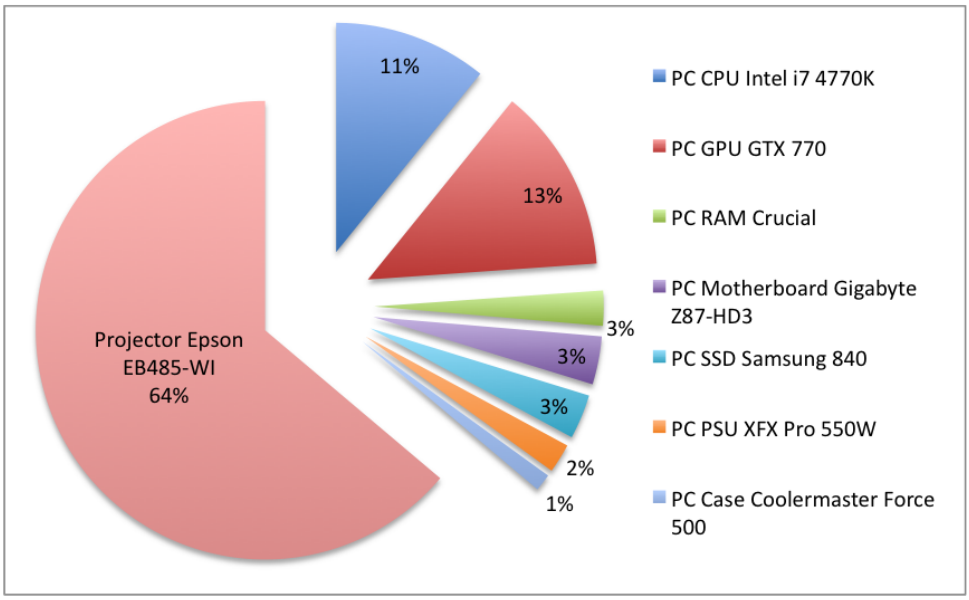


Figure 9.2: Total cost of all project components relative to total cost

9.3 Hardware block diagram

In figure 9.3 a basic overview of the relations between the hardware components is given. The left block represents the computer that performs all the computations. The computer is connected to the right block. This is the package of the interactive beamer that is used for the human-machine interaction. The specifications of the components were mentioned in section 9.1.

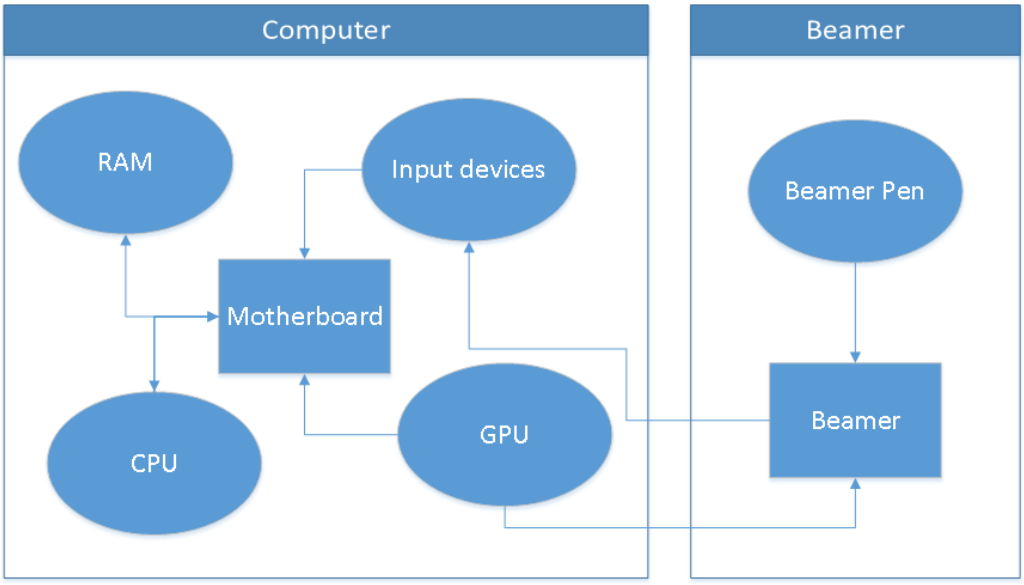


Figure 9.3: Diagram containing relations between hardware components of tool

9.4 Electrical block diagram

The electrical equipment for the product is very obvious, therefore no complete block diagram is made. To get the tool working, it is necessary to connect both the PC and the beamer to the electrical grid via a plug socket. Additionally, each pen of the beamer requires a standard AA-battery.

9.5 Configuration/layout

In section 3.3 and 9.3 the internal program configuration was explained. This section presents the physical external appearance of the interactive airflow simulation tool. As can be seen in figure 9.3, the two main components are the computer and the interactive beamer that projects the user interface and result.

Figure 9.4 shows the configuration of the tool. In this configuration the interactive function of the tool is emphasized just as well as the possible educational purposes. In the figure the interactive beamer is placed on the surface that is used to project the GUI. This position can be adjusted to many different surfaces for optimum accessibility.

The computer can be placed at any position depending on its environment where it will be used. It can be simply placed on the floor or one complete integrated package could be delivered where the PC is placed on top of the beamer in a single box. This does require a stronger mounting device.

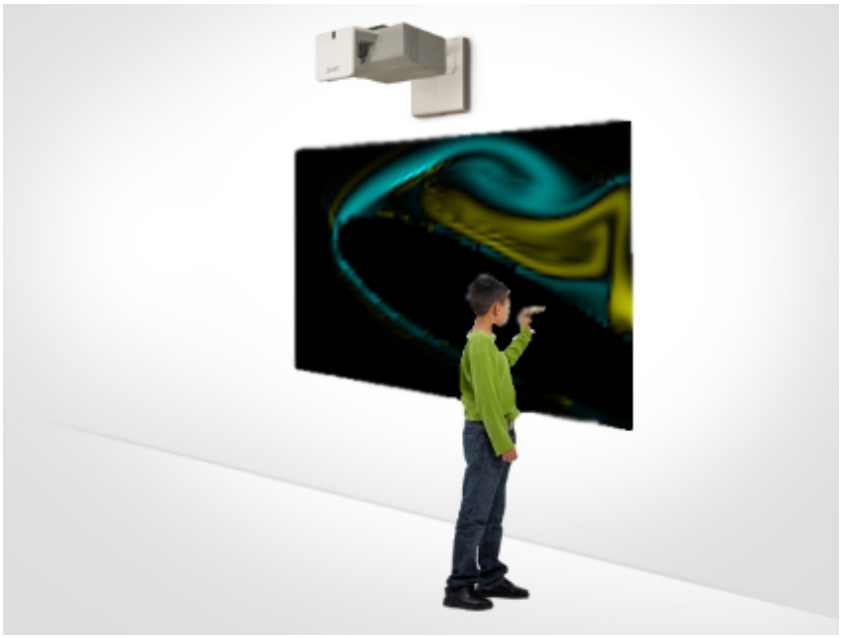


Figure 9.4: Configuration of the tool

9.6 Hardware Performance

The hardware that had been purchased had to be capable of meeting the minimum performance standards set by the team. As explained, any performance that was considered to be above and beyond that which was needed could be used to create larger grid sizes. During early development of the tool the significant lack of speed was only due to insufficient use of the capabilities of the PC. Over the course of the project the speed of the tool gradually increased; different alterations to the software led to speed-ups of various significance. These events are visualized in figure 9.5, where the frames per second (FPS) are displayed versus the dates of the product development.

In figure 9.5 there are two notable increases in FPS over the course of the project. These jumps are most evident around the beginning of December and the middle of January. In December the more computationally expensive portions of the code were processed on the GPU. In January, the AMG solver was given a proper preconditioner and the creation of the Poisson matrix and solver-build hierarchy were taken out of the running loop of the program.

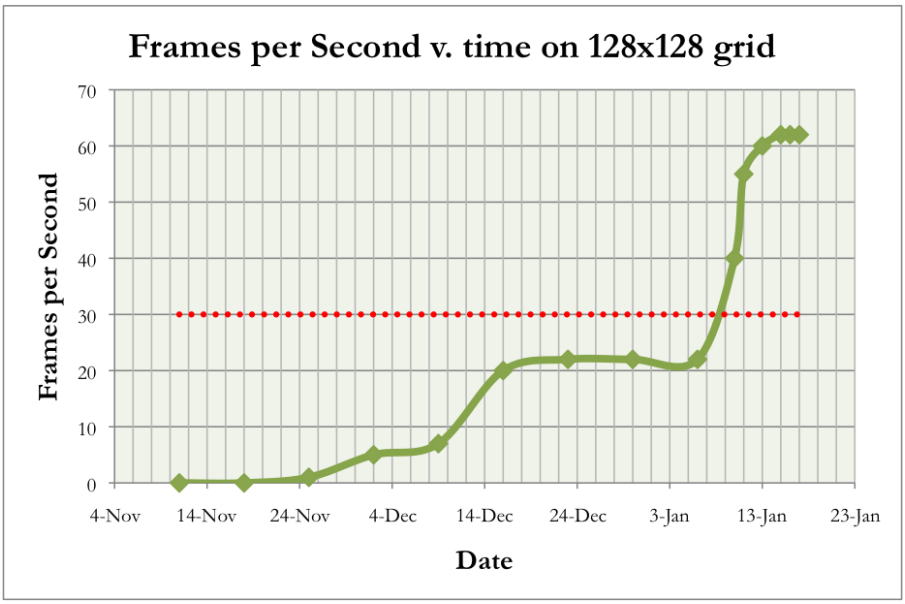


Figure 9.5: The increase in FPS over time

For comparative purposes, a Phoronix GPU test has been performed on the PC. This will allow future developers and those attempting to use the software on their own systems to benchmark their hardware against what was used herein. The results of the test can be seen in figure 9.6.

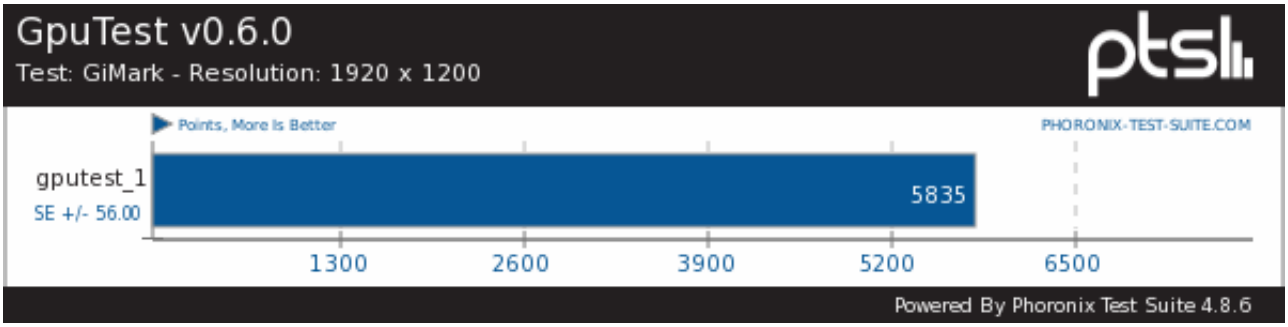


Figure 9.6: Results of the Phoronix GPU test

10. Final prototype

This chapter presents all the analyses that have been done on the final product that resulted from the design and development process of Eddy simulation tool. First, the verification and validation procedures are discussed, followed by a sensitivity analysis. After that the operations and logistics concept is explained and finally a compliance matrix is presented, where is discussed whether the design requirements are met.

10.1 Verification & validation procedures

To develop a properly working prototype, there is need for verification and validation. The general approach for both verification and validation is specified in this section. The software program of this prototype can be split up into Pre-processing, CFD solver, Post-processing, Graphical output and GUI. Validation and verification will be performed on these individual parts and on the system as a whole.

10.1.1 Verification

In the next two sections the verification procedures will be explained for the individual parts and for the complete system as a whole.

Individual parts

Each component will need to be verified individually. For the input this will be done by simply checking if the stored input corresponds to the input specified by the user. The CFD solver will have its intermediate steps verified by comparing them with the algebraic versions. Also some known input output data will be used if available to verify that the CFD solver is working as intended. For the PP block, the verification will be performed in a similar manner as to the CFD solver. For the graphical output, verification is not that straightforward. To make sure that it works properly, some trivial input can be checked to have its expected output. If this does not give any unexpected results, some more extreme cases can be verified in addition. The GUI will be verified by checking if all the functionality of the tabs and buttons is as intended.

Complete system

The system as a whole needs to be verified because unforeseen intermediate errors can lead to an increased error at the end of the program. This will be done by running the entire system and verifying that the output is as intended.

10.1.2 Validation

Because this rapid wing prototyping tool is designed to simulate reality, it will need to be validated to see how it compares to what it should be simulating. More specifically, the values of pressure, velocity and vorticity surrounding the airfoil need to be compared to real life airfoils. To validate the obtained values of velocity, pressure and vorticity, standard NACA-airfoils will be used as input after which the C_p output will be compared to C_p-plots obtained by running XFoil or Javafoil on the same airfoils. Program output data can also be compared to data of real life tested airfoils from the book [1]. It should also be validated if the requirements upon which the verification is performed still coincide with what the customer really wants.

10.2 Sensitivity analysis

The degree of feasibility of the final prototype strongly depends on the performance of the hardware system on one hand, and of the code on the other. Sensitivity of the two components to changes will be discussed in this section.

10.2.1 Hardware system

Changes in the hardware system of the product after the conceptual design phase will affect decisions based on this concept, such as chosen programming language, means of input and output, computing power and in turn computational speed. Since those parameters froze after the conceptual design phase, and have been tested positively to perform as desired in combination with all program components, a fully working prototype is feasible. The prototype will however be sensitive to hardware defects. Such defects could be inconvenient, but would not be fatal since standard laptops could be used as a back-up during repairing or buying and delivery of replacement components. Defects concerning the interactive beamer could be solved by repairing it as well, but could turn out to be fatal for presentation of the final prototype, since repairing could take a lot of time relatively.

10.2.2 Code

Since the code behind the tool is being adjusted and changed continuously during development of the program, the success of the final prototype is dependent on the integration of the codes of the different program blocks. Overwriting working codes and incorrect integration of the different codes could result in errors. The tool is therefore very sensitive to changes in the code.

10.3 Technical risk assessment

In this section the technical risks that can occur during the project are evaluated. The main source of risk comes from failures in the hardware that will be used. Another source of risk is the implementation of the software. Since for each program block different software is used, compatibility between those programs is very important.

In the list below every item is indicated with a number. This number has a place in the risk map which indicates the probability and the severity of the specific risk, see table 10.1.

- Possible problems with the interactive beamer:
  1. The pen which serves as a mouse pointer might have problems with its left- and right button
  2. The discrepancy between the actual position of the pen and the detected position is greater than 1cm
  3. The distance in depth between the location of the pen and where the beamer thinks the pen is, is >1cm
  4. The readability of the beamers projection is too low (due to brightness) with the lights turned on at the working place
  5. The suspension of the beamer might not be strong enough to suspend the beamer to the ceiling
  6. The beamer might overheat more than once an hour
  7. The delivery takes more than one week
  8. The tools for mounting the interactive beamer are not available
  9. Parts for mounting the interactive beamer get lost
  10. Compatibility of the interactive beamer with Linux differs from that of Windows

- Possible problems with the computer:
  11. The computer does not have enough computing capacity to display the flow in real-time
  12. A critical component (e.g. GPU) breaks down during the project
  13. RAM is not sufficient to cope with the big sparse matrices
  14. The CPU overheats during the use of the program
- Software Problems:
  15. The user puts in a shape that is not compatible with the device
  16. Implementation of packages will not work on the computer
- Possible problems at the working place
  17. The power shuts down at the working place for more than two hours

Table 10.1: Risk map

probability ↑	high	0	7	0	0
	medium	8,17	2,4,14,15	3,10,11,13	0
	low	0	5,9	1,6,16	12
		negligible	marginal	critical	catastrophic
		severity →			

10.3.1 Plan to mitigate risks

Most risks are difficult to mitigate, because the risks happen beyond the control of the group. It is however possible to schedule extra time to deal with e.g. unexpected bugs or delays in hardware delivery. The following list serves a risk plan:

- The interactive beamer had to be acquired before the end of week five such that in case of problems the beamer could be returned
- The deadline for an initial version of the program was set at the end of week seven
- Back-ups of every working version of the program should be saved
- The prototype needs to be finished before January 29 to reserve time to test it in symposium set-up

10.4 Operations and logistic concept description

Since the resulting product of this design project will not need structured support, an extensive operations and logistics concept description was found to be outside of the scope of this project.

After acquisition of the product, the client only needs to mount the equipment. This could be done using the instructions of a manual. Then the only physical maintenance required for the product will be in terms of replacing the filter or lamp of the beamer. The user could install updates when improved versions for the software have been developed. In case of problems with the product, the user could contact the design team to solve the issues.

10.5 Requirements compliance matrix

This section discusses whether the design requirements have been met that were defined at the start of the design phase. First a short overview of the requirements is given, then a compliance matrix is made to check whether the requirements are met. In case that the design does not meet certain requirements, an explanation and the required modifications will be discussed in the feasibility analysis.

10.5.1 Product requirements

All the different types of requirements and constraints, that are extracted from the clients demands and wishes, were established in the baseline report [2] by making a requirements discovery tree. The outcome of the requirements analysis is shown below in a numbered list. Next to that, a check is performed on the functionalities of the prototype tool. Since the tool could always be extended with extra features and improved in terms of speed, the listed functionalities and requirements hold for the symposium tool version that was defined in the priority scheme in chapter 2.3.

Constraints

1. For the design process ten people are available.
2. The design time-frame is bound by ten weeks.
3. A budget of €2,500 is available for acquiring hardware.
4. The selling price for the design tool should be no higher than €10,000.

Requirements

1. The tool has to be interactive.
2. Results of the calculations done by the tool should be rapidly displayed at minimal 30 fps.[1]
3. The tool shall have a maximum response time of 2-3 seconds.[1]
4. The tool should be able to simulate unsteady, incompressible, viscous flows around airfoils.

Functionalities

1. Input
    1. The user shall be able to load preset shapes.
    2. The user shall be able to reshape preset shapes.
    3. The user shall be able to draw shapes on the grid domain.
    4. The user shall be able to draw multiple objects.
    5. The user shall be able to set parameters for α, c, μ, Re and freestream parameters like v, ρ, p and T.
2. Output
    1. The tool should display a vector field of the flow.
    2. The tool should be able to make a pressure coefficient graph.
    3. Aerodynamic parameters like C_l, C_d, C_m, L/D, L, D, M, x_{cp} and x_{ac} should be shown to the user.
    4. Errors and warnings shall be displayed in case of peculiar user input.
3. Layout
    1. The tool shall have an interactive menu that is intuitive to use.
    2. The user shall be able to select the preferred output.

[1] For a grid size of 128 by 128

10.5.2 Compliance matrix

Table 10.2 represents the requirements compliance matrix in which is checked whether each requirement and constraint is met or not. The first column represents the number of the requirement or constraint, referring to the list mentioned in section 10.5.1. The second column indicates whether the requirement is met or not. The last two columns show the difference in the specified value of the requirement and the value of the actual designed product.

Table 10.2: Requirements compliance matrix

Number	Compliance	Set value	Actual design
Constraints			
#1	✓	10 people	10 people
#2	✓	10 weeks	10 weeks
#3	✗	€2,500	€2,707
#4	✓	€10,000	€9,999
Requirements			
#1	✓	-	-
#2	✓	30FPS	60FPS [2]
#3	✓	2-3 seconds	<1 second [2]
#4	✓	Unsteady, viscous, incompressible	Unsteady, viscous incompressible

A check on the functionalities of the prototype tool can be found in table 10.3. Further remarks about the compliance of the features are given in the feasibility analysis. A more detailed overview of the features that the current prototype will be capable of at the symposium is shown at the end of this chapter.

Table 10.3: Functionality compliance matrix

Functionalities					
Input		Output		Layout	
#1.1	✓	#2.1	✓	#3.1	✓
#1.2	✓	#2.2	✗	#3.2	✓
#1.3	✓	#2.3	✗		
#1.4	✓	#2.4	✓		
#1.5	✓				

10.5.3 Feasibility analysis

In the compliance matrix it can be seen that not all requirements and constraints are fulfilled. The reasons why the design does not meet the requirement, or the modifications required to meet it, will be discussed in this section.

Although constraints define the limits of certain design aspects, the design team managed to exceed the initial budget limit that was available for acquiring hardware. Firstly, this could have been prevented by buying a cheaper beamer. The consequence would be that the design tool loses its interactivity, which is of major importance for the client. Secondly, less expensive computer hardware components with lower performance could have been ordered. Since real-time simulation was estimated to be a killer requirement, it was necessary to buy the higher performance hardware components. When consulting these aspects with the client before the actual ordering of the hardware, the client was willing to raise the budget by 10%. This second budget limit (of €2,750) has not been exceeded.

[2]Measured values for displaying rate and response-time are based on simulations at 28-01-2014 on a 128x128 grid.

The three remaining constraints have not been exceeded. The prototype of the tool has been made with ten people within ten weeks. The selling price of the design tool could even be lower than €10,000.

The degree of interactivity is hard to express in numbers. The product is made interactive by enabling drawing as input. Furthermore the tool is designed such that it is intuitive to use, for example the drawings are easy to adapt. The interactivity requirement is validated by the satisfaction of the client. Although there are more features possible to make the tool even more interactive, the client is pleased with the degree of interactivity of the tool.

The other three requirements have also been met and therefore the design of the prototype can be seen as a success. By using the correct equations and assumptions, the tool is able to simulate unsteady, viscous and incompressible flows. For a 128x128 grid, the tool has a displaying rate well above 30 FPS and the response time required for initial computations is even below one second.

Simulations with multiple objects and highly turbulent flow have a displaying rate of approximately 55 FPS (on a 128x128 grid). In case of a more smooth flow, using an airfoil with a small angle of attack, the tool can reach 70 FPS. The goal of speeding up the tool still remains, because larger grid sizes can be used while keeping the current displaying rate. At the moment of writing the grid size can be increased to 180x180 while satisfying 30 FPS.

The same approach holds for the initialization time. For a 128x128 grid, the response time for initial calculations is about 1/10 seconds. When enlarging the grid to 256x256 units, the initialization time just meets the requirement of three seconds.

The prototype tool contains almost all functionalities that were set for the symposium tool. At the moment of writing this report, the only issue that is not fixed yet is the implementation of the pressure. This causes that the prototype could not perform all the desired functionalities yet. However, it is estimated that this could be implemented before the symposium.

For the input, the absence of the pressure does not have consequences. All functions mentioned in the list regarding user input are implemented.

Since the pressure computation is not implemented yet, the tool is not capable of providing the aerodynamic parameters and a pressure coefficient graph. The programming code for both the graph and all aerodynamic parameters are fully worked out and can be integrated rapidly as soon as the pressure is computed and validated.

The two requirements for the layout are also met. The user could select the preferred output, and the user interface is intuitive to use.

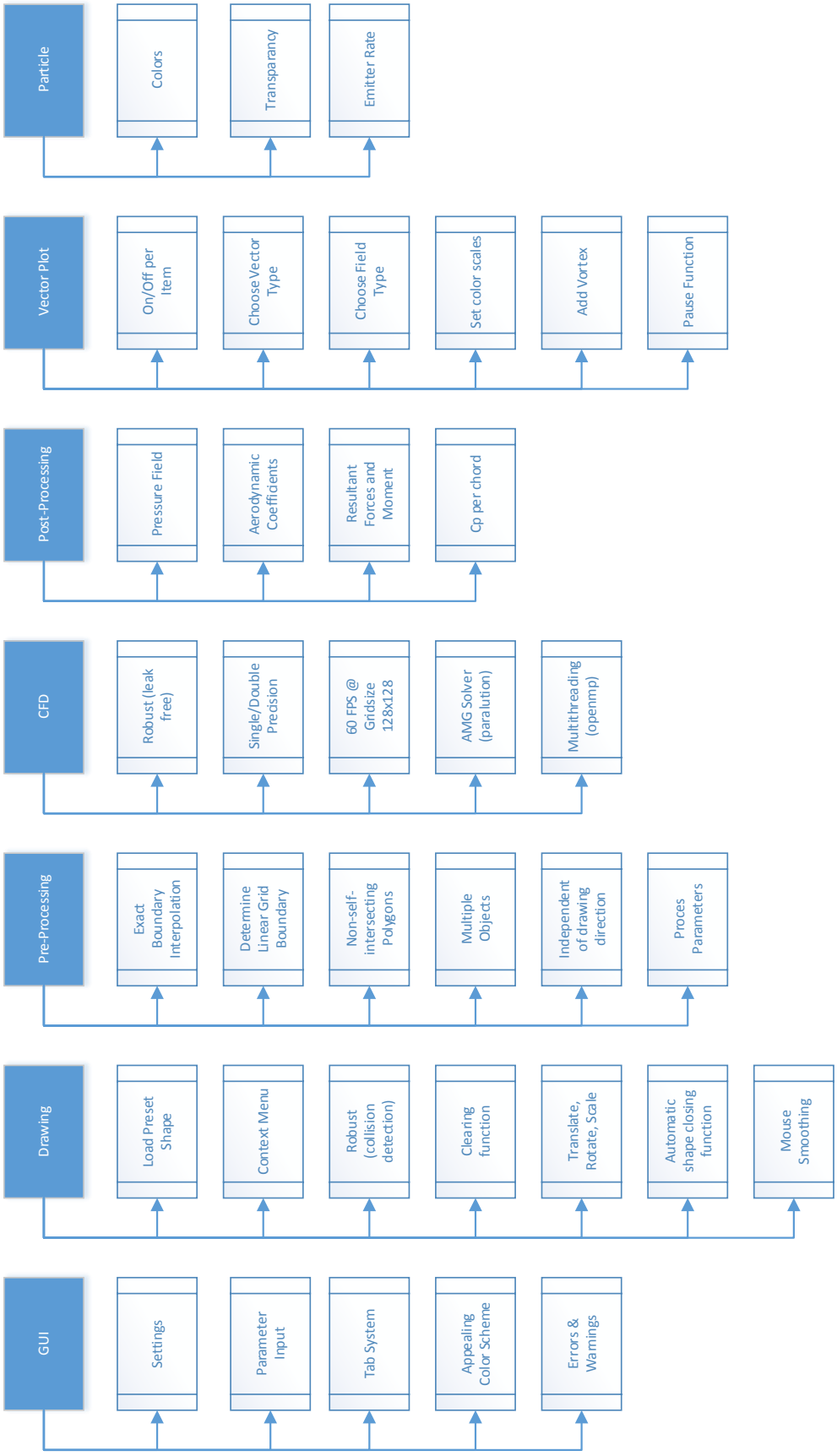


Figure 10.1: Features of current prototype

11. Future development

In this chapter, further development of the prototype tool for post-DSE activities is discussed. First, a variety of improvements is suggested. Then, a project design and a logical development description is given and at last a market analysis and return of investment is performed for a fully developed three dimensional wing design tool.

11.1 Improvements for tool

The prototype that is going to be delivered at the symposium has many features and functionalities. However, the full capability and functionality of the tool is yet to be realized; this can be realized with more development time. A lot of improvements could be implemented by either adjusting existing features or adding new ones. In figure 11.1 all small improvements are listed as well as more significant changes that could be implemented.

11.1.1 Small improvements

The small improvements are subdivided in four categories:

- Extend speed
- Extend analysis
- Extend visualization
- Extend features and input parameters

The improvements enhancing computational speed all concern the CFD block of the program. This is because the computations performed in this block are the bottleneck of the program at the moment. The improvements listed under *Extend Analysis* come down to adding aerodynamic features such as letting the program take into account a ground effect and downwash. The *Extend visualization* improvements concern the visuals that are shown to the user as program output. The last category hosts all the remaining improvements. These range from the possibility to input additional parameters to the ability to add control surfaces to a main wing in a specific flight condition.

These changes are all considered relatively easy to implement and would be the first ones to attempt in further development of the tool.

11.1.2 Significant changes

Whereas the small improvements can simply be added to the tool, the significant changes result in complete adjustment of the existing program. The first change could be making the program simulate and visualize airflows around 3-dimensional objects. This would require a lot of additional computation power and more efficient programming. Letting the program allow for compressible flow computations would be a second improvement that could make the tool more useful for engineering purposes. To this end the CFD solver should be adjusted or replaced by a solver which calculates compressible flow characteristics. Finally, the ability to set an object material could be implemented in future development to take into account the weight or mechanical properties of the input object.

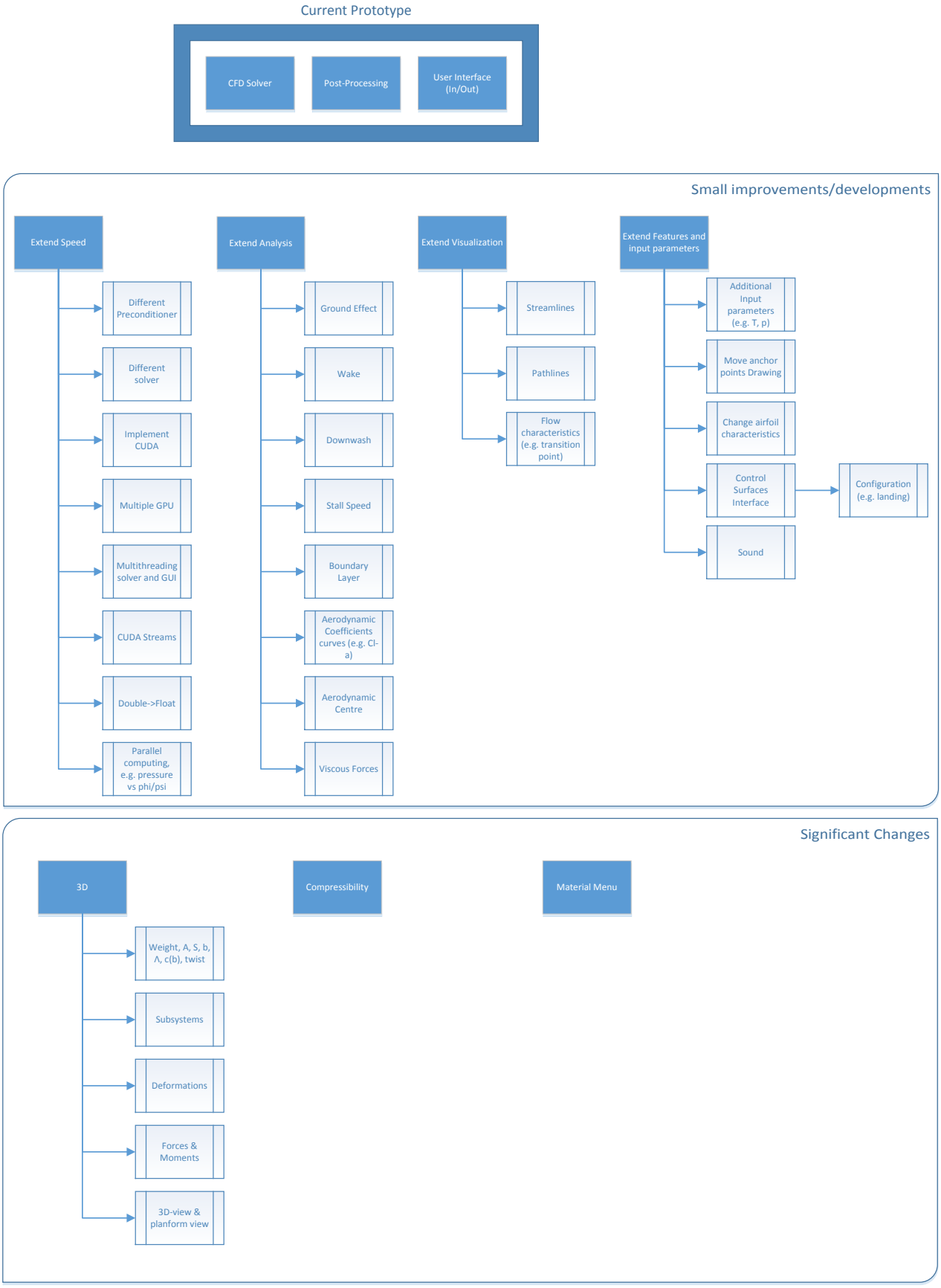


Figure 11.1: Future developments of wing design tool

11.2 Project design and development logic

In figure 11.2 the logical order of activities that will be performed after the DSE is presented. During the DSE, documentation is made to serve as a user's guide. This is necessary to maintain development of the tool after completion of the DSE. This way, external people can use this documentation as a springboard from which the tool can be further improved. It will then be mounted in the Aerodynamic lab, where it can be used as a means of demonstration for educational and communicative ends. The program code can be further elaborated and improved regarding the features discussed in section 11.1. This can be done when it is open source ware.

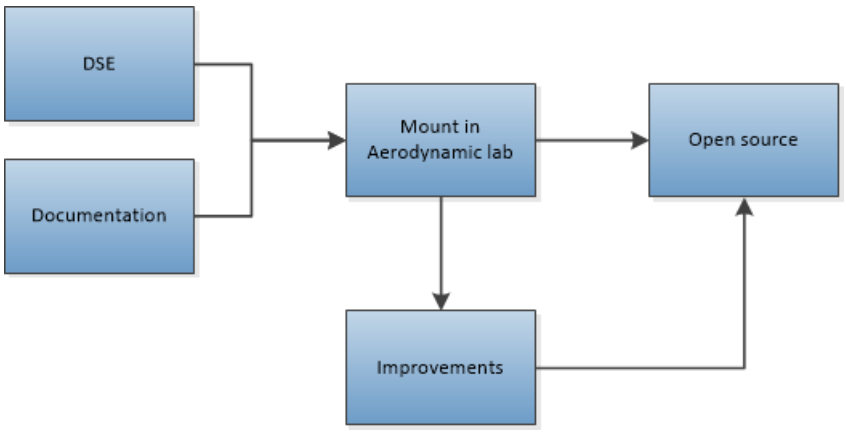


Figure 11.2: Post-DSE activities

11.3 Market Analysis

The major purpose of this market analysis is to establish a target cost for the rapid wing prototyping tool. Systems Engineering (SE) tools will be used to achieve this. For the Market analysis the Multidimensional Market Definition (MMD) and SWOT-analysis are the primary tools available.

11.3.1 Multidimensional Market Definition (MMD)

In order to identify market segments, an MMD graph is used. This is a 3-dimensional graph with on its axes:

- Functions of the product
- Technology/materials of the product
- Customers

Every axis is segmented and shown in table 11.1.

Table 11.1: Axis segmentation for the MMD graph

Customers	Technology	Functions
Government	CFD	Educational
Non-profit		Communicative
Private		
Companies		

Each cell represents a market segment. This way it is assured that all market segments are taken into consideration. All market segments, with the exception of the government, are considered potentially

profitable; because the government is a body which would have no sufficient need for a specific tool like this, it is not considered profitable. Also, the private market is not considered to be of much interest for this reason. The non-profit market is considered to be of interest. Technical universities, for example, could use this design tool for educational purposes, but also for communicative purposes internally or in relation to companies. Aerospace companies are also a potential market. They probably would not use it for educational purposes, but as a communicative tool it could be very effective as a means to communicate their ideas to customers. Museums are also potentially interesting, but the market is very small.

11.3.2 SWOT Analysis

The rapid wing prototyping tool itself, in relation to markets in general, is analyzed with a SWOT analysis. Here the strengths, weaknesses, opportunities, and threats are evaluated to give a more clear view of the potential success of the product on the market. It can also give insight into what could be improved to increase this potential. The SWOT analysis is given in figure 11.3.

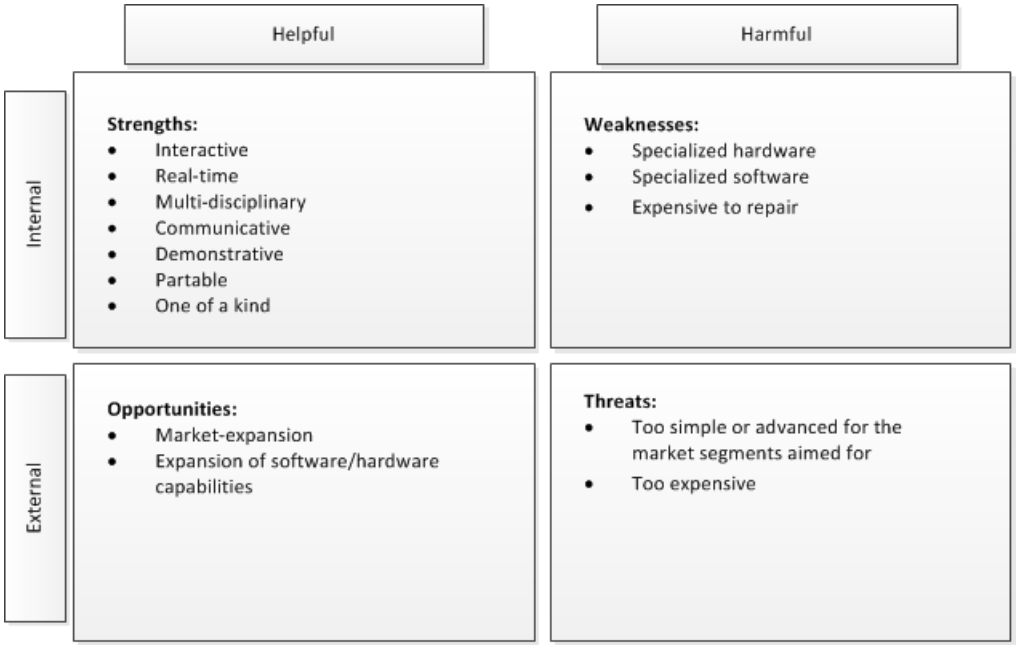


Figure 11.3: SWOT analysis for rapid wing prototyping tool

From the strengths and opportunities it becomes clear that there is a lot of potential. The main reasons for this are that the product is one of kind and it could therefore be a first in a new market. If it proves to be effective in the aerospace sector, similar products could be developed for other technical disciplines.

The downside of this product is that it will be difficult to develop the software and hardware which could cause unwanted tool behavior. Another concern is that if the tool does not provide for the needs of the specified markets, chances are that the tool will waste its potential. When selling one of the tools to a science museum one could argue that a price of €10,000 is too high. And aerospace related companies could argue that they already have 2D or 3D CFD tools that are more advanced and fulfill their needs. It is therefore important to make clear to those companies that this is not so much an analysis tool, but a tool for designers to communicate their ideas.

It can be concluded that there is potential for this tool. However, getting it on the market and selling the numbers needed to get a profit is going to be a challenge. A marketing campaign that targets aerospace companies specifically and makes sure that they understand the concept is going to be crucial for the success of this tool.

11.3.3 Target Cost

One of the constraints for the development of an interactive and rapid flow simulation tool was the maximum cost price of €10,000. The final product should therefore not have a higher cost price than €10,000.

11.3.4 Target Volume

The interactive flow simulation tool is designed mainly as a communicative tool for design engineers from aerodynamic departments of engineering companies. Also, it could be used as an educational tool as well in the environment of classrooms or science museums. When aiming to sell the tool to engineering companies it should be noted that the competition is rather high since a various amount of similar tools exist. Emphasizing the interactive and communicative nature of the tool, it is found feasible to sell the tool to these companies. Looking at the amount of aerospace companies in the Netherlands and Europe, and estimating that the product could be sold to 30% of those companies, a total of 75 units could be sold in the first year.

In the education target group about five products can be sold to engineering faculties in the Netherlands and about 55 can be sold to science museums all over Europe. The total amount of sold products after the first year strongly depends on the specific features of the final product, which is something that is not fixed at this point. An estimation of the amount of products that will be sold ranges from 50 to 250 units. To be able to make an estimation for the selling price, it is assumed that 135 units will be sold in the first year.

11.4 Development Costs

Development costs of the interactive airflow simulation tool is based on the amount of labor hours and the qualification of the team that spends these hours on the development of the tool.

It is estimated that constructing the final product would require 4,800 man hours, which would be ten people working full-time for three months. This estimate is based on a skilled team that has a basic understanding of aerodynamics, large knowledge on computational modeling and advanced programming experience. A team capable of this task would consist of three programmers, five aerodynamicists (preferably with programming experience), one graphical designer and one hardware expert.

The salary of a programmer with about five years experience will be about €4,100 per month [1]. The salary of an aerodynamicist will be approximately €3,200 per month [2]. It is estimate that a graphical designer will cost €2,500 per month and a hardware expert €3,500. Taking into account taxes and collective agreement costs an employee will cost roughly 1.3 times as much for the employer [4]. This will lead to a total of personnel costs of €133,700.00 as is calculated by equation 11.1.

$$\begin{aligned} &3\,months \cdot (3 \cdot salary\ programmer + 5 \cdot salary\ hardwareexpert + salary\ graphical\ designer \\ &\quad + salary\ hardware\ expert) \\ &= 3 \cdot ((€4,100 \cdot 1.3) \cdot 3 + (€3,200 \cdot 1.3) \cdot 5 + (€2,500 \cdot 1.3) + (€3,500 \cdot 1.3)) \\ &= €133,770 \end{aligned} \tag{11.1}$$

[1] "Salaris Programmeur." Rousch. N.p., n.d. Web. 21 Jan. 2014.
[2] "Aerodynamicist Salary (United Kingdom)." PayScale. N.p., n.d. Web. 21 Jan. 2014.

11.4.1 Material and production costs

The material cost consist of hardware for the calculations part and the visualization part. The cost breakdown of all the hardware is depicted in table 11.2.

Table 11.2: Hardware cost breakdown

Component	Product	Price
CPU	Intel i7 4770K	€292.99
GPU	GTX 770	€353.44
RAM	Crucial	€66.89
Motherboard	Gigabyte Z87-HD3	€91.90
SSD	Samsung 840	€84.99
PSU	XFX Pro 550W	€55.89
Projector	Epson EB485-Wi	€1728.00
Integrated product case	TBD	± €30.00 +
		€2704.10

As an integrated product it is desired that the beamer and computer will be integrated with one another. All components will be put into one compact casing that provides for both the calculating power and the camera. The hardware expert has to breakdown the beamer into parts and integrate it into the casing. This process will have a steep learning curve and is estimated to cost about five man hours per product, this is included in the development cost breakdown.

$$\frac{€3,500 \cdot 2}{160} \cdot 5 \cdot 250 = €54,700 \tag{11.2}$$

A basic toolbox needs to be acquired for assembly of the product. The estimated costs for this is no more than €100.00. It is estimated that some start up problems will arise, for this a buffer of €500.00 should be available. This brings the total material and production costs on €365,653.50, as is equation 11.3.

$$€2704.10 \cdot 135 + €100 + €500 = €365,653.50 \tag{11.3}$$

11.4.2 Return on investment

Setting a selling price for a this product can be rather difficult, since it depends not only on the plain production and development costs, but also on the selling prices of similar products, the nature of the differences between those products and the expected return on investment (ROI). Due to these variables and the little knowledge on the selling opportunities at this stage, the selling price is assumed to be equal to the amount that was defined as a maximum by the requirements. The return on investment (ROI) can be calculated according to equation 11.4 and 11.5.

$$\begin{aligned} investment &= development\ costs + material\&production\ costs \\ &= 133700 + 365653 \\ &= €499,353 \end{aligned} \tag{11.4}$$

$$\begin{aligned} ROI &= \frac{gain\ from\ investment - cost\ of\ investment}{cost\ of\ investment} \\ &= \frac{10000 \cdot 135 - 499353}{499353} = 1.7 \end{aligned} \tag{11.5}$$

From the ROI can be concluded that for a selling price of €10,000 that further development of the interactive airflow simulation tool is a profitable investment. Since the actual selling price is more likely to be smaller than €10,000, the actual ROI will be smaller than 1.7. As long as the selling price is above €7,400, the ROI will be positive. A final selling price should thus be in the range of €7,400 to €10,000.

12. Conclusion

In this report the design process that led to the realization of an airflow simulation tool has been discussed. The team has achieved to create a prototype that is interactive and gives (near) real-time visualization outputs.

The prototype is a combination of a custom made PC and an interactive beamer. The software that has been developed, is based on a CFD-solver, that is robust, fast and gives accurate results. To allow for intuitive use, a graphical user interface is made that can handle hand drawn inputs and quickly responds to the input given by the pointer. The team also decided to make the tool open source, such that others are stimulated to use, elaborate and improve the prototype. The team has presented several ideas for improvements and elaboration in section 11.1, that can be interesting for others that like to further develop the tool.

It can be concluded that the team has achieved their goal to finish a working prototype, to demonstrate it at the symposium and will keep making improvements up until the last day before the symposium.

With the emphasis on increasing the performance of the tool, this report proposes several areas of improvement.

The authors recommend investigating on improvements to increase the speed of the CFD solver, the pressure solver and the graphical output, since these are the bottlenecks of the prototype. Furthermore improvements can be made by elaborating the functionalities of the prototype, depending on the application for which the tool will be used. Also, attention should be given to the limitations with respect to the robustness of the tool.

To use the tool for communicative and educative purposes, the team proposes installing the interactive beamer and the PC with the tool working on it, in the Aerodynamics Lab of the Faculty of Aerospace Engineering of the TU Delft.

Bibliography

[1] I. H. Abbott and A. E. von Doenhoff. *Theory of Wing Sections: Including a Summary of Airfoil Data*. Dover Publications, 1959.

[2] J. Barnhoorn, S. Brust, D. van Herwaarden, et al. DSE - Wing design tool. Baseline Report, November 2013.

[3] J. Barnhoorn, S. Brust, D. van Herwaarden, et al. DSE - Wing design tool. Project Plan, November 2013.

[4] BerekenHet.nl. Loonkosten voor de werkgever. `http://www.berkenhet.nl/ondernemen/loonkosten-werkgever.html`, 2014. [checked January 28, 2014].

[5] L. Chen. Finite difference methods. October 2013.

[6] G.-H. Cottet and P. Poncet. Advances in direct numerical simulations of 3d wall-bounded flows by vortex-in-cell methods. *Journal of Computational Physics*, (193):136–158, 2003.

[7] R. Dwight and M. Kotsonis. Interactive wing design using rapid prototyping. Project Guide, November 2013.

[8] R. W. Franklin. Pnpoly - point inclusion in polygon test. `http://www.ecse.rpi.edu/~wrf/Research/Short_Notes/pnpoly.html`, 12 2009. [accessed 10-December-2013].

[9] J. John D. Anderson. *Fundamentals of Aerodynamics*. McGrawHill, 5th edition, 2011.

[10] P. Poncet. Analysis of an immersed boundary method for three-dimensional flows in vorticity formulation. *Journal of Computational Physics*, 2009.

[11] M. S. Selig. Uiuc airfoil data site. `http://aerospace.illinois.edu/m-selig/ads/coord_database.htmll`, 2013. [accessed 20-January-2014].