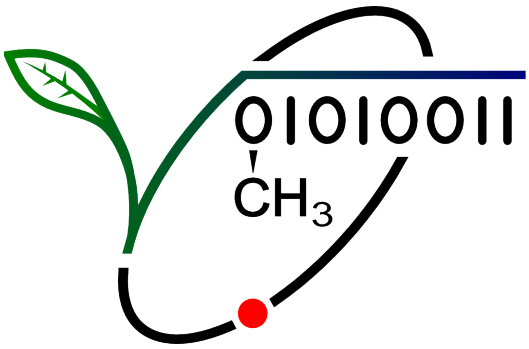
Spezialschulteil des Albert-Schweitzer-Gymnasiums Erfurt



Final course paper
School years 2018 to 2020

Determination of the drag coefficient of laser scanned bodies using a fluid flow simulation in 3D space

Subject supervisor:	Mr. Brenner
Course paper supervisor:	Mrs. Bangsow-Bösa
Students:	Adrian Kühn (12b) Frank Long (12a) Paul Alexander Marschall (12a)
Date:	20.12.2019

Contents

Introduction

Every form of mobility on our planet depends on its level of efficiency. This is how we transferred the aerodynamic properties of penguins and seals to our cars, trains and planes. But we also need to understand the principle behind the motion of a solid through a fluid in order to advance our technique. The decisive factor here is the consideration of the drag coefficient, also known as the Cw-value. Investigations of this degree of aerodynamic slippage of a body are currently carried out in wind tunnels. But these systems are expensive and maintenance is difficult. So we asked ourselves whether these examinations could alternatively be carried out with the help of appropriate software.
In our seminar paper we want to create a program that makes it possible to analyze flows around bodies using a virtual wind tunnel. For this we have to deal with the basic physical and mathematical models of fluid dynamics. We then plan to transfer these models to a numerically feasible simulation method. In this method, it is important to define the specifications of our virtual wind tunnel as well as variables that can be analyzed. Finally, we can calibrate our simulation on standard bodies.
With our program it should be possible to import three-dimensional objects. For this purpose, the physical properties of the fluid flowing around are defined and inserted into a corresponding simulation space. A simulation is then carried out in which the fluid flows around the body. Here, as the core of our work, specific data is collected, which we can later use to deduce the Cw value of the body. Finally, we want to implement a visualization of the flow that is as comprehensive as possible and illustrate the collected data.
We chose this topic ourselves because we are very interested in flow problems, especially in technology. With our program it should be possible for hobbyists or students to examine objects with regard to their aerodynamic properties.

At this point we would like to thank our supervisor Mr. Brenner for his active support regarding the understanding of physical and mathematical models. We would also like to thank the student research center and thus Mr. Paulig and Dr. Wagner, who made it possible for us to work on a wind tunnel and provided us with a 3D scanner. We would also like to thank our project supervisor Ms. Bangsow-Bösa for her support in preparing the written part of our work. We would also like to thank our families for their moral support and everyone else who helped us to accomplish our tasks.

1 Mathematical-physical consideration of fluids

1.1 The physical principles of fluid dynamics

The aim of our work is to determine the flow resistance coefficient of bodies by simulation. In the course of this, we will first deal with the physical basics of fluid dynamics in this chapter. Accordingly, we first consider the macroscopic model of fluid dynamics, which is based on the Navier-Stokes equations. This describes the dynamics of volume elements in the fluid. However, the equations go far beyond the school material, which is why we have dealt with the basic elements of the equations. Operators and operands that are initially unknown to us are used, which can be understood from equations (1) to (5).[35, 36]

$$\nabla \cdot \vec{v} = 0 \tag{1}$$

$$\rho\left(\frac{d\vec{v}}{dt}\right) = -\nabla p + \mu\nabla^2\vec{v} + \rho F \tag{2}$$

$$\rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) = \rho g_x - \frac{\partial p}{\partial x} + \mu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) \tag{3}$$

$$\rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) = \rho g_y - \frac{\partial p}{\partial y} + \mu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) \tag{4}$$

$$\rho(\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z}) = \rho g_z - \frac{\partial p}{\partial z} + \mu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) \tag{5}$$

The equations describe the behavior of incompressible fluids. Equation (1) represents the conservation of mass. Equation (2), whose three components are written out again separately in equations (3) to (5), represents the conservation of momentum, i.e. Newton's second law, in the fluid. Both equations include the operator Nabla (∇), which can be viewed as a vector. Depending on the application, this generates a vector field or a scalar field. In equation (1), the operator ∇ comes before the physical variable speed. This is also a vector, which is why the scalar product of both quantities creates a scalar field, the so-called divergence. This describes the occurrence of sinks and sources in the vector field of the velocity. If it is equal to zero, both phenomena do not occur and, as mentioned, we can assume conservation of mass. In the second equation, ∇ comes before pressure. The pressure in a fluid is a scalar quantity. The application of ∇ results in a vector field, the so-called gradient. This points in the direction of the greatest pressure change and thus represents the direction of the acting force. The change in momentum is described by all the acting forces, which are represented by the right-hand side of the equation.
In order to be able to apply the Navier-Stokes equations, one needs initial conditions. We want to understand such an application in the following using a simple example. The choice fell on a funnel through which it flows (compare Figure 1).

Since turbulence is difficult to capture numerically, a laminar flow is assumed. So that the velocity at individual points in the flow field is time-independent, the flow is still stationary. We also assume that the fluid is ideal, which also satisfies the incompressibility requirement of the Navier-Stokes equations.
The goal is now to set up a velocity equation for volume elements of the fluid. P_0 is the starting point of a volume element, we are looking for the velocity at a specific point on the streamline. The cone spans from H to h. For the mathematical model we assume that the velocity in plane H is 0 and that the volume elements in a ray move towards the focal point 0[a]. We assume that the speed on this orbit increases linearly[b]. Another assumption is that the speed is the same within a plane. So we can use the Torricelli formula $v = \sqrt{2g(H-z)}$ for the magnitude of the velocity.

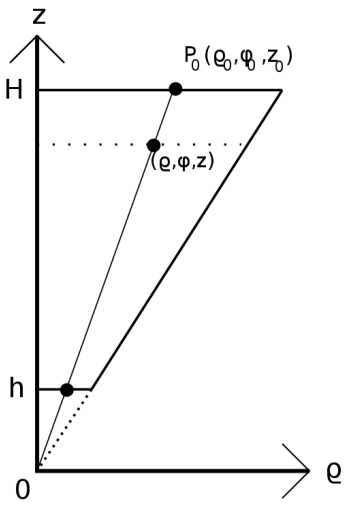


Figure 1: Model of the trajectory of a volume element in a funnel flow [61]

[a] The origin of coordinates
[b] You can find the corresponding orbital equations in the appendix on page 22.

Apart from gravity, fluid is only affected by funnel walls. We assume Equation (6) for the location-dependent velocity vector.

$$\vec{v} = \frac{\sqrt{2g(H-z)}}{\sqrt{\varrho^2+z^2}}\begin{pmatrix} -\varrho \\ 0 \\ -z \end{pmatrix} \tag{6}$$

We have to check this equation below. For this we put them into equation (1), which we adapt to the cylindrical coordinate system [20].

$$\frac{\varrho}{z} = \frac{\varrho_0}{z_0} \tag{7}$$

$$\varphi = \varphi_0 \tag{8}$$

$$\nabla \cdot v = \frac{1}{\varrho}\frac{d}{d\varrho}(\varrho v_\varrho) + \frac{1}{\varrho}\frac{d}{d\varphi}(v_\varphi) + \frac{d}{dz}v_z \tag{9}$$

The result is the expression to be understood in equation (10).

$$\nabla \cdot v = \frac{1}{\varrho}\frac{d}{d\varrho}(\varrho\frac{\sqrt{2g(H-z)}(-\varrho)}{\sqrt{\varrho^2+z^2}}) + \frac{d}{dz}(\frac{\sqrt{2g(H-z)}(-z)}{\sqrt{\varrho^2+z^2}}) = -\frac{\varrho(4H-5z)}{\sqrt{(\varrho^2+z^2)2g(H-z)}} \neq 0 \tag{10}$$

This represents an inequality, so we can conclude that Equation (6) does not satisfy Equation (1). So the divergence is not zero for all points in the funnel.
An alternative method could be to find an alternative expression for equation (6) using computer approximation methods for equation (2). An exact solution of equation (2) applied to the funnel flow is not known to us at the moment.
In summary, it can be concluded that the assumed equation for the velocity v is wrong.

1.2 On the linear approximation problem

In chapter 1.1 we mentioned computer-aided approximation methods that can be applied to the Navier-Stokes equations. One of these methods is the linear approximation, which we now want to take a closer look at.
The general form of such an approximation is the explicit Euler method. This is a simple method for solving a differential equation numerically. In it, the growth of the graph is calculated at predetermined intervals, which then completes the adjoining graph.[1]
The numerical method has to be applied, since an explicit representation of the solution is not possible for numerous differential equations.

In the following example, the y-value of a function is present as an initial value. Equation (11) can be used to determine the following y-values:

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + r(x,\Delta x) \tag{11}$$

However, in the 3D coordinate systems used in our simulation, three values change, replacing Equation (11) with Equation (12).

$$f(x_0+\Delta x, y_0+\Delta y) \approx f(x_0,y_0) + \frac{\partial f}{\partial y}\Delta y + \frac{\partial f}{\partial x}\Delta x \tag{12}$$

The slope of the coordinate planes is calculated using the respective partial derivative. A 3D graph is obtained.
The $\approx$ sign in equation (12) comes from neglecting $r(x,\Delta x)$. For example, r (compare Figure 2) is the distance from the derivative to the actual function, which is why the adjoining graph moves further and further away from the original graph. However, r gets smaller the smaller Δx is. Therefore, one can disregard r when solving differential equations numerically, because one calculates here with minimal Δx values.

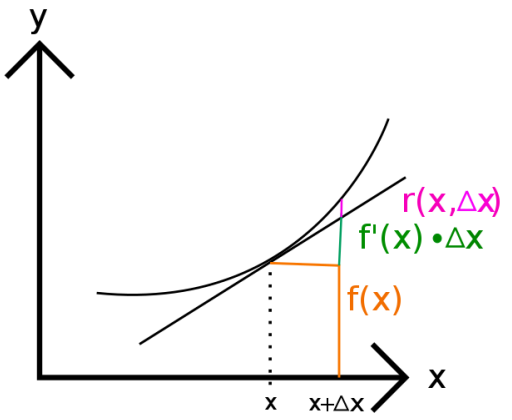


Figure 2: Approximation of a function using its derivative [61]

In order to be able to determine whether a graph is getting steeper or flatter, you need the 2nd derivative of its function. Finally, equation (13) is obtained from equation (11). We approximate the additional term r using the mean value theorem.

$$f(x_0+\Delta x) \approx f(x_0) + f'(x_0)\Delta x + \frac{f''(x_0)}{2}\Delta x^2 \tag{13}$$

As an example of such a calculation, we now consider a function $f(x) = ln(x)$. Since the derivative of ln is already known, equation (14) is obtained for y.

$$y \approx ln(x) + \frac{1}{x}\Delta x + (-\frac{1}{x^2})\Delta x^2 \tag{14}$$

We set Δx to 0.01 and consider the function and its approximation in the interval from 5 to 5.07. We now calculate the points of approximation numerically with the help of a computer. Now we can find the discrepancy between the calculated points. This amounts to 0.02 for the x value of 5.07.

[1] Appendix p.22 Figure 16: Explicit Euler method

2 Microscopic approach to flow simulations

2.1 Motivation and basic idea

In Chapter 1.1 we found that a simulation based on macroscopic models is hardly feasible. In this chapter we try to implement a simulation based on the laws of thermodynamics. So we change the perspective from the macroscopic to the microscopic view of fluids.
The basic idea is to simulate the components of a fluid, atoms or molecules. What is meant is to determine a number N of the most favorable possible components and to give them a simulation space using natural constants. In terms of computer science, the individual component is now an object to which we can assign attributes and methods. Likewise, the body to be examined is an object, but of a different class. With the help of their methods, we can try to make these objects interact as naturally as possible in the simulation space. Based on certain events that occur, it is then possible to conclude how slippery the test body is in terms of aerodynamics.

2.2 Implementation of a test program

We initially selected helium as the fluid. Helium is an inert gas, which is why, as a pure substance, it does not form any molecules. The components of the fluid can therefore be viewed geometrically as spheres, helium atoms. We set the temperature of our simulation room to $0°C$, or $273.15K$. Now we can define all parameters of our simulation space using the kinetic gas theory and natural constants. They can be found in Table 1.

Table 1: Defined and determined parameters of our simulation room [2]

Size	Variable	Value	Calculation / Formula / Constants
Temperature	T	$273,15K$	Determination
Number of atoms	N	z.B. 100	Determination, variable
Atomic radius	r_{Atom}	$3,1 \cdot 10^{-11} m$	Natural Constants
Atomic volume	V_{Atom}	$1,2 \cdot 10^{-31} m^3$	$V_{Atom} = \frac{4}{3} \cdot \pi \cdot r_{Atom}^3$
Atomic mass unit	u	$1,660539040 \cdot 10^{-27} kg$	Natural Constants
Atomic mass	m_{Atom}	$6,64647 \cdot 10^{-27} kg$	$4,0026 \cdot u$
Bolzmann constant	K	$1,38065 \cdot 10^{-23} J \cdot K^{-1}$	Natural Constants
Density of helium (normal pressure)	ρ	$0,1785 kg \cdot m^{-3}$	Natural Constants, for $T = 273,15K$
∅ kinetic energy of an atom	$E_{kinetisch}$	$5,65687 \cdot 10^{-21} J$	$E_{kinetic} = \frac{3}{2} \cdot K \cdot T$
∅ speed of an atom	$v_{\varnothing}$	$1304,69 m \cdot s^{-1}$	$v_{\varnothing} = \sqrt{2 \cdot E_{kinetic} \cdot m_{Atom}^{-1}}$

However, this does not yet define the simulation space as space. We choose it here first as a cube whose edge length s can be determined using the above values. So we are looking for: $s(N)$ (compare equation (15)).

$$s(N) = V(N)^{\frac{1}{3}} = (N \cdot V_{Atom} \cdot C)^{\frac{1}{3}} \tag{15}$$

The volume of the simulation space does not correspond to the sum of the atomic volumes. However, there is a fixed connection between the two variables, which is compensated here using C. C can be determined from the density of helium. The result is the following expression (16):

$$C = \frac{m_{Atom}}{\rho \cdot V_{Atom}} = 310303,45 \tag{16}$$

Now a number N of atoms can be defined. You get a random starting position and a random initial speed within the simulation space. The amount of the initial speed can deviate from the calculated average speed of the atoms. The basis for this is the energy distribution according to Boltzmann. The atoms initially have only one method that updates their position based on a time step and their velocity. In addition, they bounce off completely elastically at the boundaries of the simulation space. We don't let any forces act on the atoms, and they don't collide with each other. We test this program primarily for its suitability with regard to its runtime. [2]

2.3 Evaluation of this simulation method

The simulation runs smoothly up to a number of 5000 atoms without the effects of force or more complex methods of updating the atoms or components. With this number of atoms we are in the area of an edge length of the simulation space of $10^{-8} m$. In addition, we have so far ignored the interaction of the particles with each other. Calculating collisions of balls in a three-dimensional space is not a problem. Ultimately, however, we need to give the system a clock that updates it in certain time steps. Tracking interactions of more than two objects within one time step would further increase the computational complexity of the program. The microscopic approach is therefore not suitable for flow simulation.

[2] You can find a visualization in the appendix: page 23, figure 17

3 Implementation of the computer program for simulating flowing fluids

3.1 Procedure for program development

3.1.1 Introduction to the Lattice-Boltzman method

Both the macroscopic and the microscopic observation of the fluids could not lead to a satisfactory fluid simulation. Accordingly, we now use the mesoscopic viewing level, which acts between the others. Here we use simulation methods of numerical fluid mechanics, which solve the complexity of the fluid-mechanical problems, based on the Navier-Stokes equations, approximately and numerically. Instead of simulating individual particles, computational grids are now used as an approximation, which is illustrated in Figure 3. The most common simulation methods in computational fluid mechanics for simulating flowing fluids are the finite difference method (FDM) and the finite volume method (FVM). However, we did not apply these methods directly and instead switched to the Lattice-Boltzmann method (LBM). From our point of view, the Lattice-Boltzmann method was clearer, more efficient and easier to implement in a programming language. Due to the internal structure, which includes a low memory and computing requirement per cell, the method is suitable, among other things, for the calculation of flows in complex geometries.
The Lattice-Boltzmann method was developed in the late 1980s [13] and has its theoretical basis in statistical physics. As already mentioned above, the simulation space or the simulation level is discretized by a grid. The interaction of the microscopic particles is described by the Boltzmann equation. (17)[13]

$$\left(\frac{\partial}{\partial t} + \vec{v}\cdot\nabla_{\vec{x}} + \frac{\vec{F}}{m}\cdot\nabla_{\vec{v}}\right) f(\vec{x},\vec{v},t) = \left.\frac{\partial f}{\partial t}\right|_{Stoß} \tag{17}$$

The left side of the equation represents the distribution density of a fluid as a total time derivative. This distribution density is described by the collision integral on the right side of the equation.

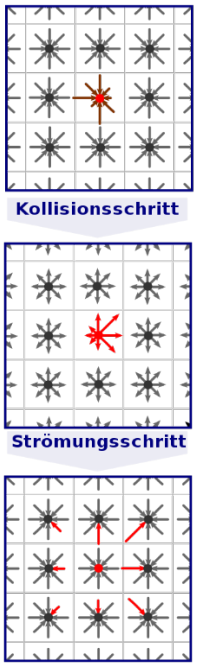


Figure 3: Substeps of the Lattice-Boltzmann method [56]

First we go into the Lattice-Boltzmann method in two-dimensional space, i.e. in the plane. As mentioned above, the plane is divided into many equal squares, so-called cells, by a grid. Eight direction arrows and one zero arrow are assigned to each cell. The directional arrows represent how likely the velocity of the particles associated with a cell is in the direction of the arrow at that cell. A fluid particle can remain in the same place per time step or move into the respective adjacent cells of the square lattice.
The algorithm can be divided into two sub-steps, the flow step and the collision step. In the collision step, the arrows from the neighboring cells collide in the cell to be calculated. Depending on which simulation is aimed at, collision terms with equilibrium functions can be added to the arrows, which depend on the viscosity of the fluid.
However, we did not apply these collision theorems in the first step of our program development and replaced them with self-developed functions in order to keep the program as simple as possible.
In the flow step, all eight directional arrows are forwarded to the next grid point according to their direction. The zero arrows are not changed, so the particles of the zero arrow remain in the cell. The arrows shifted in this way again form the initial situation for the next collision step.

3.1.2 Implementation of the simulation method in 2D space

We implemented the derived simulation method using the Python programming language. This program is based on a finite loop with *Update_Fluids*, *Friction*, *Resonant*, *Show_Fluids* and *Obstacle* functions.
In the first *UpdateFluids* function, the arrows are shifted in the direction of movement to the next cells and displayed in gray in *Show_Fluids* depending on the arrow density of the respective cell on the display area. In the *Friction* function, the arrow sizes are reduced by a constant and reset one cell, so that the fluid does not move infinitely far in space. The function *Eigenoscillation* simulates the propagation of the particles, independently of *Update_Fluids*, by dividing each arrow in a cell into the arrows with the same arrow direction in the surrounding cells. The reflection of the particles on the obstacle is processed by the *obstacle* function. The obstacle is represented by the edge cells of the obstacle. These are colored red in Figure 4.
Each obstacle cell is assigned a reflection angle between 0° and 360°, which describes the reflection of the fluid at this cell. If an obstacle cell is to have several different reflection angles, each side of the square can also be defined as a reflection side, where the fluid is reflected at a 90° angle. Furthermore, the obstacle must be continuously mapped with obstacle cells, so that in the case of a slope, the corners are not the connection to the next obstacle cell, but rather the edges of an intermediate obstacle cell, colored blue in Figure 4. This means that each incline is represented by a step shape and it is also guaranteed that no particles can get through the incline into the interior of the obstacle using the *Update_Fluids* function. However, at some reflection angles, the fluid can still be reflected into the interior of

the obstacle. If this is the case, the fluid moves through the cells of the interior of the obstacle, colored green in Figure 4, and is released back out into the plane at the end of the obstacle without reflection.
Furthermore, the parameters viscosity, flow resistance, propagation speed and size of the arrow directions can be defined in the program.
In order to better understand the particle movement, we have created a particle wall in Figure 4 on the left (600x600) which is 400 cells high and 2 cells wide.

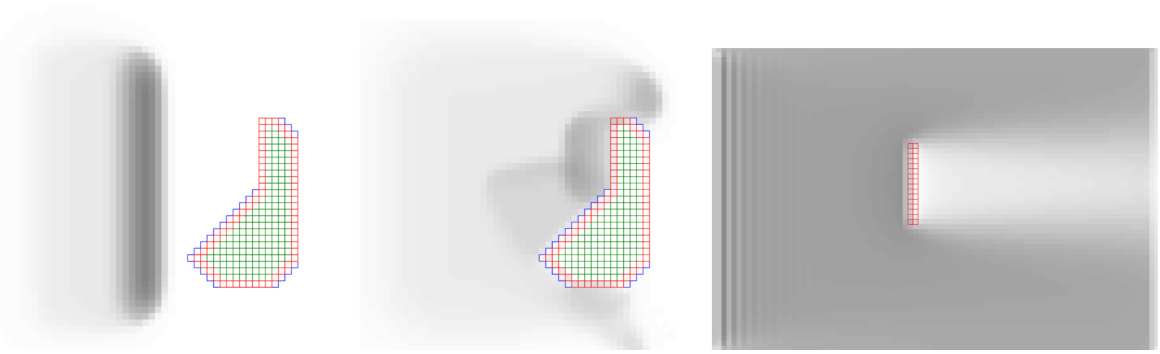


Figure 4: Graphic output of our flow simulation in two-dimensional space with different fluid sources [61]

In reality, however, in a flow simulation, the entire space is filled with flowing particles. This simulation is shown in Figure 4 on the right (900x600) with a simple wall as the obstacle. The fluid source takes up the entire space.

3.1.3 Implementation of the simulation method in 3D space

In the two-dimensional model, 8 directional arrows were used. In three-dimensional space, on the other hand, we work with 26 directional arrows. The program structure is similar to the 2D model.
Only the angle input of the individual obstacle cells differs. For each obstacle cell, a reflection angle must be stored once, which determines the reflection in the two-dimensional plane and another reflection angle, which defines the reflection perpendicular to the two-dimensional plane. This allows each obstacle to be clearly displayed.
Instead of Tkinter, the Poly3DCollection module from Matplotlib is now used for the display.
Figure 5 above shows the reflection from any body without the *Resonant* and *Friction* functions. If one includes the friction and natural vibration, the result is shown in Figure 5 below.

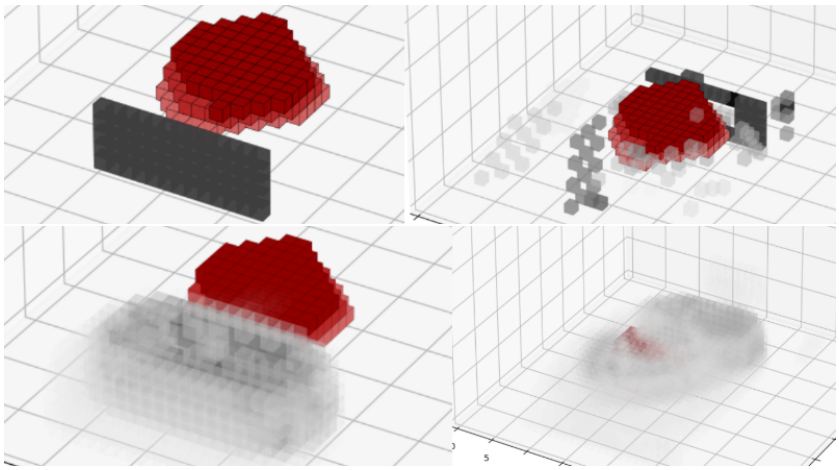


Figure 5: The graphical output of our flow simulation in three-dimensional space with various flow functions [61].

Unfortunately, at the end of these two program implementations, we had to admit that they were too slow and not suitable for real-time displays. Each calculation step required 2 seconds in 3D space at the above resolution.

In order to be able to display a smooth simulation, you would have had to take a picture of each calculation step and then edit it into a greatly accelerated video. In addition to the time problem, there were other errors in the simulation that were not negligible. The accumulations of fluid in front of the obstacles have increased with each time step and have only slightly leveled off at a stable level. In addition, for an incompressible gas, the density in each cell should be the same. This requirement was also not met in our simulations.

3.1.4 Optimization of the flow simulation by collision thermal

Our first attempt to replace the collision term with the simplified functions *Friction* and *Resonant* was ultimately unsuccessful and we had to look more closely at the Boltzmann equation.
First, however, we optimized the flow step in our program and switched from large lists to array systems by Numpy to

make the flow step more efficient. We have also shortened the very time-consuming reflection angle calculation. Now the particles are reflected towards their original position.
Ultimately, the biggest change was the use of the collision term and the associated equilibrium function in the collision step, which replaces the $Friction$ and $Resonant$ functions more efficiently and realistically. These allow each cell to have the same density of particles. This enables flow animation based on particle velocities. These will be explained in more detail below.
The Lattice-Boltzmann method is based on a discretization of the BGK equation[3], which is a simplification of the Boltzmann equation. Equation (18) is solved: [16]

$$N_i(t+1, x+c_i) = (1-\omega)N_i(t,x) + \omega N_{ie}(t,x) \tag{18}$$

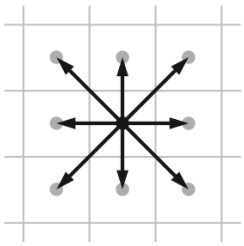
This results in the Boltzman-Maxwell distribution (19): [14, 16]

$$f(v)dv = \left(\frac{m}{2\pi kT}\right)^{\frac{3}{2}} 4\pi v^2 e^{-\frac{mv^2}{2kT}} dv \tag{19}$$

The equilibrium distribution required for the Lattice-Boltzmann method (20) is obtained from the Taylor expansion of the Maxwell distribution: [16]

$$D(\vec{v}) \longrightarrow \omega_i \left[1 + \frac{3\vec{e}_i \cdot \vec{u}}{c} + \frac{9}{2}\left(\frac{\vec{e}_i \cdot \vec{u}}{c}\right)^2 - \frac{3}{2}\frac{|\vec{u}|^2}{c^2}\right] \tag{20}$$

Where c=1 and w_i has a different value for each direction arrow. The different values for w_i result from the fact that the direction arrows, which run horizontally or vertically, cover a smaller distance per flow step than the arrows running diagonally. Some arrows (compare Figure 6) would therefore move faster per calculation step. However, since this does not correspond to reality, the following weightings must be introduced for all arrows (compare formulas (21) to (32)): [16]



$$\vec{e}_0 = 0 \quad (21) \qquad \vec{e}_5 = (1,1) \quad (26)$$
$$\vec{e}_1 = (1,0) \quad (22) \qquad \vec{e}_6 = (-1,1) \quad (27)$$
$$\vec{e}_2 = (0,1) \quad (23) \qquad \vec{e}_7 = (-1,-1 \quad (28)$$
$$\vec{e}_3 = (-1,0) \quad (24) \qquad \vec{e}_8 = (1,-1) \quad (29)$$
$$\vec{e}_4 = (0,-1) \quad (25)$$

$$\omega_0 = \frac{4}{9} \tag{30}$$
$$\omega 1 = \omega 2 = \omega 3 = \omega 4 = \frac{1}{9} \tag{31}$$
$$\omega 5 = \omega 6 = \omega 7 = \omega 8 = \frac{1}{36} \tag{32}$$

Figure 6: Representation of arrow directions with associated weights [57]

Ultimately, the equilibrium function is based on the laws of fluid dynamics and describes the behavior of the flowing particles in an ideal, incompressible gas. In simplified terms, one can say that the equilibrium function in the collision step has the task of balancing several cells with different particle velocities at the same densities. Finally, in order to obtain the final equilibrium function used in our program, the sum (density) ρ of the velocities of a cell must be multiplied by the respective equilibrium function. With this we get the expression (33): [16]

$$n_i^{eq} = \rho\omega_i \left[1 + 3\vec{ei} \cdot \vec{u} + \frac{9}{2}\left(e_i \cdot \vec{u}\right)^2 - \frac{3}{2}|\vec{u}|^2\right] \tag{33}$$

In order to obtain the new velocities of each cell from the equilibrium function, the following term must be applied for each cell velocity (34): [16]

$$n_i^{new} = n_i^{old} + \omega\left(n_i^{eq} + n_i^{old}\right) \tag{34}$$

Here ω can theoretically have a value between 0 and 2 and essentially reflects the viscosity of the fluid. The viscosity ν is calculated as follows (35): [16]

$$\nu = \frac{1}{3}\left(\frac{1}{\omega} - \frac{1}{2}\right) \tag{35}$$

[3] Bhatnagar–Gross–Krook - equation

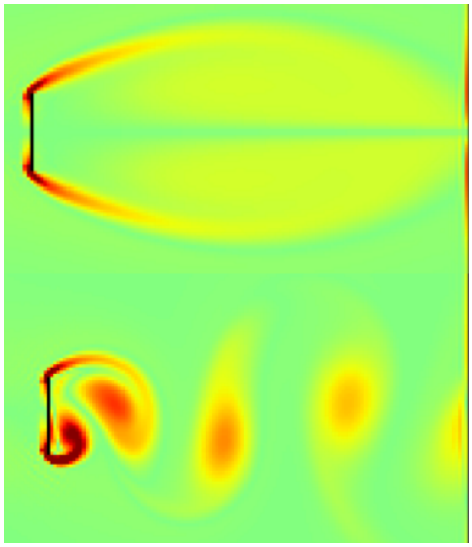


Figure 7: Updated two-dimensional flow simulation for laminar and turbulent flow [61]

However, depending on the flow velocity u_0 of the fluid, instabilities in the simulations already occur at viscosity values above 1.96 and below 0.25.
By optimizing the collision step and the flow step, we were able to speed up our program in the two-dimensional area by a factor of about 100. Figure 7 shows 2 simulations of a simple wall as an obstacle with the new program.
The wake vortices are clearly visible in the lower simulation in Figure 7 behind the obstacle. They are an important feature of a realistic simulation. Because irregular wake vortices occur in every wind tunnel. This also confirms the correctness of the underlying fluid dynamic models and the equilibrium function. Because with the old 2D program, it was not possible for us to generate wake turbulence behind the obstacle. Nevertheless, the wake turbulence can only be generated under certain parameter settings. Accordingly, the viscosity must be very small in relation to the speed and the factor w must therefore be very large. This connection follows directly from the physical flow theory with the Reynolds number. The Reynolds number results from (36): [32]

$$Re = \frac{v_m \cdot d}{\nu} \tag{36}$$

The higher the speed and the lower the viscosity, the higher the Reynolds number. The relationship between viscosity and speed is shown in Figure 30 in the appendix. If the Reynolds number exceeds a critical value, the laminar flow turns into a turbulent flow. Experiments have succeeded in determining the critical Reynolds number at 2040 ± 10. A flow is laminar if turbulence and other imbalances break down again due to external stimuli, such as an obstacle. The smaller the Reynolds number, the faster the turbulence dissolves. However, if the turbulences no longer dissipate or become even stronger, the flow is turbulent.

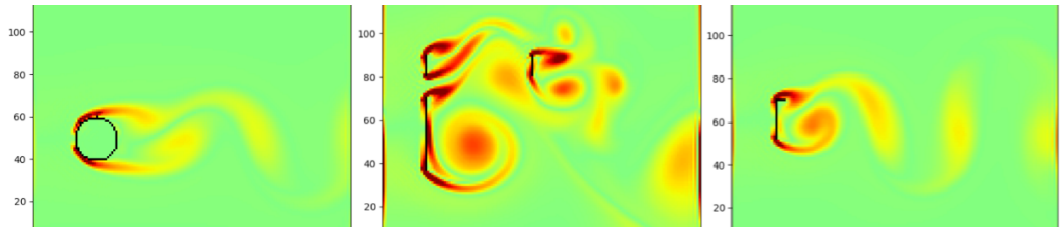


Figure 8: Representation of turbulent flow in our simulation program [61]

In our flow simulations, we can generate wake vortices (compare Figures 7 and 8) and thus exceed the critical Reynolds number. However, we still need external stimulation and imbalance in order to generate turbulence and ultimately wake turbulence. We create these by either creating asymmetric obstacles or obstacle combinations, which can be seen in the middle and right part of Figure 8, or omitting the obstacle reflection of an edge of a cell, which is shown in the left part of Figure 8.

The collision term in three-dimensional space

Of course, we have also implemented the new collision and flow steps in the three-dimensional realm. However, we first had to check what number of arrows would work best for our simulations. Because in three-dimensional space, the Latice-Boltzmann method distinguishes between 15-(D3Q15), 19-(D3Q19) and 27-(D3Q27) arrows. We tried the last two models and could not find any significant difference in the simulations. So we opted for the D3Q19 model in favor of the runtime. However, it must be noted that the weighting of the individual arrows must be changed again for a stable simulation. The new weights are shown in Table 2.

Model	W_0	W_1	W_2	W_3
$D1Q3$	2/3	1/6	0	0
$D2Q9$	4/9	1/9	1/36	0
$D3Q15$	2/9	1/9	0	1/72
$D3Q19$	1/3	1/18	1/36	0
$D3Q25$	1/3	1/36	0	0

Table 2: Weighting factors of the different simulation models of the LBM method [17]

With the new program for 3D simulations we can even achieve a speed increase by a factor of 500 compared to the old program.
You can see an example of a three-dimensional flow simulation with wake vortices in Figures 9 and 10.

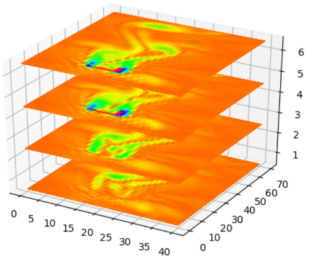


Figure 9: Three-dimensional flow simulation with horizontal wake vortices [61]

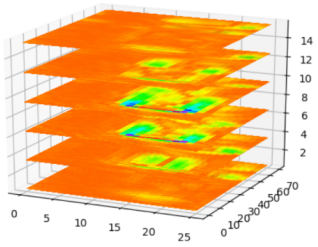


Figure 10: Three-dimensional flow simulation with vertical wake vortices [61]

In summary, we are satisfied with the resulting programs and can use them to calculate the drag coefficients of obstacles.

3.2 Virtual bodies in the program

3.2.1 Preview of handling bodies in the program

Since we want to do 3D simulations with the help of our program, we need a way to deal with bodies. In this case bypass means:

(i) to uniquely define and store them as well

(ii) to perform calculations on them.

Since point (i) should not fill the core part of our work, we decided to use existing software to create bodies in digital form.

3.2.2 Importing bodies and handling file formats

In graphics or in 3D editing programs, bodies are represented using polygon geometry. A body is therefore a polygon compound and consists of flat N-corners. Curves and non-planar surfaces, as they occur in free body geometry, can only be approximated. This approximation is sufficient since we will later convert the geometric solids into boolean solids. They are rendered into a 3D grid, with each cell of space storing the boolean value that indicates whether it is part of the body.
We can create geometric bodies with the help of a 3D editing program. The resulting data on the polygons of the polygon composite can then be exported. Many different file formats are available for this. In order to be able to read such a file format with Python, it must use the ASCII code. A corresponding format is the $.obj$. This is open and can be used without legal restrictions.[4]
A polygon consists of edges and vertices that form its area. We name the edges and the surface and enter them in lists that store the data for all polygons. We interpret the vertices as position vectors. We store these as an array using the Numpy module. We prefer this method to simple tuples because it allows us to later access certain arithmetic operations of the module.
At this point we are able to manipulate the entire body. It is important, for example, to change its orientation in space in order to be able to carry out flows from certain directions later. Once we have made all the necessary adjustments, we need to convert the geometric body into a boolean body. We will go into this step in the following chapter.

3.2.3 Conversion of geometric solids to boolean solids

In the Boolean definition[5] the space is divided into discrete cells which contain the information as to whether they are part of the body. With the division of space into cells, however, it also loses its volume and extent. So the cell size for this data is crucial. A cell thus possesses, so to speak, the information about its extension and position in conventional, Cartesian space. Now, to give it the crucial information that defines it as part of the body, we need to verify that the cell is part of the original body. We do that by determining whether its center is within the body volume.

[4] More information on the $.obj$ file format can be found in the appendix.
[5] Pictures of converted Boolean bodies can be found on page 24 in the appendix.

So we need a function that calculates whether any point within the Cartesian coordinate system is an element of the volume of a polygon composite.
In two-dimensional space there is the so-called beam method.[24] With this method, a ray is defined from the test point. If this ray enters this polygon by crossing an edge of a polygon, it must leave it again by crossing another edge. In the case of concave polygons, the ray can enter the body several times and thus also leave it again several times. The number of times the ray intersects edges of the polygon can be used to determine whether the test point is inside or outside the polygon. An odd number of intersection points means that the ray must have started in the polygon. The test point is therefore within the area of the polygon. No intersection or an even number of intersections means the test point is outside the polygon.
In Figure 11, point H is tested. The ray emanating from it crosses three edges of the polygon: c, e, and f. H lies inside the face.

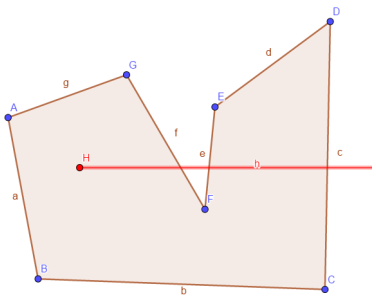


Figure 11: Representation of the ray method for the test point H [61]

We transferred this method to three-dimensional space. For this we define the ray emanating from the test point as a straight line. We replace the individual polygons of the composite with levels. So we determine the number of intersection points of the straight lines with the planes. The following positional relationships can occur between a straight line and a plane:

1. The line and the plane are truly parallel, there is an intersection at infinity.
2. The line is a subset of the plane, there are infinitely many intersection points.
3. The line intersects the plane at exactly one point.

In the first case there is no definable point of intersection. In cases 2. and 3., however, we must use the two-dimensional space method to determine whether the intersection point is inside or outside the polygon representing the plane. The test point itself is used for 2., while for 3. the intersection of the straight line and the plane must first be calculated. For this we use the Gauss algorithm[6]. It is also important that the point of intersection lies in the positive direction of the straight line, so that we can view it as a ray. Again, if the number of intersections is even or no intersection, the test point is inside the body and if the number of intersections is odd, it's outside.
When using this method or methods, we must be aware that intersections with multiple edges or polygons can occur simultaneously at corner points. We must eliminate these multiple increases. In these specific cases, rounding errors also occur, which makes the filtering process more difficult, since there are different intersections at a corner point that differ in decimal places.
Now we can check the centers of all cells for their position in relation to the polygon blend. A Boolean matrix is then created that replaces the geometric body. A cell of the matrix reflects a cell of the previous room. We can now carry out simulations on the resulting Boolean body using the Lattice-Boltzmann method.

3.2.4 Direct import of boolean bodies

In 3D editing programs there is a function that converts the created polygon composite into alternative shapes. One of these forms is the conversion to a cubic lattice. This cube lattice is basically a new polygon composite. However, we can take advantage of this special bond. The cubes already represent a division of space into cells. Bodies of this shape can be exported again as $.obj$ and read out using an alternative function. This does not create lists of edges and faces or position vectors, but immediately the boolean matrix.
In doing so, we treat each vertex of a cube that makes up the body as the center of a cell, since we already know that they are in the body. The disadvantage of this method is that the dicing created by the 3D editing program is supplemented by a layer of cubes in each direction at the edge of the body. Furthermore, the orientation of the body in space cannot be changed afterwards. Each new flow direction therefore requires the import of a newly adapted and modified body.

3.2.5 Graphic representation of bodies

We use the Tkinter module for the graphical user interface of our simulation program. This provides a variety of widgets and can be connected to the Matplotlib module. The latter is used to visualize a wide variety of data with Python and is designed for scientific work. The graphs created with Matplotlib can be embedded in a Tkinter window.
In the program we consider two different types of bodies. On the one hand we deal with geometric bodies consisting of vertices, edges and faces and on the other boolean bodies. The body types require different representations. We visualize geometric bodies using polygon collections and a wireframe, i.e. using their surfaces and their edge grid.

[6] A structogram of the Gauss algorithm can be found in the appendix.

The surfaces are also given a transparency effect. Matplotlib provides a 3D projection for this.[7] A boolean body can be viewed in 2D as an image made up of black and white pixels. In 3D, the pixels are no longer squares but cubes. Matplotlib offers the so-called voxel plot for this purpose. In order to minimize the display volume, we place layers in the room that display 2D sections of the body as an image.

3.3 Basic structure of the simulation program

The program structure is based on the front end, the graphical user interface. The backend, all calculation and simulation functions, are therefore associated with an object on the graphical user interface. The basic building block of the program is the $programinterface()$ class. This inherits from the $Tk()$ class of the $Tkinter$ module we use. Here the graphical interface

1. The entry of a body.
2. Making specific settings.
3. Running a flow simulation based on the settings.
4. Evaluating the project.

According to the explanation above, all four core functions are based on the graphical user interface. Conversely, this means that they each get their own $Frame()$ object. So they are all implemented as an object that inherits from $Frame()$. In this way, your graphical frontend can be initialized and, at the same time, the required functions are implemented as methods of the object that form the associated backend. Since it makes sense to us that functions two and three can be performed multiple times on a body, they are subordinated to another object. We also have objects that serve as information or help windows. Another, separate object is also made for the body processed in a project. This stores body-specific attributes, such as the coordinates of the corner points and information about edges and surfaces. It also has methods that can edit these attributes or derive additional attributes from them. In the program there is therefore a constant change between methods of the graphical user interface and methods of the body object. So when we implement the program, we determine which object a function belongs to. For example, if it edits the vertices of the body, it is a method of the body object and must also be implemented there. However, if a function is to give feedback when a button is pressed on the program interface, it belongs to the respective interface object. This is also the case if the only purpose of this method is to call a function of the body object.
The user thus first gets to the program interface. Here he can start a new project and thus get to the input window of the project. There he can enter the body and continue to the simulation window. A new simulation is created in this window, in which the user first makes all the necessary settings on the settings frame. Once this has been done, he can carry out a simulation on the simulation frame. Now the user can start further simulations with new settings or evaluate the project. Finally, all data collected in the simulations are evaluated in the evaluation window.
Furthermore, we have created a database in which projects can be saved [8]. We have implemented functions that apply to mathematical problems in a separate document, since they are not methods of the body and must be called from several interfaces.

4 Application of our simulation program

4.1 The connection between theory and reality

We can now state that we have created a realistic flow simulation of particles around bodies. However, the actual goal of the seminar work, the calculation of the Cw values of the bodies in the flow, has not yet been dealt with in more detail. In general, the Cw value is a variable value depending on the Reynolds number. The Reynolds number is a dimensionless number named after the physicist Oswald Reynold and indicates the ratio of inertial and viscous forces in a flow. (37)[13]

[7] In a previous version of our program, we projected geometric bodies onto a two-dimensional plane. The corresponding conversion of the coordinates can be found in the appendix.

[8] The structure of the database can be seen in the appendix.

$$Re = \frac{\rho \cdot v \cdot d}{\mu} = \frac{v \cdot d}{\nu} \tag{37}$$

The dependence of the Cw value on the Reynolds number is illustrated in Figure 12.
It becomes clear that the Cw value can vary greatly with the Reynolds number. It only levels off at a constant value at a Reynolds number between 1000 and 100,000.

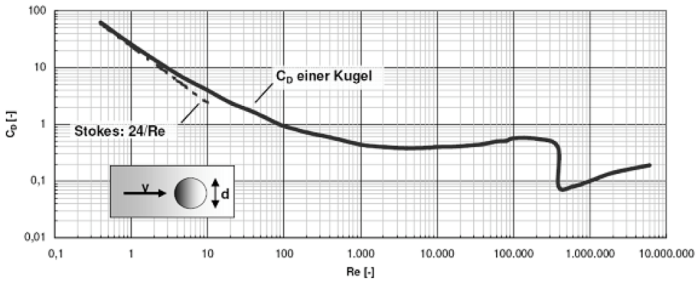


Figure 12: Cw value of a sphere depending on the Reynolds number [62]

This is the area in which the drag coefficients of bodies are calculated in wind tunnels today.
For these Reynolds numbers, the Cw values of common bodies have already been measured and are generally accepted. The sphere has a drag coefficient of 0.47, the cube 1.05 and the disc 1.1.
In order to be able to classify our results of the later Cw value calculation, we have to make sure that our simulations work with Reynolds numbers in the mentioned range.
In general, the Reynolds number is defined by the formula shown above and can be calculated in practice. However, we have no units in our simulation and can only calculate ratios from the viscosity and speed, which can be varied in the program. Other approaches must therefore be chosen to determine the Reynolds numbers used. On the one hand, we have made use of the fact that laminar and turbulent flows are distinguished by a critical Reynolds number. This critical Reynolds number varies depending on the body and flow space. As an approximation, the critical Reynolds number can be set at around 2500. So we have calculated a theoretical Reynolds number from the relationship between viscosity and velocity in the program, neglecting the characteristic length d. Ultimately, we were able to determine the critical Reynolds number to the theoretical Reynolds number of 18 that we calculated. Accordingly, with a theoretical Reynolds number of 18, which in reality corresponds to a Reynolds number of 2500, we would be between the desired range of 1000 to 100,000.

This ratio would be suitable for calculating the drag coefficient.
In order to check this result, we calculated the Cw value of a sphere depending on several theoretical Reynolds numbers. This resulted in the graph in Figure 13, which is almost identical to the graphs of a cube and a cone[a].

If one compares the diagram from Figure 13 with the diagram from Figure 12, the section can be classified almost identically in the real Reynolds number range from 1000 to 5000. In the diagram (Figure 13), the theoretical Reynolds number of 20 corresponds to the real Reynolds number of 2500 and the result of the critical Reynolds number is confirmed.

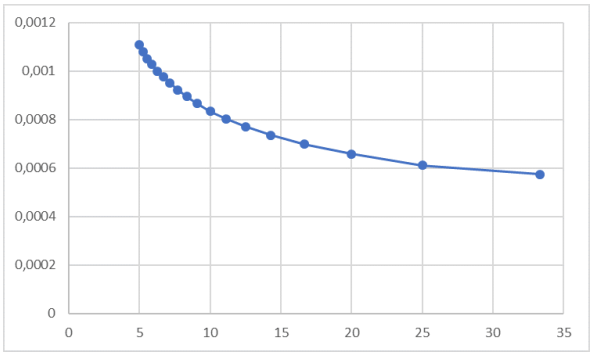


Figure 13: Cw value of a sphere depending on the Reynolds number [61]

[a] The corresponding graphs can be found in the appendix on page 24.

4.2 Optimization of the Cw value calculation

A diagram with the calculation results of the Cw value is already shown above. How exactly this calculation works is explained below. The Cw value is generally defined by the formula (38): [19]

$$c_W = \frac{F_W}{q \cdot A} = \frac{2 \cdot F_W}{\rho \cdot v^2 \cdot A} \tag{38}$$

In our simulation, we calculate the force F, which acts on the body in the direction of flow, by summing up all the directional arrows from the surrounding grid cells of the body in the direction of flow. We used the D3Q19 Lattice Boltzmann model. Density is determined by summing all the directional arrows of each cell in the entire space and dividing it by the number of cells in the space minus those in the body. Furthermore, the area is determined by the number of cubes directly flown by the body and the speed can be set in the program. This allows all the variables required for a correct Cw value calculation to be accessed using the formula mentioned above.
Then you can calculate the theoretical Cw value for each body, compare the result with the values from reality and adjust the theoretical results by a general factor a so that they are as close as possible to the real Cw values. Before doing this, however, it must be checked that all Cw value calculations are carried out with the various bodies under the same conditions. This is the only way to define a general factor a that applies to all theoretical results. To ensure

this, the Reynolds number must be the same for each simulation. On the one hand, we guarantee this by keeping the relationship between viscosity and speed constant in the program. On the other hand, while determining the critical Reynolds number, we also had to realize that the Reynolds number and thus the Cw value depend, among other things, on the scaling and the positioning of the body in space. This relationship can be explained by the dependence of the Reynolds number on the characteristic length d of the body.

Because the greater the length of the space in the direction of flow, the smaller the critical Reynolds number. The Reynolds number levels off at a constant value from a spatial length behind the body and in the direction of flow that corresponds to six times the three-dimensional maximum extent of the body. The critical Reynolds number already mentioned above was determined with a factor of 6. The relationship mentioned is illustrated again in Figure 14. The maximum extent is described by the distance between the minimum and maximum X, Y and Z coordinates.

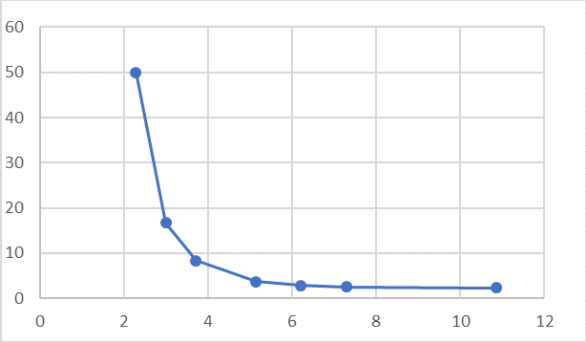


Figure 14: Dependence of the critical Reynolds number on the maximum extent of space [61]

Accordingly, we use a factor of the maximum expansion of 6 for each body in the program, because this means that the Reynolds number always remains almost constant, even with different body shapes.
In order to make the Cw value calculation less error-prone, we have shortened the formula for the Cw value to $c_W = \frac{F}{A}$. Because the adjustable Reynolds number with the viscosity and speed remains the same for every calculation. For the density, too, we have proven through calculation results between different bodies that it is independent of the body and only deviates by an average of 0.4% from the mean. The slight deviation can be attributed to an inaccurate calculation of the volume of the bodies. With the exception of the area A and the force F, all variables of the drag coefficient formula are constant and are omitted. Since turbulence occurs in the flow simulation with higher Reynolds numbers, which causes the Cw value to fluctuate almost periodically depending on the flow step, we calculate the resulting Cw value from the average of the Cw value between flow step 1000 and 1400. The fluctuations are in Figure 22 shown on page 25 in the appendix. To determine which Reynolds number gives the best Cw value results, we performed calculations using various theoretical Reynolds numbers and twelve simple bodies whose Cw values can be found in the literature. With our program we can cover theoretical Reynolds numbers between 5 and 50, which in reality reflect Reynolds numbers between 700 and 7000. We created various diagrams with drag coefficients. It turned out that the general factor a between the theoretically calculated drag value and the real drag value of a body is not constant, but represents a linear function. (compare figure 15)

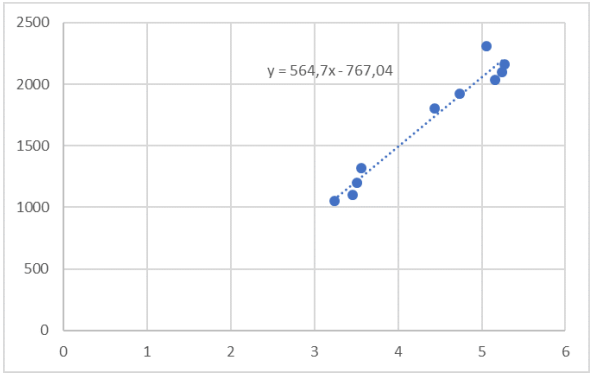


Figure 15: Factor a depending on the calculated drag value [61]

We found the smallest deviation from the factor a to the straight line at a cube accuracy of 14 with a theoretical Reynolds number of 13. With a cube accuracy of 20, the minimum deviation was at a theoretical Reynolds number of 9 after more than 70 hours of calculation. With larger Reynolds numbers we exceed the limit of the critical Reynolds number 18 and the turbulence increases. As a result, the fluctuations in the Cw values between the individual flow steps also increase and the Cw value results become inaccurate. In the case of smaller theoretical Reynolds numbers, inaccurate Cw values occur, since one no longer calculates in the Reynolds number range between 1000 and 100,000 and the Cw value results therefore no longer represent the Cw values already measured in the literature.

4.3 Results of the drag coefficient calculation

4.3.1 Results for simple bodies

After we had optimized the Cw value calculation as best as possible, we calculated the Cw values of 12 simple bodies with different cube accuracies. With an accuracy of 14 dice, we were able to achieve an average deviation of 4.4 %, which is illustrated again in Table 3. [16, 17, 12, 32]

Body	calculated Cw value	real Cw value
Cone	0.344	0.340
Hollow sphere	0.411	0.380
Hemisphere	0.426	0.420
Sphere	0.443	0.470
Cube (rotated 45 degrees)	0.773	0.800
Cylinder	0.905	0.910
Hemisphere (opposite)	1.058	1.170
Cube	1.108	1.050
Disc	1.149	1.100
Cone (opposite)	1.164	1.140

Table 3: Calculated Cw values in comparison with values from the literature [61, 16, 17, 12, 32]

With a dice accuracy of 20, the average deviation is 5.1%. In the calculations, the simple bodies were each flown centrally from the front. In order to show the dependency of the angle of attack on the Cw value of the body, we have created an illustration of the Cw value of a cube and a hemisphere depending on the angle of attack in Figures 23 and 24 in the appendix. It is striking that the theoretically calculated Cw values are highly error-prone for extremely large and small values. In real terms, a streamlined body has a drag coefficient of 0.04. Our program, on the other hand, calculates a Cw value of 0.6, which is greater than the Cw value of a sphere. This is where the simulation reaches its limits, which are explained in more detail in the error analysis. We also find a similar limit with extremely large drag coefficients, such as that of a semi-hollow sphere oriented against the flow. This is normally 1.4. Our simulation, on the other hand, calculates a value of 0.9.

4.3.2 Lamborghini results and scanned body

Since the beginning of our work, we have made it our main task to calculate the drag coefficient of a Lamborghini. We obtained the Lamborghini's $.obj$ file from the Internet free of charge. Using Blender, we edited the file so that our program can dice it. Unfortunately, we were not able to record any representative results in the Lamborghini's Cw value calculation compared to the real value of 0.33. We explain this by saying that the Lamborghini has a small inflow area compared to the entire surface of the body. However, we will go into more detail about this in the error analysis. In order not to completely ignore our Lamborghini, we also compared the flow pattern calculated by our software with the flow pattern of a Lamborghini from the wind tunnel. The two flow images look almost identical and speak for the underlying simulated flow. The comparison is shown in Figure 27 in the appendix.
So that we can also calculate other and, above all, more complex bodies, we have procured a 3D scanner compatible with Apple tablets from the student research center in Erfurt. This scans the bodies we want and converts them into compatible $.obj$ files. The $.obj$ files can then be aligned in the flow direction using the Blender graphics software and, if necessary, corrected for defective parts. It should be noted that the 3D scanner is hardly subject to any restrictions. Both large objects such as cupboards and tables and small objects such as a pencil case or computer mouse can be scanned. Our program can then calculate the drag coefficient. However, this time there is no real comparison value available from the literature, because we used bodies such as a nutcracker or a cup, among other things. These have not previously been measured in a wind tunnel. That's why we used the wind tunnel in the student research center to carry out our own Cw value measurements with the scanned bodies.
In order to be able to classify our measurements later, we first determined the Reynolds number used in the wind tunnel. Depending on the size of the body, we were able to determine an average range between 20,000 and 50,000. This is also in the range of the Reynolds numbers with a constant Cw value and the measurements from the wind tunnel can be compared directly with our theoretical calculations. We have briefly summarized the results in Table 4 in the appendix on page 30. There we continued to explain our measurements, how the wind tunnel works and the accuracy of the results more explicitly.
For the complex bodies, we can record an average deviation from the measured Cw value of 11.4%. It should be noted here that the wind tunnel has an average error rate of 15%. Based on the deviations of simple bodies, our software works with 5% more precisely than the wind tunnel. Accordingly, we can only check that the drag coefficient we calculated is roughly in the range of the value measured in the wind tunnel, plus minus 20% deviation. This is fulfilled for every calculated drag coefficient. We conclude that our software is also suitable for complex bodies, taking into account the limitations in error analysis. With the wind tunnel it is also possible to visualize the flow pattern around the body. We recorded this with several camera images for the bodies. We then compared the images with the images from our own simulation of the respective body. The pairs of flow images of a body look almost identical, which is illustrated in Figures 28 and 29 in the appendix. Together with the Lamborghini's flow pattern, this underscores the accuracy and realism of the simulation we developed.

4.4 Error analysis for the simulation method we developed

As has already become clear in the results for the simple bodies, our Cw value calculation reaches its limits with extremely large and small Cw values. However, there were also gross errors in the Lamborghini's drag coefficient. We examined the causes of errors in other, self-created bodies and were able to identify two significant sources of error. In the chapter Optimizing the Cw value calculation, we already mentioned that the Reynolds number and thus the Cw value depend on the scaling of the body in the flow space. From a room length based on six times the maximum extension of the body, the Reynolds number has leveled off at a constant level. However, for bodies that are very long in one of the dimensional directions perpendicular to the direction of flow, this approximation no longer seems to work. A square disc normally has a drag coefficient of 1.1. Our software calculates this as 1.15. However, if we stretch the disk perpendicularly to the direction of flow Y, for example in the X direction, we already have a Cw value that is twice as large with a ratio of three between the X length and the Z length of the body. This relationship is illustrated again in Figure 26 in the appendix. The second source of error arises from the relationship between the inflow area and the total surface area of the body. When determining the force acting on the body, we determine the difference of the five directional arrows acting on the body in the X direction by default. The central arrow in the X-direction can only act on the inflow surface. However, the four oblique arrows in the X-direction can act on a body surface parallel to the direction of flow. If the total area of the body is larger, but the inflow area remains constant, more oblique arrows can affect the body. The final sum of the forces acting on the body is thus greater, although the inflow area remains constant. Ultimately, this process results in an increased drag coefficient. We tried to calculate the force only by the central force in the X-direction, but these results were even less representative. To show this connection, we used a disk and continuously extended it in the Y direction. The course of the Cw value is shown graphically in figure 25 in the appendix. The drag coefficient of a cube is normally 1.05. If you lengthen it by a factor of 3.3 in the Y-direction, so that the ratio of the inflow area to the surface is 15, the Cw value calculated by our program is almost twice as much. Ultimately, this also explains the Cw value of the Lamborghini and the flow body, which was incorrectly calculated by our program. The Lamborghini has a ratio between the faces of 16 and the ratio between the X and Z dimensions of the body is 1.6. However, for the program to calculate the correct Cw value, based on the two charts, the area ratio should not be greater than 11 and the ratio between X and Z dimensions should not be greater than 1.8.

4.5 Result classification

In conclusion and in summary, we are satisfied with our results. With our program we have developed a realistic and clear flow simulation for bodies, with which the flow behavior of a body can be examined in a user-friendly manner. Our animation of the flowing fluids is in no way inferior to the flow images in the wind tunnel. On the contrary, it is even more descriptive and informative since it includes multiple perspectives and different forms of representation. In the wind tunnel, the depiction of flow is limited to a few colored flow stripes that run around the body. Furthermore, we can achieve extremely precise calculation results, which have an error rate of only 5% in the range of Cw values between 0.25 and 1.2. Inaccuracies occur outside of this range. The software should not be used directly here. The conditions mentioned in Chapter 4.4 must be taken into account. Our software cannot yet completely replace the wind tunnel.
Furthermore, our software is practical. With a 3D scanner you can simply scan in the desired objects, align them with Blender and have our software calculate the drag coefficient. If the objects do not exist physically but only on the computer in $.obj$ format, they can still be imported. Ultimately, using our software is a lot more efficient than using a wind tunnel. On the one hand, our software is more accurate in this area than the wind tunnel we tested at the student research center, which is one of the highest-quality wind tunnels that is privately available. On the other hand, it is also more time-saving, because scanning and calculating the Cw value takes a maximum of 30 minutes. Based on experience, scanning the body and aligning it with Blender takes an average of 5 to 10 minutes. For the calculation of the Cw value, around 7 minutes are to be planned for a cube accuracy of 14 and around 20 minutes for a cube accuracy of 20. Finding a wind tunnel that is the right size and with the desired level of accuracy can take days and can quickly cost hundreds of thousands. So we have developed a complete solution that anyone can use to determine the drag coefficient of a body easily, quickly and with little effort. Due to the flexibility of the 3D scanner, both large and small objects can be calculated that would not fit in the wind tunnel or are too small for it. This makes our complete solution suitable for almost everyone, from simple hobbyists at home to physics teachers in class.

5 Reflection on our work

The aim of our work was to develop software for the flow simulation of fluids around bodies, which enables the Cw value of defined bodies to be calculated. We asked ourselves how representative the theoretical results are and to what extent it is still necessary today to use a wind tunnel to calculate the drag coefficient.
We dealt with the basics of fluid mechanics, especially the Navier-Stokes equation, and went into mathematical models and their components and operations. Based on the Boltzmann equation and the Lattice-Boltzmann method, we have created a program for flow simulation in two-dimensional and three-dimensional space. For this we implemented methods for reading and dicing of bodies, which allow us to work with official file formats. With the help of this program complex and an extensive GUI[9], fluid-mechanical data of a flow can be collected, based on which we can approximate the flow resistance coefficient of a body. It is also possible to understand the flow through diagrams and animations. Based on several bodies for which the Cw value has already been measured and is recognized, we have optimized the Cw value calculation and were ultimately able to achieve an average deviation from the real values of around 5%. We then applied our calculations to scanned bodies and compared them with the wind tunnel. We were able to show that our simulation achieves better results than the wind tunnel, but we also explained where our simulation reaches its limits.

In summary, we can say that we have achieved our goals and that we are very satisfied with the results of our work. We gained a deep insight into the field of fluid dynamics and multiplied our program development skills. The working atmosphere in our group was always constructive and motivating. We have consistently adhered to our schedule and hereby conclude our seminar paper.

[9] Graphical User Interface

6 Appendix

Trajectory of a volume element for the funnel flow example

Here we set up the path equation of a volume element for the funnel flow. The trajectory of the element runs parallel to the velocity field. This results in the following relationship for $(\varrho_0, \varphi_0, z_0)$ in the cylindrical coordinate system (39):

$$\begin{pmatrix} \varrho - \varrho_0 \\ \varphi - \varphi_0 \\ z - z_0 \end{pmatrix} = t \cdot \begin{pmatrix} \varrho_1 - \varrho_0 \\ 0 \\ z_1 - z_0 \end{pmatrix} \tag{39}$$

$(\varrho_0, \varphi_0, z_0)$ is at the bottom of the funnel and $(\varrho_1, \varphi_1, z_1)$ is the the starting point of the trajectory at the top of the funnel. For these sizes, the following ratio (40) applies:

$$\frac{\varrho_1}{\varrho_0} = \frac{h}{H} \tag{40}$$

In order for the path to run on a vertical plane, ((41) and (42)) must also apply:

$$\varphi_1 = \varphi_0 \tag{41}$$

$$\varphi = \varphi_0 \tag{42}$$

If you write out the first equation, you get 6 equations that describe the path of a volume element through the point $(\varrho_0, \varphi_0, z_0)$.

Projection of three-dimensional coordinates in two dimensions

The projection of three-dimensional bodies onto a two-dimensional surface was part of our program before we decided to use $Matplotlib$ as a display module. There are many different methods for this process, which we will not go into further here. It was important for us that the projection was efficient in terms of runtime. So the easiest way would be to set all z-coordinates to zero, for example. This can be achieved by overriding the Z coordinates. However, this solution is not particularly elegant. A better way is to multiply a matrix by the point. This matrix can be adjusted using variables and can therefore also be used to automatically zoom the projected points. By multiplying them by a factor, their distance from each other can be changed. In addition, the projection is not limited to the XY plane alone. For example, to represent a three-dimensional point on the YZ plane, the calculation would look like this (43):

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0 + Y + 0 \\ 0 + 0 + Z \end{pmatrix} = \begin{pmatrix} Y \\ Z \end{pmatrix} \tag{43}$$

Structure of Wavefront 3D object files

```
# Blender v2.79 (sub 0) OBJ File: ''
# www.blender.org
mtllib Cube_fine.mtl
o Cube_Cube.001
v -4.197928 -2.718778 5.815319
v -4.197928 7.281222 5.815319
v -4.197928 -2.718778 -4.184681
v -4.197928 7.281222 -4.184681
v 5.802072 -2.718778 5.815319
v 5.802072 7.281222 5.815319
v 5.802072 -2.718778 -4.184681
v 5.802072 7.281222 -4.184681
vn -1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 1.0000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
usemtl None
s off
f 1//1 2//1 4//1 3//1
f 3//2 4//2 8//2 7//2
f 7//3 8//3 6//3 5//3
f 5//4 6//4 2//4 1//4
f 3//5 7//5 5//5 1//5
f 8//6 4//6 2//6 6//6
```

Above you can see the *.obj* document of a cube. On the right are the lists for the data of the polygon compound as we use them in our program. The first list contains the individual vertices of the body. Each vertex is stored by a name and its location vector. We store the position vector as an array object of the Numpy module. The second list contains the edges and the third list contains the faces of the body. Also they are stored by a name. It also includes a list of the names of the vertices they define.

```
[['P0', array([[ 0.],
       [ 0.],
       [10.]])],
 ['P1', array([[ 0.],
       [10.],
       [10.]])],
 ['P2', array([[0.],
       [0.],
       [0.]])],
 ['P3', array([[ 0.],
       [10.],
       [ 0.]])],
 ['P4', array([[10.],
       [ 0.],
       [10.]])],
 ['P5', array([[10.],
       [10.],
       [10.]])],
 ['P6', array([[10.],
       [ 0.],
       [ 0.]])],
 ['P7', array([[10.],
       [10.],
       [ 0.]])]]
[['K0', ['P2', 'P0']],
 ['K1', ['P0', 'P1']],
 ['K2', ['P1', 'P3']],
 ['K3', ['P3', 'P2']],
 ['K4', ['P6', 'P2']],
 ['K5', ['P3', 'P7']],
 ['K6', ['P7', 'P6']],
 ['K7', ['P4', 'P6']],
 ['K8', ['P7', 'P5']],
 ['K9', ['P5', 'P4']],
 ['K10', ['P0', 'P4']],
 ['K11', ['P5', 'P1']]]
[['F0', ['P0', 'P1', 'P3', 'P2']],
 ['F1', ['P2', 'P3', 'P7', 'P6']],
 ['F2', ['P6', 'P7', 'P5', 'P4']],
 ['F3', ['P4', 'P5', 'P1', 'P0']],
 ['F4', ['P2', 'P6', 'P4', 'P0']],
 ['F5', ['P7', 'P3', 'P1', 'P5']]]
```

Wavefront 3D object files are used to store geometric properties of objects. Objects can be individual bodies but also body accumulations. The corner points and the surfaces of objects are saved. A normal is also specified for each surface. The normal is known from vector calculation and points out of the object here. The information stored in the *.obj* file is read line by line. Lines storing vertex data are denoted by the letter v. The position vector for a point is specified in a Cartesian coordinate system. Normals are marked by vn and the faces by f. A surface consists of at least 3 points. Point numbers are given based on their order in the document. The normal of the surface is specified for each point, also via its number. Point and normal number are separated by a double slash. Individual entries in a line are separated by a space. In the document excerpt above, the eight vertices of a cube with edge length 10 and the six normals and associated faces are stored.
However, there are other lines in the document that are irrelevant for our purposes. Lines one and two start with a hashtag and are therefore comments. The fourth line starts with an o. After that is the name of the saved object. This has nothing to do with the file name. It is useful when working with several objects at the same time in a 3D editing program and thus serves the user-friendliness. After a g there would be the name of a group, i.e. body aggregation. The letter s, found in the document in front of the faces, marks the smoothing of the object. In this case it is disabled. For example, it plays a role in curved bodies when it comes to 'smoothing' the surface approximated by polygons. Finally, the *.obj* file refers to a material library with the file extension *.mtl*. It is identified by $mtllib$. After that is the path to the file. However, in the example above, the noted file is marked as not needed by the line $usemtlNone$. Furthermore, textures of the body can be specified in the document, marked by the beginning of the line vt.
On the right in the document excerpt you can see the translation of the file for our program. We must note that we start with zero when numbering the points, edges and faces. In addition, the coordinates of the corner points are adjusted in such a way that the smallest coordinate per dimension is 0 in each case. This avoids calculating with rounding-prone floating-point numbers, which in turn prevents rounding errors by the program. Finally, you can see that the corner points of an area are already specified in the *.obj* document. This allows us to determine the edges of the body. Just make sure to avoid duplication.

Structure of the database for saving projects of our simulation program

A database can be implemented in Python using the *sqlite*3 module. Our database consists of two main tables and associated side tables. The first main table is called *IDs* and is used to store the IDs used in the database. With their help we can create new IDs automatically and make sure that there is a unique mapping between the tables. We do not use the names of the bodies for this, since duplication of names can occur at this point. Projects edited in the program can now be saved in the second main table called *Projects*. An entry consists of the created identification ID, the name of the project and the body used and a path to the *.obj* file of the body. The ID of the project is still used as the name of the simulation table belonging to the project. All simulations that were carried out in the course of the project are saved here. A simulation entry also receives an automatically generated ID. The simulation name and the simulation settings are also part of the entry. Finally, the data collected in the simulation is missing. A measurement table is created for them with the simulation ID as the name.
Since the last states of all simulations are to be visualized when a project is opened, the associated variables must also be saved. These are the arrow density for each cell and the arrow direction sum per cell for each dimension. When storing this data, we must keep in mind that it is a 3D matrix that must be entered into a two-dimensional table. Thanks to the *reshape*() function of the Numpy module, this can be reversibly converted into a one-dimensional matrix. This can now be entered in a column of any length in the database table. The table created for this bears the ID of the associated simulation as a name, just like the measured value table, but with an additional abbreviation.

Graphics and visualizations

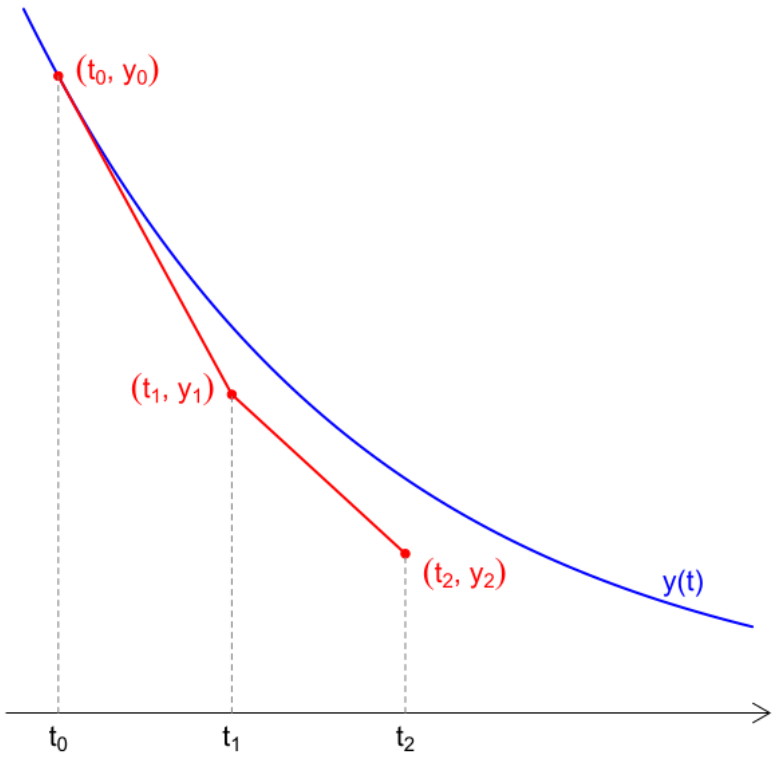


Figure 16: Model of the explicit Euler method [26]

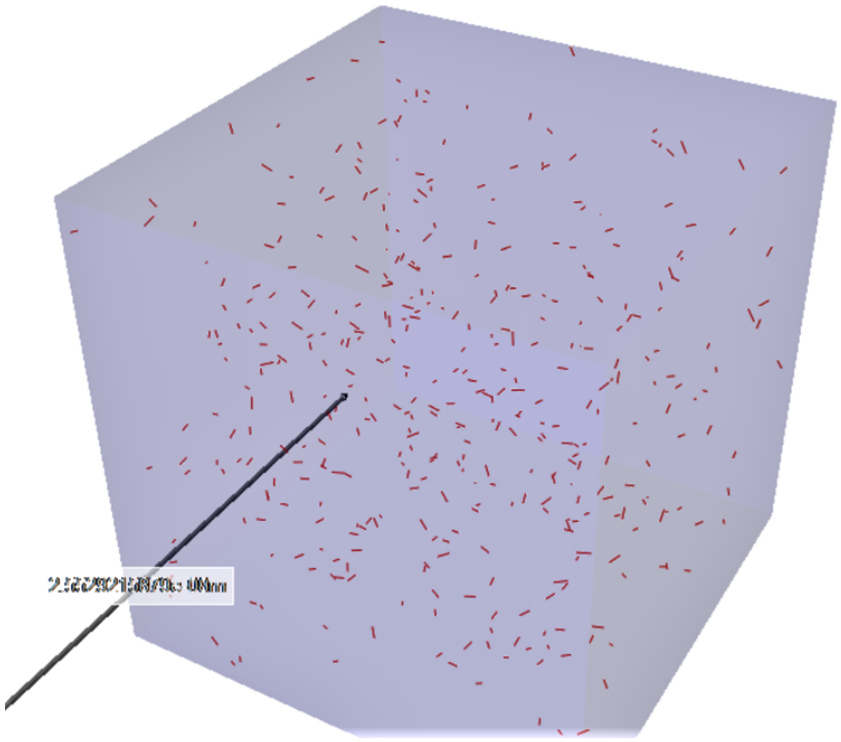


Figure 17: Presentation of the microscopic simulation method [61]

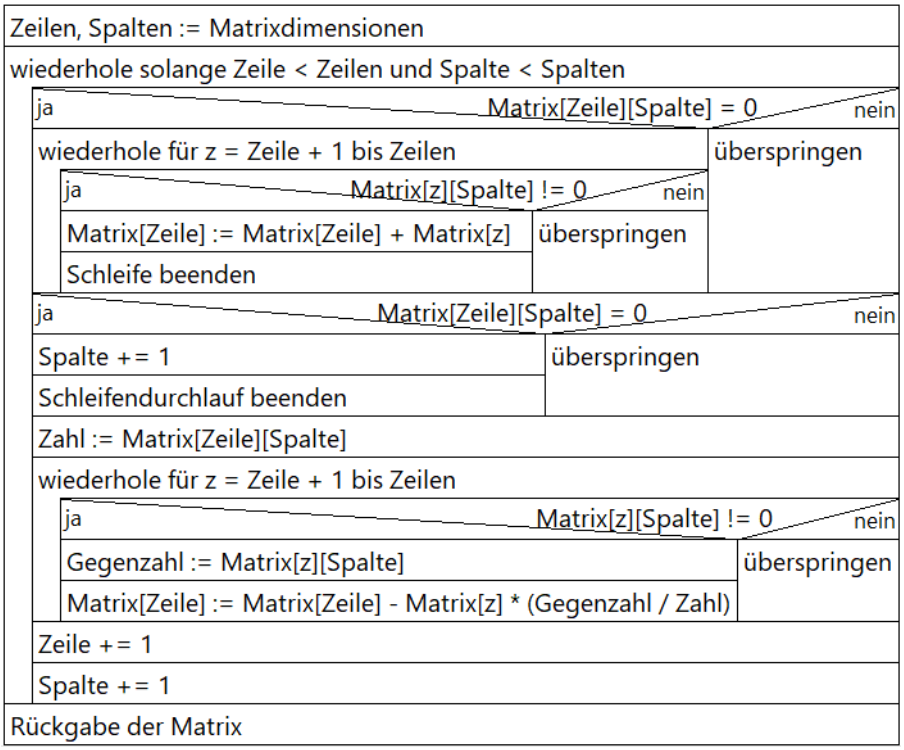


Figure 18: Structogram for the Gauss algorithm [61]

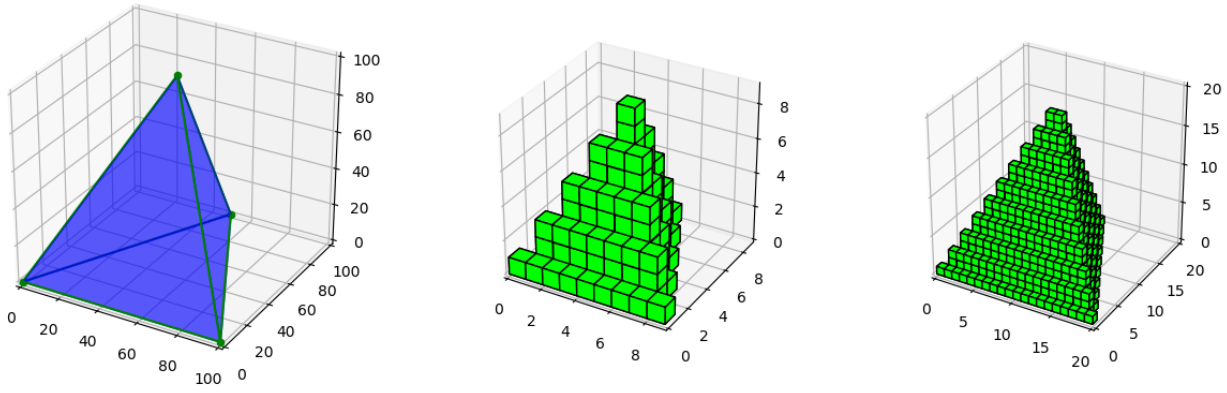


Figure 19: Conversion of a geometric body into a boolean body with our program [61]

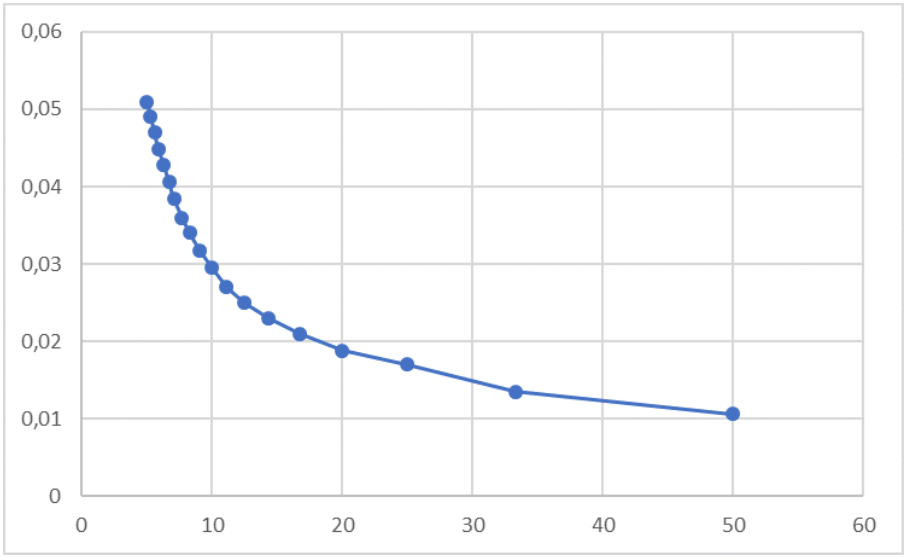


Figure 20: Determined Cw value of a cube depending on the Reynolds number [61]

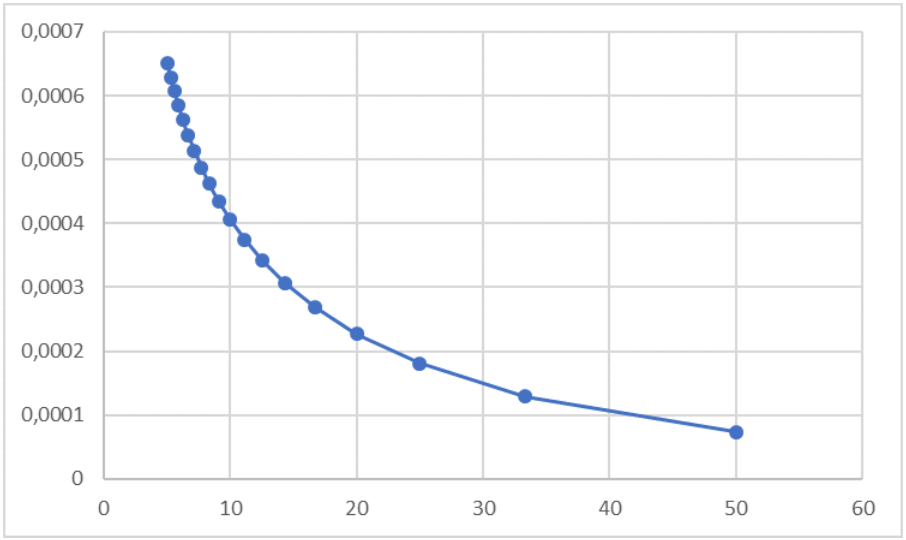


Figure 21: Determined Cw value of a cone depending on the Reynolds number [61]

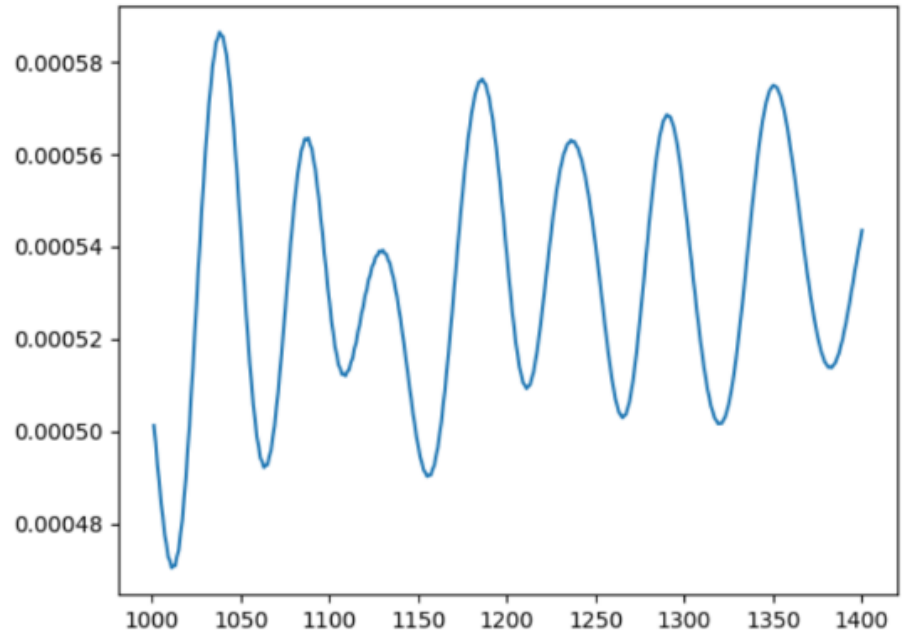


Figure 22: Fluctuations in the Cw value depending on the simulation step for a disk in a flow [61]

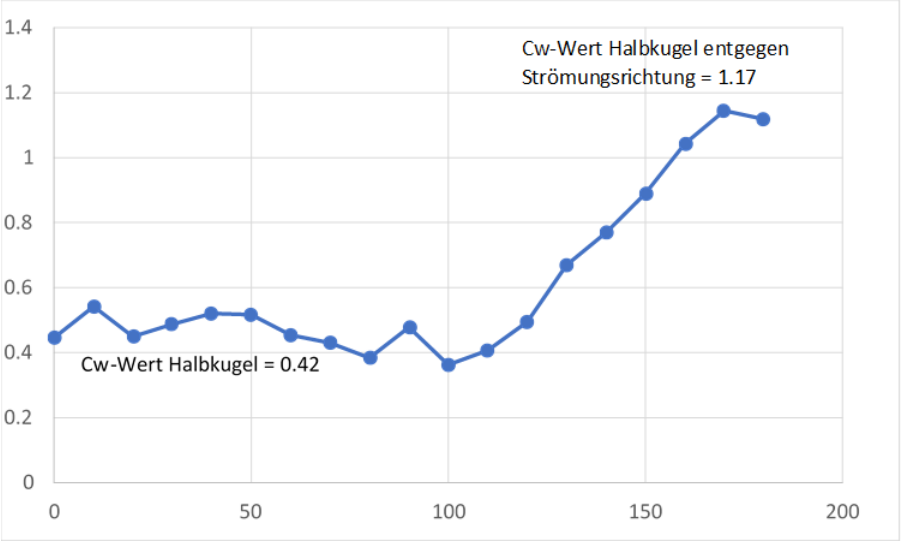


Figure 23: Cw value of a hollow hemisphere depending on the angle of attack [61]

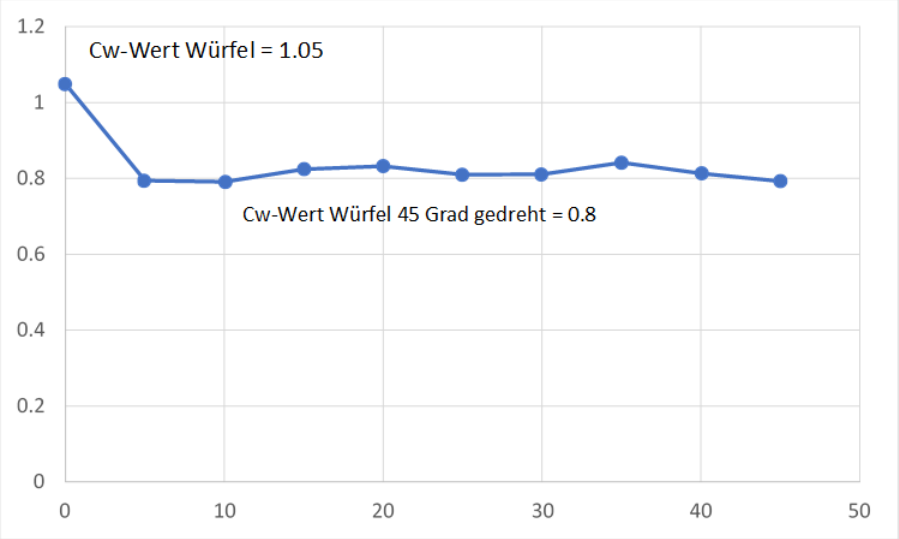


Figure 24: Cw value of a cube depending on the angle of attack [61]

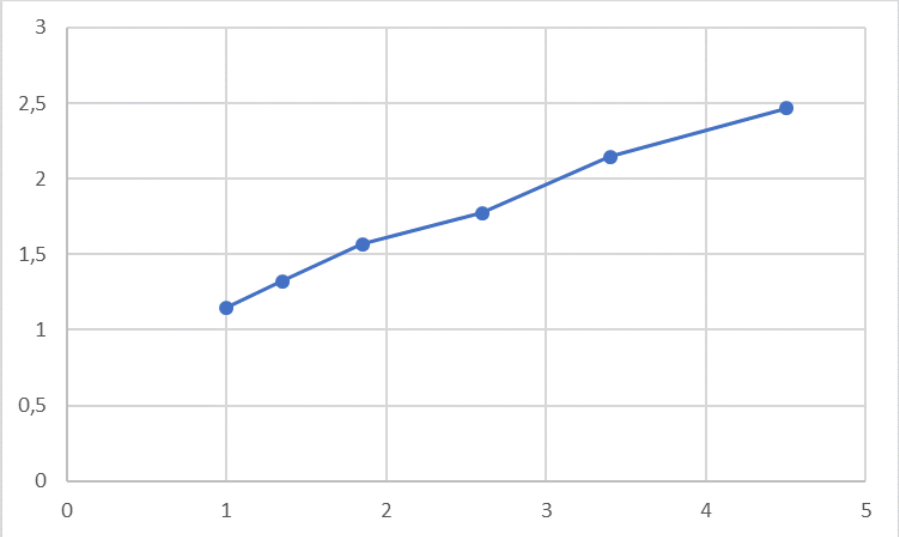


Figure 25: Cw value depending on the inflow area to the surface of the body [61]

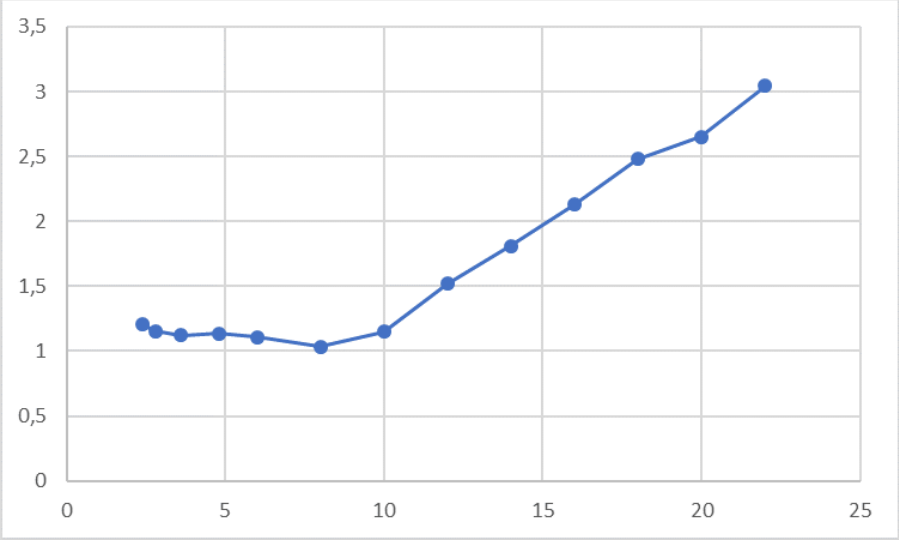


Figure 26: Cw value as a function of the ratio of X expansion to Z expansion of the body [61]

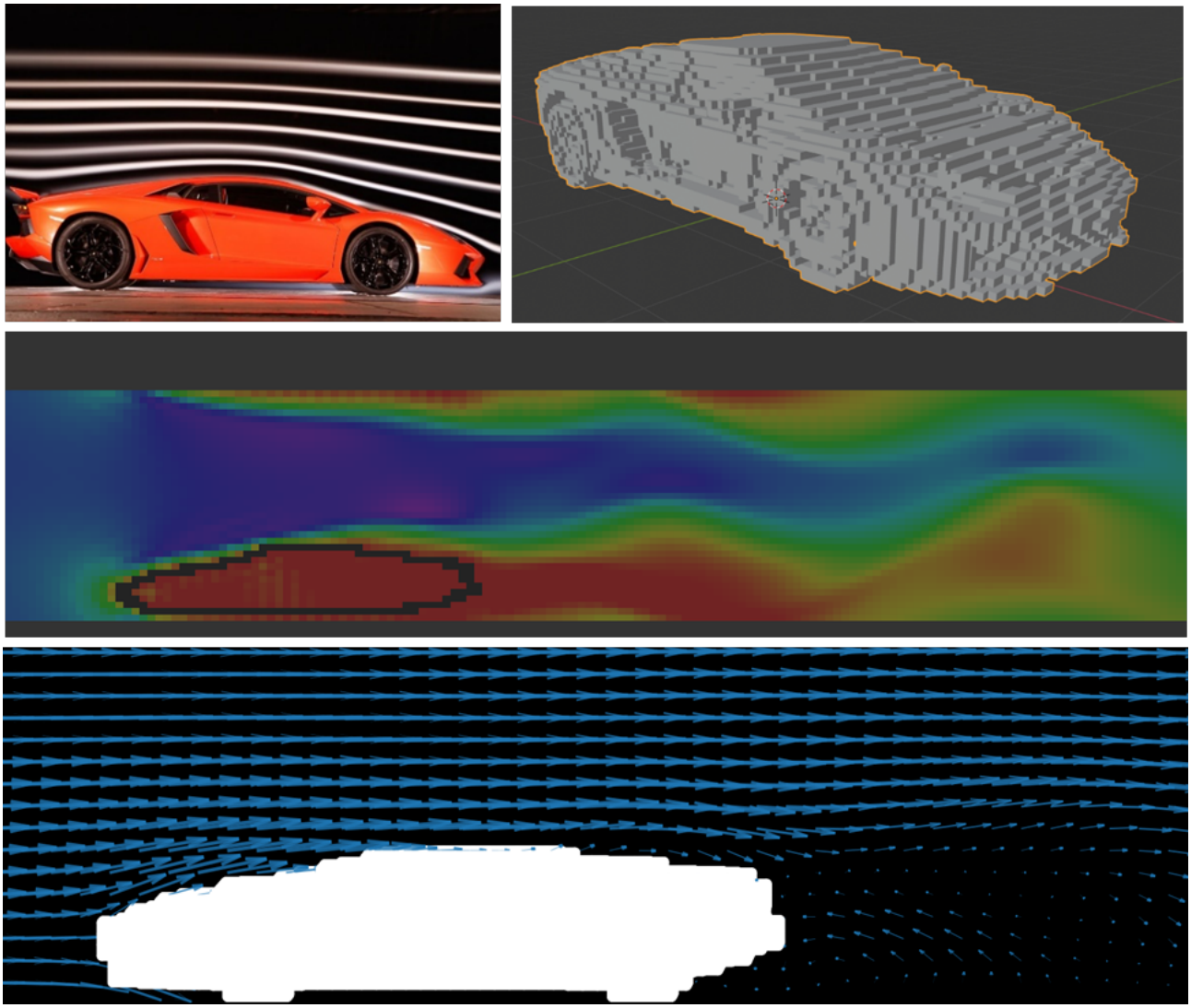


Figure 27: Comparison of our flow simulation around a Lamborghini with the flow visualization in a wind tunnel [61, 60]

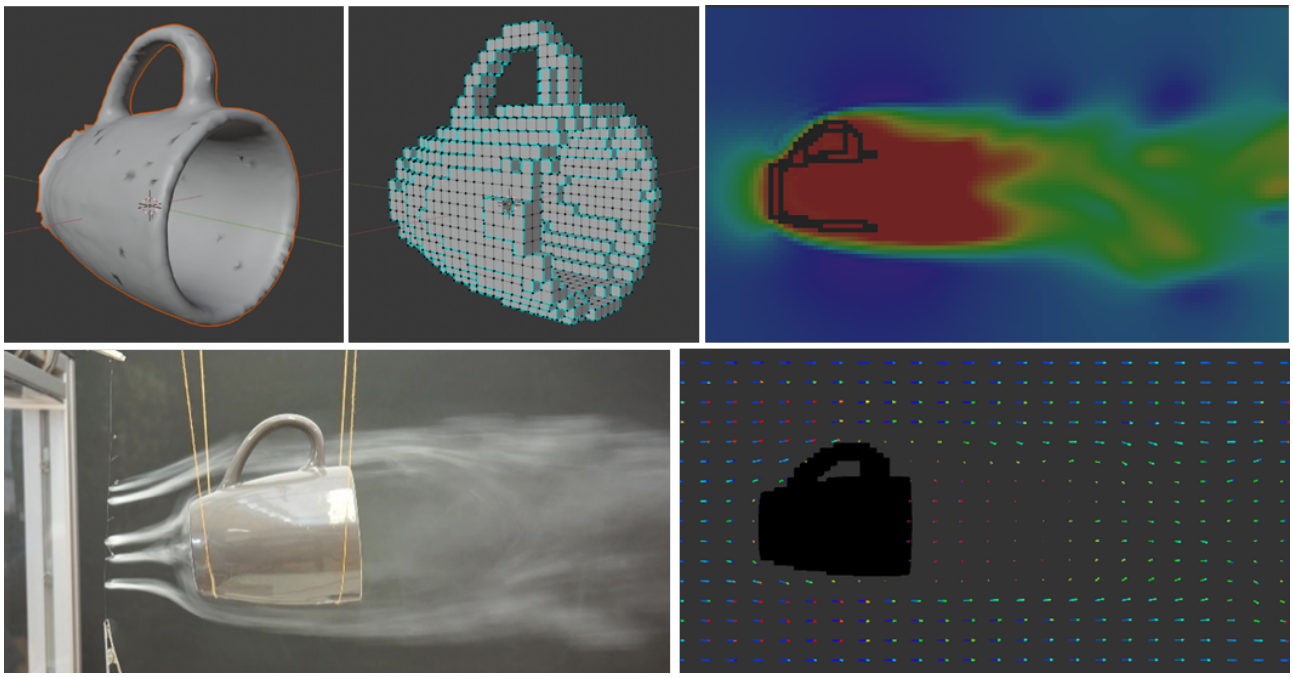


Figure 28: Comparison of our flow simulation around a cup with the flow visualization in a wind tunnel [61]

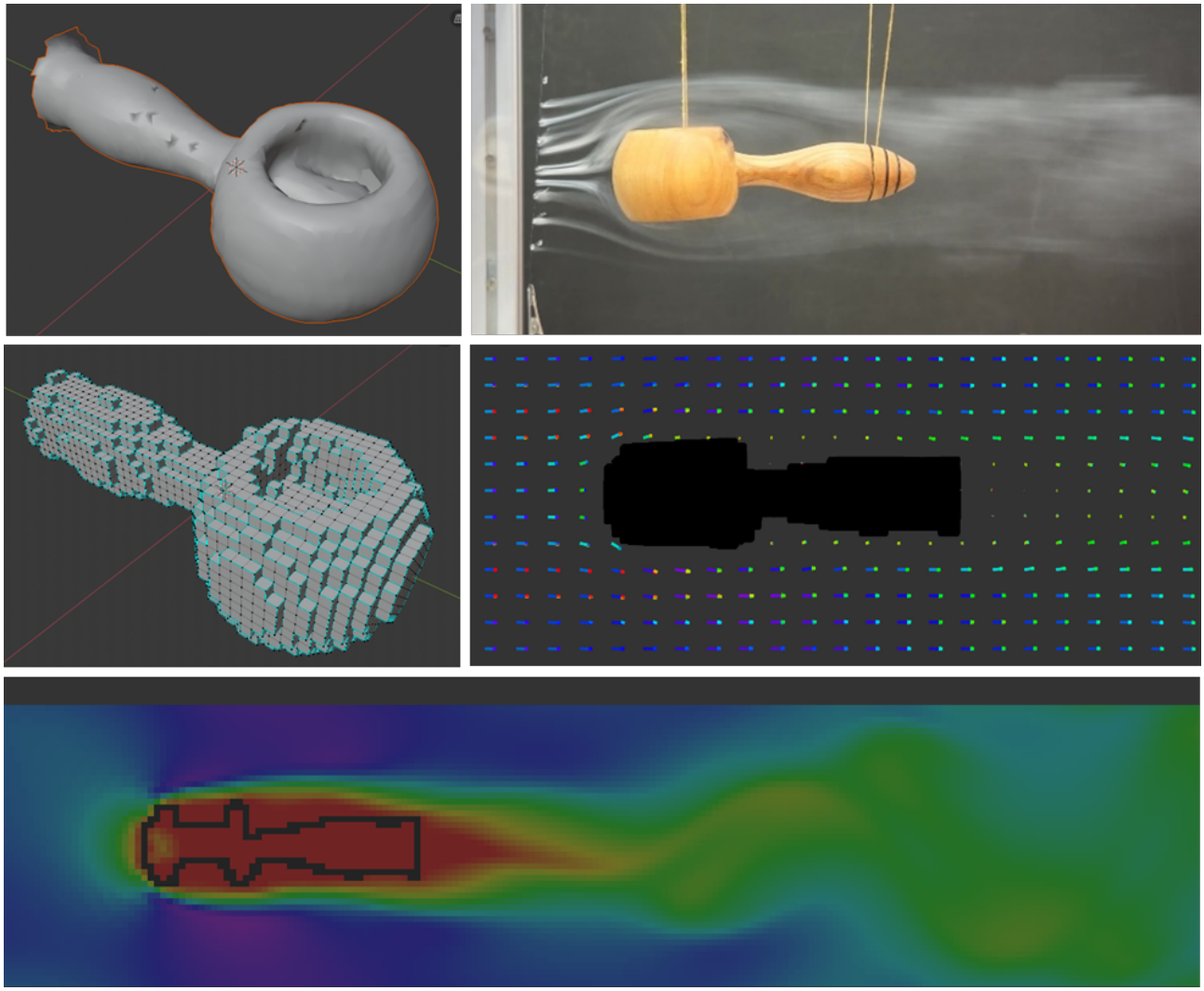


Figure 29: Comparison of our flow simulation around a nutcracker with the flow visualization in a wind tunnel [61]

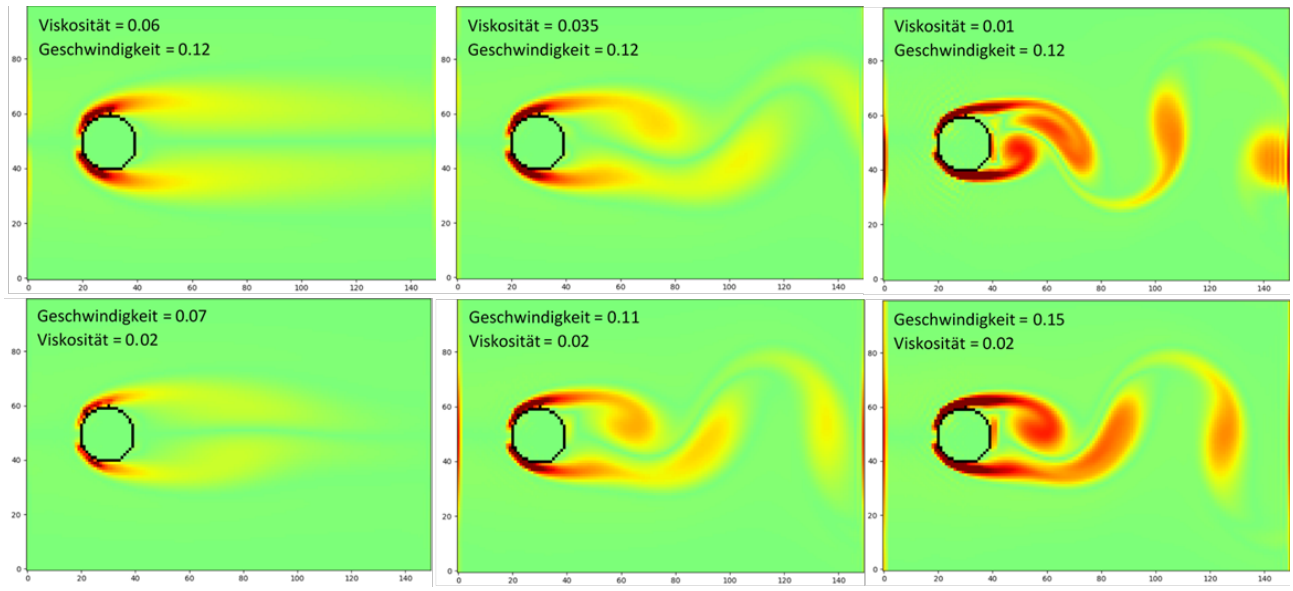


Figure 30: Dependence of the flow pattern on viscosity and flow velocity [61]

Classification of the mesoscopic approach

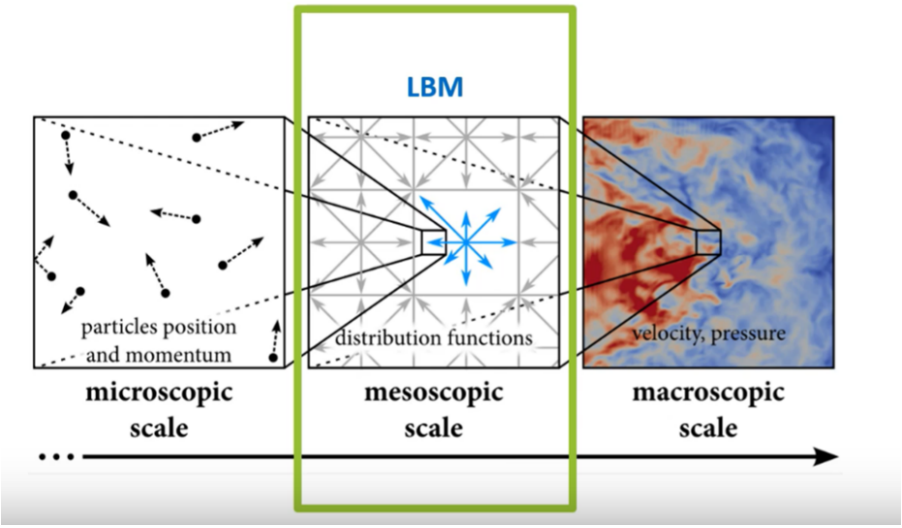


Figure 31: classification of the mesoscopic plane of observation between the microscopic and macroscopic plane of observation [59]

The macroscopic viewing plane is often compared to what is visible to the naked eye. The microscopic level of observation, on the other hand, is in the size range of atoms and molecules, i.e. in the range between 1 and 100 nanometers. The mesoscopic approach using the Lattice-Boltzmann method used represents the transition between the two levels.

The measurements in the wind tunnel

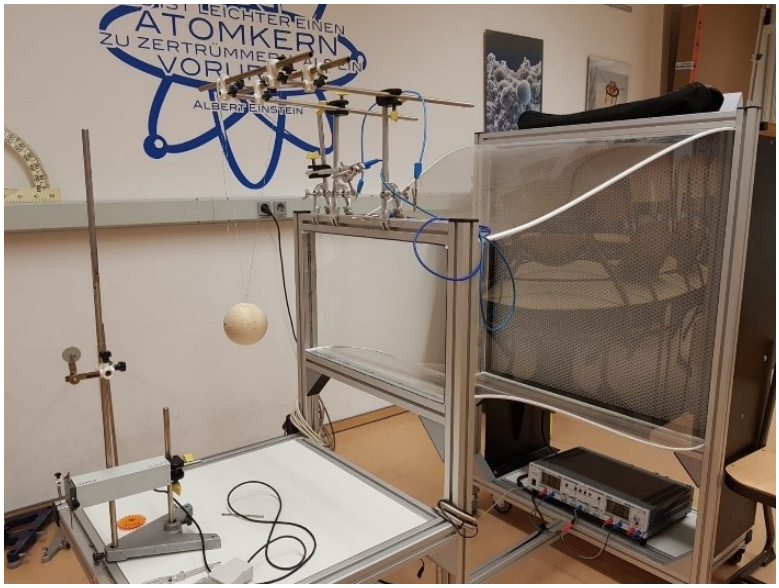


Figure 32: The wind tunnel in the student research center [61]

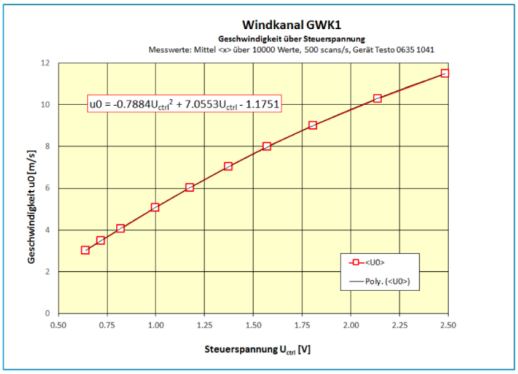


Figure 33: Data sheet for the wind tunnel in the student research center [63]

In order to control our Cw value calculations from the scanned bodies, we measured the Cw values of these bodies with the wind tunnel of the student research center. Based on its technical design, this is an open wind tunnel, also known as the Eiffel Canal. The air is sucked in by 25 fans, which can be controlled via the control panel. The control voltage of the fans and thus the speed of the air flow in the wind tunnel can be adjusted with the control panel. The maximum speed is around 11.5 m/s with a control voltage of 2.5 V. The dependency of the speed on the control voltage is shown in the diagram at the top right. The turbulence in the intake air is filtered out by a rectifier made of thin tubes with a diameter of 12 mm and a depth of 50 mm. Ultimately, the air flows through a contraction path, reducing the cross-section of the flow by 75% to $0.5 \cdot 0.3$ m. The beam outlet finally hits the body, which is a sphere in the picture above left.
In order to measure the force acting on the body, it is hung on several threads from a suspension device above the wind tunnel. Another thread is behind the body and is transmitted to a force gauge via a pulley. The dynamometer measures the force acting on the body in the direction of flow and transfers the data to a laptop via USB cable. There, in turn, the data can be evaluated graphically with the Cassy-Lab 2 program. In order to determine the air resistance

force acting on the body for the Cw value measurement, the body is hung on the dynamometer once without a counterflow and then again with a counterflow. The force measured at the force sensor is reduced by the countercurrent. The difference between the two measured forces gives the drag force. We have measured drag forces from bodies at multiple speeds ranging from 3.6 to 11.5 m/s. Ultimately, we used the measurements between 6.4 and 8.8 m/s for the Cw value calculation, since the force measurement was inaccurate at lower speeds and the body swayed around too much at higher speeds. In order to calculate the Cw value from the force measurements for the respective body, in addition to the measured drag force, the inflow area, the speed and the density of the air are missing. As shown above, the speed can be determined from the control voltage. We determined the inflow area approximately with a caliper gauge. We calculated the density of the air at a room temperature of 20 degrees Celsius using the thermal equation of state for ideal gases to be 1,204 kg/m³.
In order to be able to classify our drag coefficient measurements, we determined the Reynolds number used in the wind tunnel. These were determined using the formula of viscosity, speed, density and the characteristic length already mentioned above. The average length of the body perpendicular to the direction of flow is used as the characteristic length. The dynamic viscosity of air is $18 \cdot 10^{-6} Pa \cdot s$. All Reynolds numbers are in the range of Reynolds numbers with a constant Cw value, based on the course of the Cw value as a function of the Reynolds number of a sphere. With the approximation that the Cw value curve is the same for the other measured bodies, the measurements from the wind tunnel can be directly compared with our theoretical calculations. We have briefly summarized the results in the table below.
Based on the first four measured bodies in the table, we have an average deviation of 11.4% from the Cw value that we calculated with our program and the Cw value that was measured in the wind tunnel. The last two values in the table have larger deviations and are therefore not representative.

Table 4: Calculated drag values compared to measured values

Body	calculated Cw value	measured Cw value (wind tunnel)	Reynolds number with $v = 7.6 \frac{m}{s}$
Cup	0.47	0.41	44424
Nutcracker	0.69	0.67	29871
Pencil case	0.53	0.45	47743
Bullet	0.46	0.52	38297
Hair gel tube	0.92	0.70	24511
Lunch box	1.11	0.50	37276

The drag coefficient calculated by our software for these bodies is in the right range based on experience with similar bodies. However, the measurements in the wind tunnel deviate greatly from this, although we have repeated them several times. We have included the two values in the table because we want to show that errors can also quickly occur in the wind tunnel. On the one hand, we used a sphere to calibrate the wind tunnel. We measured a drag coefficient of 0.52. In reality, however, the drag coefficient of the ball is 0.47. This is a deviation of almost 10% and can be equated to the standard error of the wind tunnel. This standard error has also been proven for other bodies, such as a disk. However, other random errors also occur. Light bodies or strongly asymmetrical bodies begin to oscillate quickly in the suspension at higher speeds. Furthermore, the force transmission of the air resistance through a thread to the dynamometer is not exactly vertical and the bodies only hang approximately parallel to the flow direction of the wind tunnel. For example, if a pane hangs only slightly at an angle to the direction of flow in the wind tunnel, the drag coefficient drops significantly. When considering the deviation between the measured and calculated Cw values, it must also be mentioned that the shape of the bodies in the program only approximately corresponds to the shape in reality due to the scanning and dicing. This becomes clear in Figures 27 to 29. In summary, we can assume an average error of between 10 and 15% for measurements in the wind tunnel.
In addition to the Cw value measurement, we were also able to generate a flow pattern around the body in the wind tunnel. For this we used a wire with twisted turns, which hangs between the body and the wind tunnel. We dripped a glycerin solution onto these coils. The wire was then connected to a voltage source and heated until the glycerine evaporated and a flow pattern based on several strips was made possible. This process can be seen in Figures 27, 28 and 29.

Bibliography

Literature sources

[1] ARNOLD, KARIN; DIETRICH, VOLKMAR; EBERLE ANDREAS; GRIMMER, ANDREAS; JENCKEL, ASTRID; GRIMMER, ANJA; KARHOS, MARIANNE; LABAHN, BETTINA; LÜTTGENS, UWE; MALZ, RALF; PETERS, JÖRN; SCHÄFER, STEFFEN; TEICHERT, BORIS; TISCHENDORF, KERSTIN; *Chemie Oberstufe*; Cornelsen Verlag; 2015 1. Auflage 4. Druck

[2] KUCHLING, HORST; *Taschenbuch der Physik*; Fachbuchverlag Leipzig im im Carl Hanser Verlag; 2011 20.Auflage

[3] MESCHEDE, DIETER; *Gerthsen Physik*; Springer-Verlag; 2002 21.Auflage

Web sources

[3D Grafics]

[4] Autoren Kollektiv um: 129.132.239.8; STL-Schnittstelle
`https://de.wikipedia.org/wiki/STL-Schnittstelle`

[5] Autoren Kollektiv um: Arilou; Dateiformat
`https://de.wikipedia.org/wiki/Dateiformat`

[6] Autoren Kollektiv um: Cavasin, Christian; Wavefront OBJ
`https://de.wikipedia.org/wiki/Wavefront_OBJ`

[7] Autoren Kollektiv um: Coumans, Erwin; FBX
`https://en.wikipedia.org/wiki/FBX`

[8] Autoren Kollektiv um: Moskopp, Nils; Offenes Format
`https://de.wikipedia.org/wiki/Offenes_Format`

[9] Autoren Kollektiv um: Vierge, Marie; Rendern (Design)
`https://de.wikipedia.org/wiki/Rendern_(Design)`

[10] Wavefront Technologies; .OBJ Dateierweiterung
`https://www.reviversoft.com/de/file-extensions/obj`

[Lattice-Boltzmann]

[11] Autoren Kollektiv um; Abas, Aizat; Lattice Boltzmann Model of 3D Multiphase Flow in Artery Bifurcation Aneurysm Problem
`https://www.hindawi.com/journals/cmmm/2016/6143126/`

[12] Crowl, Lindsay; The Lattice Boltzmann Method Computational Fluid Dynamics
`https://www.math.utah.edu/~crowl/pres.pdf`

[13] Autoren Kollektiv um: Debenben; Lattice-Boltzmann-Methode
`https://de.wikipedia.org/wiki/Lattice-Boltzmann-Methode`

[14] Fitzpatrick, Richard; The Maxwell distribution
`http://farside.ph.utexas.edu/teaching/sm1/lectures/node72.html`

[15] Autoren Kollektiv um: Harris, Stewart; Boltzmann equation
`https://en.wikipedia.org/wiki/Boltzmann_equation`

[16] Autoren Kollektiv um: Mandl, F.; Maxwell–Boltzmann distribution
`https://en.wikipedia.org/wiki/MaxwellâĂŞBoltzmann_distribution`

[17] Mattila, Keijo; Implementation Techniques for the Lattice Boltzmann Method
`https://jyx.jyu.fi/bitstream/handle/123456789/24953/9789513939915.pdf?sequence=1`

[18] Two Phase Flow and Heat Transfer; Lattice Boltzmann Method
`https://www.youtube.com/watch?v=Cg-IRE19BEw`

[19] Weber State University; Lattice-Boltzmann Fluid Dynamics
`http://physics.weber.edu/schroeder/javacourse/LatticeBoltzmann.pdf`

[Mathematical basics]

[20] Autoren Kollektiv um: Alva2004; Divergenz mit Zylinder Koordinaten
https://de.wikipedia.org/wiki/Divergenz_eines_Vektorfeldes

[21] Arvo, James; Rotationsmatrix
https://en.wikipedia.org/wiki/Rotation_matrix

[22] Autorenkollektiv um: Franzl aus tirol; Drehmatrix
https://de.wikipedia.org/wiki/Drehmatrix

[23] Gräber, Peter-Wolfgang; Vektoralgebra
https://tu-dresden.de/bu/umwelt/hydro/iak/ressourcen/dateien/systemanalyse/studium/folder-2009-01-29-lehre/systemanalyse/folder-2010-04-12-1173264546/wws-02.pdf?lang=de

[24] Autorenkollektiv um: Hagman; Punkt-in-Polygon-Test
https://de.wikipedia.org/wiki/Punkt-in-Polygon-Test_nach_Jordan

[25] Universität Stuttgart; Differentialoperatoren
http://vhm.mathematik.uni-stuttgart.de/Vorlesungen/Vektoranalysis/Folien_Differentialoperatoren_in_Zylinderkoordinaten.pdf

[26] Autoren Kollektiv um: HerrHartmuth; explizite Euler-Verfahren
https://de.wikipedia.org/wiki/Explizites_Euler-Verfahren

[27] Weitz / HAW Hamburg; Gauß-Verfahren
https://www.youtube.com/watch?v=kfopPCDY1F0

[Physical basics]

[28] Autoren Kollektiv um: Alva2004; Navier-Stokes-Gleichungen
https://de.wikipedia.org/wiki/Navier-Stokes-Gleichungen

[29] Butkevich, Andrey; Schmiedel, Benjamin; Hydrodynamik - Euler- und Navier-Stokes-Gleichungen
https://www.thphys.uni-heidelberg.de/~mielke/Mechanik17-9d.pdf

[30] Autoren Kollektiv um: Bammel, Katja; Navier-Stokes-Gleichungen
https://www.spektrum.de/lexikon/physik/navier-stokes-gleichungen/10145

[31] Autoren Kollektiv um: Bauhofer, Prof. Dr. W.; Eulersche Gleichungen
https://www.spektrum.de/lexikon/physik/eulersche-gleichungen/4573

[32] Autoren Kollektiv um: Duesi; Reynolds-Zahl
https://de.wikipedia.org/wiki/Reynolds-Zahl

[33] Autoren Kollektiv um: Funkmich008; Torricelli Formel
https://de.wikipedia.org/wiki/Ausflussgeschwindigkeit

[34] Graber, Peter-Wolfgang; Torricelli Formel
https://tu-dresden.de/bu/umwelt/hydro/iak/ressourcen/dateien/systemanalyse/studium/folder-2009-01-29-lehre/systemanalyse/folder-2010-04-12-1173264546/wws-02.pdf?lang=de

[35] LearnMechE; Description and Derivation of the Navier-Stokes Equations
https://www.youtube.com/watch?v=NjoMoH51UZc&list=PLebuRGYfFXg28n878rvd3lDGTqJfYvABN&index=8&t=20s

[36] Numberphile; Navier-Stokes Equations
https://www.youtube.com/watch?v=ERBVFcutl3M&t=124s

[37] Autoren Kollektiv um: Svebert; ideale Flüssigkeit
https://de.wikipedia.org/wiki/Ideale_Fl$%$C3$%$BCssigkeit

[38] Uni Magdeburg; Körperumströmung - reibungsbehaftet
http://www.uni-magdeburg.de/isut/LSS/Lehre/Arbeitsheft/VII.pdf

[Python]

[39] Autorenkollektiv um: Holden, Steve; Python Einführung
https://wiki.python.org/moin/

[40] Hunter, John; Matplotlib Einführung
https://matplotlib.org/

[41] Klein, Bernd; Python Einführung
https://www.python-kurs.eu/

[42] Koonce, Grayson; Multithreading
https://graysonkoonce.com/waiting-until-a-thread-is-ready-in-python/

[43] Lundh, Fredrik; Python Einführung
http://effbot.org/

[44] NumPy developers; NumPy Einführung
http://www.numpy.org/

[45] Petri, Ulrich & Gutmann, Horst; Python Einführung
https://pyformat.info/

[46] SciPy developers; SciPy Einführung
https://docs.scipy.org/

[47] sentdex; Python GUI
https://www.youtube.com/user/sentdex/

[48] Seppke, Benjamin; Bildverarbeitung Python
https://kogs-www.informatik.uni-hamburg.de/~neumann/BV-WS-2010/Uebungen/
bv-python-einfuehrung.pdf

[49] Shenk, Justin; 3D Würfel Darstellung
https://gist.github.com/JustinShenk/c407b02a4d5c19f89dfc87e9678dcc22

[50] Sherwood, Bruce; 3D-Animation Python
https://vpython.org/

[51] Stack Overflow developers; Stack Overflow
https://stackoverflow.com/

[Texmaker]

[52] Hammersley, John; Latex Templates
https://de.overleaf.com/latex/templates

[53] Safra; Latex Einführung
https://latex-kurs.blogspot.com/

[54] Stack Overflow developers; Latex Einführung
https://tex.stackexchange.com/

[55] Wipper, Joachim; Matrizen
https://mo.mathematik.uni-stuttgart.de/kurse/kurs44/seite26.html

Image sources

[56] https://upload.wikimedia.org/wikipedia/commons/c/cb/Lattice_boltzmann_3steps.svg

[57] http://physics.weber.edu/schroeder/javacourse/LatticeBoltzmann.pdf

[58] https://jyx.jyu.fi/bitstream/handle/123456789/24953/9789513939915.pdf?sequence=1 S.68

[59] https://youtu.be/qhr3p0aShjg

[60] https://imgr1.auto-motor-und-sport.de/Lamborghini-Aventador-LP-700-4-\
Seite-Windkanal-articleGalleryOverlay-9c21d659-615877.jpg

[61] Aus eigener Quelle

[62] https://upload.wikimedia.org/wikipedia/de/thumb/8/81/Kugel-Reynolds.png/
500px-Kugel-Reynolds.png

[63] Send, Johanna / Dr.Wolfgang; Anleitung zum Windkanal; Großer Windkanal ANIPROP GWK 1; Seite 11

Affidavit

We hereby declare that we have completed this work independently and without outside help. We have marked as such text passages that are literally or conceptually based on the work of third parties.

Authors signatures:

Adrian Kühn :

Frank Long :

Paul Marschall :