Studienarbeit: Efficient Wind Turbine Simulator

Hochschule für Technik Rapperswil, Abteilung Informatik
Autoren: Mario Tarreghetta & Sebastian Hug
Betreuung: Henrik Nordborg, Alain Schubiger

Herbstsemester 2018
Version 1.0

Inhaltsverzeichnis

Teil I.

Einleitung

1. Abstract

Mit der Lattice-Boltzmann-Methode können basierend auf der Boltzmann-Gleichung Strömungen simuliert werden. Da dieser Algorithmus gut parallelisiert werden kann, können schneller akzeptable Resultate erzielt werden als mit konventionellen CFD (Computational Fluid Dynamics) Berechnungstools. Das Resultat der Arbeit umfasst eine Software, die die Lattice-Boltzmann-Methode implementiert und den Algorithmus parallel auf der Grafikkarte ausführt. Es sollen weitere Modelle integriert werden, sodass schlussendlich der Luftstrom durch eine vertikalachsige Windturbine simuliert werden und das resultierende Drehmoment berechnet werden kann, das auf die Turbine wirkt. Somit kann für eine spezifische Turbinenkonfiguration der Wirkungsgrad errechnet werden. Die am Schluss erstellte Software läuft auf Windows und Linux und wird textbasiert ausgeführt. Auf einem herkömmlichen Computer (Stand 2018) konnte die Laufzeit um den Faktor 15 reduziert werden, wenn die GPU anstatt die CPU benutzt wurde.

2. Aufgabenstellung

Semesterarbeit HS 2018
GPU-optimiertes Simulationstool für die Auslegung von vertikalachsigen Windturbinen
Studierende: Sebastian Hug, Mario Tarreghetta
Betreuer: Henrik Nordborg, Alain Schubiger

2.1. Ausgangslage

Die Auslegung vertikalachsiger Windturbinen ist im Moment sehr anspruchsvoll. Vereinfachte Berechnungstools sind sehr ungenau und Computational Fluid Dynamics (CFD) sehr zeitintensiv. Mit der Arbeit soll ein neues Berechnungstool geschaffen werden, das eine schnelle Optimierung der Windturbine ermöglicht.

Kern des Berechnungstools ist die Lattice Boltzmann Methode zur schnellen Berechnung der Strömung. Da diese sich auf GPUs hervorragend parallelisieren lässt, soll die Implementierung gleich mit CUDA gemacht werden.

2.2. Aufgabenstellung

Folgende Arbeiten sind zu bewerkstelligen:

1. Einarbeitung in den Lattice Boltzmann Algorithmus (LBM)
2. Implementierung und Parallelisierung eines 2D-Strömungslösers mit LBM
3. Valdidierung mit einem 2D Objekt in CFD
4. Implementierung der vertikalachsigen Windturbine (als Kraft auf die Strömung)
5. Validierung der Simulation
6. Benchmarking
7. Einfaches User Interface (textbasiert oder in Matlab)
8. Dokumentation.

2.3. Bewertung

Das Ergebnis der Arbeit und der Bericht werden bewertet.

3. Management Summary

3.1. Ausgangslage

Momentan ist die Auslegung vertikalachsiger Windturbinen sehr anspruchsvoll. Es braucht Berechnungstools, welche den Wirkungsgrad dieser Windturbinen simulieren können. Diese sind jedoch ungenau und Computational Fluid Dynamics (CFD) sind zeitintensiv. Die Lattice Boltzmann Methode (LBM) kann dieses Problem lösen, sie wurde aber für diesen spezifischen Anwendungsfall (mit einer vertikalachsigen Windturbine) noch nicht implementiert und wenn, dann nur als serielle Berechnung, welche viel Zeit in Anspruch nimmt.

LBM lässt sich gut parallelisieren und deshalb ist das Ziel der Arbeit, den Algorithmus in eine Applikation zu verpacken, die parallelisiert auf der GPU ausgeführt werden kann. Damit kann das Drehmoment berechnet werden, das die Turbine antreibt. So kann für eine spezifische Konfiguration der Turbine und der Flügel der Wirkungsgrad geschätzt und optimiert werden.

3.2. Vorgehen

Als erster Schritt wurde der LBM-Algorithmus seriell implementiert, um sicherzustellen, dass das Verständnis für die physikalischen Grundlagen und die Lattice Boltzmann Methode vorhanden ist. Auf dieser Basis wurde dann parallelisiert. Mit der parallelisierten Applikation können nun Validierungen gegen andere Simulationssoftware und Performancemessungen stattfinden.

3.3. Technologien

Um eine Software zu parallelisieren bietet sich CUDA von NVIDIA an. CUDA ist eine Plattform mit welcher parallele Berechnungen programmiert und ausgeführt werden können (aktuell CUDA 10.0). Dafür muss der Benutzer oder die Benutzerin eine CUDA-fähige NVIDIA Grafikkarte besitzen. Als Programmiersprache kam C++11 mit vielen C Konstrukten zum Einsatz. Matlab wird verwendet, um den Output des Tools zu visualisieren. Da CUDA plattformunabhängig ist, ist die Applikation auf Linux und Windows lauffähig.

3.4. Ergebnis

Das Tool kann mit verschiedenen Parametern bezüglich der Turbine und der Windgeschwindigkeit konfiguriert werden. Es berechnet die Geschwindigkeit und Dichte des Fluids pro Ort und Zeit. Aus verschiedenen Kennzahlen lässt sich so das Drehmoment berechnen, das auf die Turbine wirkt, sowie den Wirkungsgrad der Turbine.

Die Applikation ist in C++ basierend auf dem CUDA Framework von NVIDIA geschrieben, wodurch sich die Berechnungen parallel auf der Grafikkarte ausführen lassen. Der Zeitgewinn gegenüber der seriellen Ausführung beträgt auf einem typischen Laptop einen Faktor von 15. Je besser aber die Grafikkarte, desto schneller der Algorithmus.

3.5. Ausblick

Die Software kann auf verschiedene Varianten weiterentwickelt werden. Eine grafische Bedienoberfläche würde die Bedienung ziemlich vereinfachen. Man könnte auch den visuellen Output in Echtzeit darstellen. Zudem wäre vorstellbar, auf diese Weise die Applikation fernzusteuern, sodass man die Parameter auf einem einfachen Gerät eingeben kann und die Berechnung dazu dann auf einem Grafikkartencluster ausgeführt wird. Der Output wäre dann trotzdem auf dem Inputgerät sichtbar.

Unsere Software bietet nur einen zweidimensionalen Blick auf die Turbine und das Fluid. LBM erlaubt aber auch die Berechnung im dreidimensionalen Raum. Auch dies ist eine Erweiterung der Applikation, die sinnvoll wäre.

Die Turbine wird momentan als Verteilfunktion an einem bestimmten Ort angesehen. Schöner wäre, wenn die Turbine als 3D Model eingelesen werden kann und dann mit der Umdrehung simuliert wird.

Diese beschriebenen Erweiterungen sind mit einem grösseren Aufwand verbunden und hatten im Rahmen dieser Studienarbeit keinen Platz. Sie können jedoch Bestandteil einer weiteren Studien- oder Bachelorarbeit sein.

4. Dateien und Quellen

4.1. Dateiablage

Source code https://gitlab.com/alcedo-atthis/source

Dokumentation, Taskmanagement und Sitzungsprotokolle https://gitlab.com/alcedo-atthis/doc

4.2. Weitere Quellen

Während der Arbeit wurden diverse Papers konsultiert. Diese werden während der Dokumentation nicht speziell erwähnt. Beachten Sie deshalb

- Actuator-Line Model in a Lattice Boltzmann Framework for Wind Turbine Simulations[4],
- Implementation of a Lattice-Boltzmann method for numerical fluid mechanics using the nVIDIA CUDA technology[2],
- The Lattice Boltzmann Method for Fluid Dynamics: Theory and Applications[3],
- Thermal Simulations at low to modest Reynolds numbers using the lattice Boltzmann method[5] und
- Lattice-Boltzmann Codeentwicklung[1].

4.3. Matlab Skripte

Im Anhang sind drei Matlab Skripte gelistet (Kapitel 5). Ein grosser Teil unserer Implementierung basiert auf diesen Skripten.

Teil II.

Technischer Bericht

1. Einleitung

Diese Studienarbeit ist mehr eine Forschungsarbeit als eine klar definierte Projektarbeit. Deshalb entwickelten sich gewisse Anforderungen erst während dem Semester, da deren Möglichkeiten zuerst evaluiert werden mussten. Wir halten in diesem Teil des Dokuments die aufgekommenen Herausforderungen fest, Entscheidungen, die während der Implementierung gefällt worden sind und versuchen unsere Lösung zu validieren und zu benchmarken.

1.1. Problemstellung

Die Lattice-Boltzmann-Methode soll verwendet werden, um vertikalachsige Windturbinen in einem Fluid zu simulieren und deren Wirkungsgrad auszurechnen. Der Algorithmus soll parallelisiert ausgeführt werden, damit die Berechnung schneller erfolgen kann. Obwohl der Algorithmus (siehe Abschnitt 1.2) schnell beschrieben ist, gibt es viele Details zu lösen, bevor er parallelisiert implementiert werden kann:

- Wie soll die Turbine implementiert werden? Sie hat bei jedem Zeitschritt eine andere Position und wird vom Wind zusätzlich angetrieben oder gebremst.
- SI-Units müssen in Lattice-Units konvertiert werden.
- Wie wird die Windströmung im Kanal verteilt?
- Wie sehen die Randbedingungen, also die Kollisionen an den Wänden, aus?
- Welche Parameter werden für CUDA verwendet (Anzahl Blocks und Grids)?
- Wie bringen wir einen dreidimensionalen Array in CUDA, wo nur eindimensionale unterstützt werden?
- ...

1.2. Die Lattice-Boltzmann-Methode

Mit der Boltzmann-Gleichung berechnet man die statistische Verteilung von Teilchen in einem Medium (Fluid). Die Variablen darin sind unter anderem

- Ort $\vec{x}$
- Zeit t
- Geschwindigkeit $\vec{v}$ (in mehrere Richtungen)

Daraus errechnet sich die Verteilungsdichte $f(\vec{x}, \vec{v}, t)$. Die Lattice-Boltzmann-Methode (LBM) basiert auf dieser Boltzmann-Gleichung. $\vec{x}$ muss in diesem Fall diskretisiert werden. Dafür wird ein Gitter über das Medium gelegt, das sogenannte Lattice. Mehrere Gitterformen können dafür verwendet werden, in unserem Fall, ein zweidimensionales Gitter mit neun Richtungen der Teilchengeschwindigkeit, nennt man die Form D2Q9-Modell.

Ein Zeitschritt besteht aus zwei Teilschritten, dem Kollisions- und den Strömungsschritt (Abbildung 1.2).

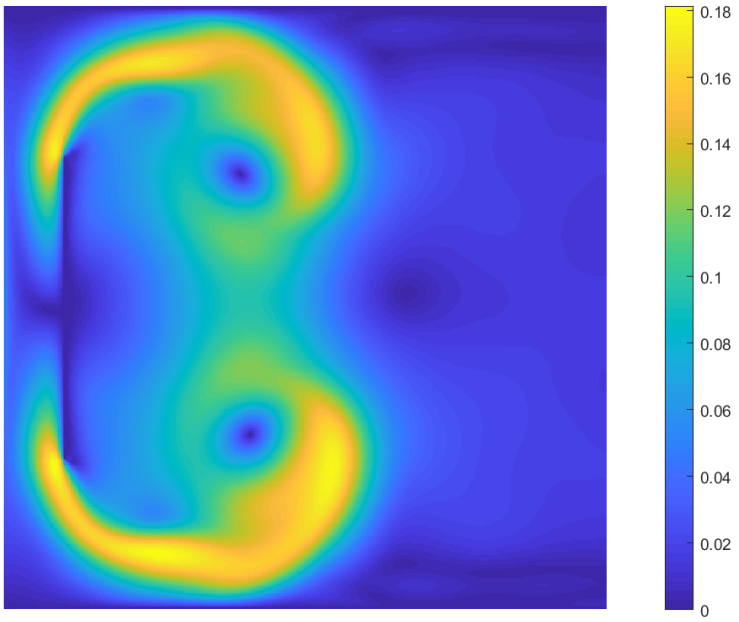


Abbildung 1.1.: Beispiel einer simulierten Strömung. Als Hindernis wurde eine Wand gewählt. (Screenshot von alcedo)

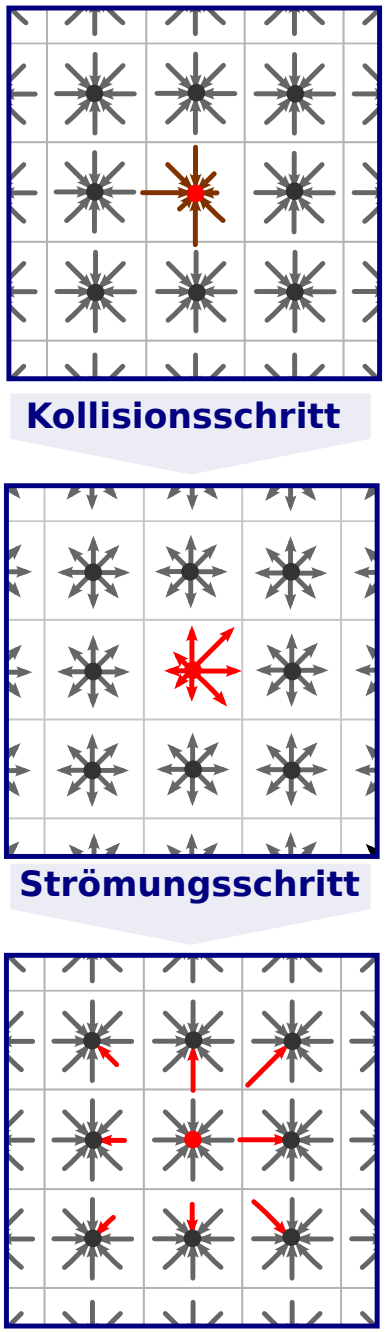


Abbildung 1.2.: Der LBM Kollisions- und Strömungsschritt[6]

1.2.1. Kollisionsschritt

Generell gilt

$$f_i^*(\vec{x}, t) = f_i(\vec{x}, t) + \Delta_i$$

Mit dem * ist die kollidierte Verteilung markiert. Das Δ steht für den Kollisionsterm, welcher zum vorherigen Zeitschritt addiert wird. Für diesen Schritt gilt das Gesetz der Impulserhaltung. Der Kollisionsterm Δ_i kann unterschiedlich berechnet werden, hier wird die Bhatnagar-Gross-Krook (BGK) Näherung verwendet:

$$\Delta_i = \frac{1}{\tau}(f_i - f_i^{eq})$$

τ entspricht der Relaxationszeit und hängt von der Viskosität ab. f^{eq} ist die Verteilung im Equilibrium.

1.2.2. Strömungsschritt

Beim anschliessenden Strömungsschritt werden die Partikel gemäss ihrer Verteilung in die nächste Zelle verschoben:

$$f_i(\vec{x}_i + \vec{c}_i, t+1) = f_i^*(\vec{x}, t)$$

$\vec{c}_i$ repräsentiert einen Richtungsvektor der Richtungen $i = 0..8$. Hiermit ist der erste Zeitschritt fertig und das Ganze beginnt von vorne.

1.3. CUDA

CUDA ist eine Plattform für GPGPU (General-Purpose computing on Graphics Processing Units). Man kann damit direkt auf dem virtuellen Instruktionsset von NVIDIA Grafikkarten programmieren. Die aktuelle Version ist CUDA 10.0. Die zu verwendende Programmiersprache ist C++ oder C oder eine Mischung von beiden (was sich aus C++ ergibt). Das CUDA SDK bringt einen Compiler (`nvcc`) mit, welcher auf `gcc` basiert. Es ist plattformunabhängig, man kann auf Linux oder Windows für CUDA entwickeln.

Parallelisierte Methoden werden als sogenannte *Kernels* geschrieben. Darin verwendete Daten müssen auf die Grafikkarte kopiert (Zeilen 4 und 6 in Listing 1.1) und zur späteren Verwendung auf der CPU wieder zurückkopiert werden. Der Aufruf eines Kernels erfolgt mit einer speziellen Syntax, die C++ eigentlich nicht kennt (Listing 1.1, Zeile 8).

Listing 1.1: Beispielcode in CUDA

```
1 const int f_size = 256 * 256 * 9;
2 float *f_in = new float[f_size];
3 float *g_f_in;
4 cudaMalloc(&g_f_in, f_size * sizeof(float));
5
6 cudaMemcpy(g_f_in, f_in, f_size * sizeof(float), cudaMemcpyHostToDevice);
7
8 collide <<<gridDimension, blockDimension>>> (g_f_in, ...);
```

`gridDimension` entspricht der Anzahl Blocks, die gleichzeitig in einem *Streaming Multiprocessor* (SM) ausgeführt werden. Die `blockDimension` definiert die Grösse eines Blocks, also wie viele Threads darin gleichzeitig laufen.

Ein Kernel ist nichts anderes als eine Methode, die Signatur enthält jedoch den Präfix `__global__` um den Kernel zu markieren (Listing 1.2).

Listing 1.2: CUDA Kernel Signatur

```
1 __global__ void collide(float *f_in, ...);
```

1.4. Anforderungen

Folgende Anforderungen ergeben sich aus der Aufgabenstellung von Henrik Nordborg (siehe Abschnitt 2.2):

- Die Software arbeitet parallel.

- Es wird das D2Q9 Modell verwendet.
- Die Simulation wird mit CFD validiert.
- Es liegt ein Benchmark zur Simulation vor.
- Eine Parameteränderung des Inputs verursacht keine Neukompilation der Software.
- Textbasiertes User Interface ist vorhanden.
- Die Arbeit wird dokumentiert.

1.5. Projektplan

Der Projektplan (Tabelle 1.1) ist relativ grob, da wir ein Forschungsprojekt und kein traditionelles Softwareprojekt bestreiten. Die einzelnen Meilensteine konnten wir jedoch festlegen und kommentieren dann dessen Erfüllung. Da die Sitzungen mit den Betreuern jeweils am Mittwoch stattfinden, sind die Meilensteine auf einen Sonntag gesetzt. So bleibt uns genügend Zeit, auf allfällige Kritik zu reagieren.

Datum	Meilenstein	Kommentar
30.09.2018	**M1** Alle verstehen LBM und es liegt eine einfache Softwareimplementierung vor	Es wird sicherlich noch weitere Fragen geben, jedoch ist das Grundkonstrukt LBM beiden klar. Der Code muss noch validiert werden.
7.10.2018	**M2** Grundalgorithmus in CUDA umgesetzt	Code funktioniert in CUDA schon ziemlich gut, jedoch nur bei korrekt eingestellten Parametern (Reynoldsnumber, Auflösung, Timesteps).
7.10.2018	**M3** Client-Server Architektur steht	Nie mit Betreuern besprochen, da keine Anforderung. Verworfen, da Priorität anders gesetzt und nicht notwendig für die Funktionalität des Endprodukts.
21.10.2018	**M4** Kraftterm implementiert	Nicht erreicht, aber notwendig. Es fehlen noch Informationen seitens Betreuung, gleichzeitig Probleme mit grosser Reynoldszahl.
21.10.2018	**M5** Output identisch mit Referenzimplementierung	Da Kraftterm noch nicht implementiert sind die Outputs nur ≪fast≫ identisch.
28.10.2018	**M6** Turbulenzmodell implementiert	Zuerst nicht pünktlich und dann nicht erreicht. Stattdessen die implizite Methode einbauen (siehe Protokoll vom 6.11.2018). Es fehlte an Zeit bei allen Parteien.
18.11.2018	**M7** Turbine eingebaut	Erreicht für gemittelte Turbine, wir lassen dabei die Flügel nicht wandern. Weitere offene Fragen müssen noch geklärt werden.
25.11.2018	**M8** Programm mit SI Units verwendbar	Wurde mit zwei Wochen Verspätung erreicht.
9.12.2018	**M9** Implementierung validiert und benchmarked	Benchmarking war erst am 11.12.2018 fertig. Validierung verlief zuerst negativ, deshalb dauerte dieser Vorgang länger, bis zum Wochenende von 15.12.2018.
21.12.2018	**M10** Dokumentation und Plakat fertig und abgegeben	Dieser Meilenstein wurde pünktlich erreicht.

Tabelle 1.1.: Projektplan

2. Architektur

2.1. Code

Die Auflistung 2.1 zeigt die Files des Projektes. Es ist eine Visual Studio Solution, die jedoch auch mit einem Makefile für Linux kompiliert werden kann. Interessant sind vor allem die Dateien im `Alcedo.Core` Verzeichnis.

Listing 2.1: Code Files im Directory `src`

```
./
Alcedo.sln
Makefile
Alcedo.Core/
    Alcedo.Core.vcxproj
    config.h
    config.cpp
    constants.h
    lbm.h
    lbm.cu
    main.h
    main.cu
    printer.h
    printer.cpp
    program_params.h
    program_params.cpp
    results.h
    results.cpp
    wing_profiles.h
    wing_profiles.cpp
```

Algorithmus-, aber auch Programmkonstanten sind im `constants.h` File festgehalten. Dort sind Errorcodes, aber auch algorithmusspezifische Werte, wie zum Beispiel die gerichteten Geschwindigkeiten, festgelegt.

Die `main` Methode gabelt das Programm schon während dem Kompilieren in die Linux- oder Windows Richtung, da das Parsing der Inputparameter auf beiden Systemen anders geschieht. Deshalb gibt es pro Betriebssystem eine eigene `main` Methode. Am Schluss dieser Methoden wird jeweils aber wieder dieselbe `lbm(...)` Methode aufgerufen. Sie bereitet den Algorithmus und die Grafikkarte vor und führt im Anschluss die zentralen Berechnungen aus.

Der LBM-Algorithmus ist in `lbm.cu` implementiert. Es ist der einzige Codeteil, der nicht objektorientiert geschrieben ist. Die einzelnen Methoden werden im Abschnitt 2.1.2 genauer erläutert.

2.1.1. Helpers

Die fünf Helferdateipaare `program_params`, `printer`, `config`, `results` und `wing_profiles` dienen dazu, den Code thematisch zu trennen. Mit dem `printer` kann Text auf der Konsole ausgegeben werden, entweder als Info, Warnung oder Error. Die `program_params` Klasse bietet vor allem Getter und Setter für die von aussen reingegebenen Parameter. Diese werden beim Setzen validiert, weshalb die Setter Methoden einen Statuscode als Integer zurückgeben. Die `wing_profiles`

speichern vordefinierte Flügelprofile ab, welche dann vom Benutzer oder der Benutzerin verwendet werden können. `config` hilft, eine vorgegebene Config zu parsen oder die Config des Outputs zu erstellen. Dateien werden über die `results` ausgegeben.

2.1.2. `lbm`

`lbm.h` definiert drei Funktionen gegen aussen. Die `init` Funktion initialisiert den Algorithmus und wird zuerst aufgerufen. Danach sind die beiden Kernels definiert, `collide` und `stream`.

Identifikation der Zelle Um im Kernel die Werte einer Zelle zu berechnen, muss bekannt sein, in welcher Zelle man sich befindet. Das wird über die CUDA Blockkoordinaten definiert. Im Listing 2.2 ist der Vorgang aufgezeichnet.

Listing 2.2: Zelle identifizieren

```
1 __global__ void collide(float *f_in, ...) {
2     const int x = blockIdx.x * blockDim.x + threadIdx.x;
3     const int y = blockIdx.y * blockDim.y + threadIdx.y;
4     ...
5 }
```

`init` Hier werden die nötigen Datenstrukturen initialisiert. Hauptsächlich die f-Verteilungen, die standardmässig ein uniformes Profil aufweisen. Auskommentiert befindet sich dort Code, um ein Poiseuille Profil des Fluids zu erreichen.

`collide` `collide` implementiert den Kollisionsschritt von LBM. Alle Verteilungen werden über die Relaxationszeit an das Equilibrium angenähert. Zusätzlich ist das Inlet und Outlet ebenfalls hier implementiert. Beim Inlet ist die Geschwindigkeit über die Einlassgeschwindigkeit vorgegeben und die Verteilungen ergeben sich daraus. Die Kraft des Flügels auf das Fluid wirkt ebenfalls im Kollisionsschritt.

`localDensity` Die Dichte pro Zelle ist in LBM theoretisch immer 1. Praktisch schwankt sie aber ganz leicht um 1. `localDensity` berechnet die Zellendichte, in dem die Verteilungen aufsummiert werden.

`localVelocity` Die Geschwindigkeit einer Zelle ergibt sich durch das Aufsummieren der Verteilungen in X und Y Richtung geteilt durch die Dichte der Zelle.

`isCellOnCircle` Hier wird für eine Position X/Y bestimmt, ob sie auf dem Rand der Turbine liegt. Da einfachheitshalber die Kraft der Flügel über den gesamten Umfang der Turbine verteilt wird, statt einzelne rotierende Flügel zu implementieren, muss in jeder Zelle, die auf dem Turbinenrand liegt, ein kleiner Teil der Flügelkraft wirken. Dazu wird die Distanz zwischen Zelle und Turbinen-Mittelpunkt berechnet und überprüft, ob diese, auf ganze Zahlen gerundet, gleich dem Radius der Turbine ist.

`reflect` Diese Methode kapselt das Verhalten von Kollisionen der Teilchen mit Hindernissen. In Zellen die als Hindernis deklariert sind invertiert `reflect` alle ankommenden Verteilungen und sendet sie in die Gegenrichtung. Das geschieht als Teil des `collsion`-Schrittes.

`calculateWingForces` `calculateWingForces` verwendet den Angriffswinkel des Flügels um für ein spezifiziertes Flügelprofil die Lift- und Drag-Koeffizienten zu bestimmen. Somit lässt sich dann die totale Kraft berechnen, die auf einen Flügel an Stelle X/Y wirkt.

`createInletDistributions / createOutletDistributions` In diesen beiden Methoden werden die Randbedingungen für den Einlass und Auslass sichergestellt. Für den Einlass bedeutet das eine konstante Geschwindigkeit für alle Zellen, nämlich die Windgeschwindigkeit. Der Auslass hingegen wird über die vorhergehenden Zellen hergeleitet.

`calculatePhi` Für die Berechnung des Angriffswinkels auf einen Flügel wird die Position der Zelle auf der Umlaufbahn benötigt. `calculatePhi` berechnet diesen Wert im Bogenmass durch Trigonometrie.

`stream` `stream` implementiert den Strömungsschritt von LBM. Pro Zelle wird jede der Verteilungen gemäss ihrer Richtung in die Nachbarzelle kopiert. Damit sich Werte nicht überschreiben werden zwei Datenstrukturen für die Verteilungen benötigt: `f_in` und `f_out`.

2.2. Entscheidungen

2.2.1. Performance

Viele Entscheide haben wir aufgrund von Performanceüberlegungen und Limitierungen bei der CUDA Programmierung gefällt.

Eindimensionale Arrays $f(\vec{x}, \vec{v}, t)$ entspricht im Code den `f_in`- und `f_out`-Arrays. Diese besitzen drei Dimensionen, zwei für die X- und Y-Position der Zellen und eine für die neun Richtungen. Mehrdimensionale Arrays in den GPU Speicher zu kopieren dauert allerdings länger als eindimensionale[1]. Einen flachen, eindimensionalen Array zu verwenden ist für den Programmierer oder die Programmiererin wiederum mühsam, weshalb wir da C Makros zur Hilfe nahmen. Dasselbe gilt für die zweidimensionalen Arrays. Die Makros sind in Listing 2.3 definiert.

Listing 2.3: Makros für vereinfachten Arrayzugriff

```
1 // lbm.h
2 #define INDEX_3D(x, y, z) (x + xSize * (y + ySize * z))
3 #define INDEX_2D() (x + xSize * y)
4
5 // lbm.cu
6 f_in[INDEX_3D(x, y, LEFT)] = f_in[INDEX_3D(x, y, RIGHT)] - ...;
```

Gleitkommazahlen Um Speicherplatz zu sparen werden Floats anstatt Doubles verwendet, jedoch reicht die Floatgenauigkeit für die Simulation in dieser Arbeit aus. Der Speicher in den Grafikkarten ist begrenzt und je nach eingestellter Auflösung sind die Arrays ziemlich speicherintensiv. Auch die resultierenden Outputfiles können schnell über 100 MB gross werden, ein gesamter Rechenoutput somit mehrere Gigabyte. Eine Halbierung dieses Speichers ist somit ein spürbarer Vorteil. Somit wird der Datenaustausch zwischen CPU und GPU schneller, welcher einer der teuersten Vorgänge bei der CUDA-Programmierung ist.

[1] https://devtalk.nvidia.com/default/topic/500681/1d-array-vs-2d-array/, 4.12.2018

Block und Gridsize Die Berechnung unserer Block- und Gridsize ist im Code 2.4 festgehalten.

Listing 2.4: Block- und Gridsize

```
const int blockWidth = 8;

const int gridX = (xSize + (blockWidth - 1)) / blockWidth;
const int gridY = (ySize + (blockWidth - 1)) / blockWidth;
const dim3 blockDimension(blockWidth, blockWidth, 1);
const dim3 gridDimension(gridX, gridY, 1);

...

stream <<<gridDimension, blockDimension>>> (g_f_in, ...);
```

Die Anzahl der Blocks ist abhängig von der resultierenden Auflösung der Berechnung. Das ist klar, denn wenn die Auflösung 1000 Zellen beträgt, müssen 1000 Zellen gleichzeitig berechnet werden, wenn sie 100'000 Zellen beträgt, müssen 100'000 Berechnungen parallel ausgeführt werden. Die Blockbreite von 8 basiert auf Experimenten (Abbildung 2.1) mit verschiedenen Blockgrössen. Mit dieser Grösse läuft der Algorithmus auf der NVIDIA Tesla V100 am schnellsten. Je nach Grafikkarte kann es jedoch von Vorteil sein, diesen Wert auf 16 oder einen anderen zu erhöhen.

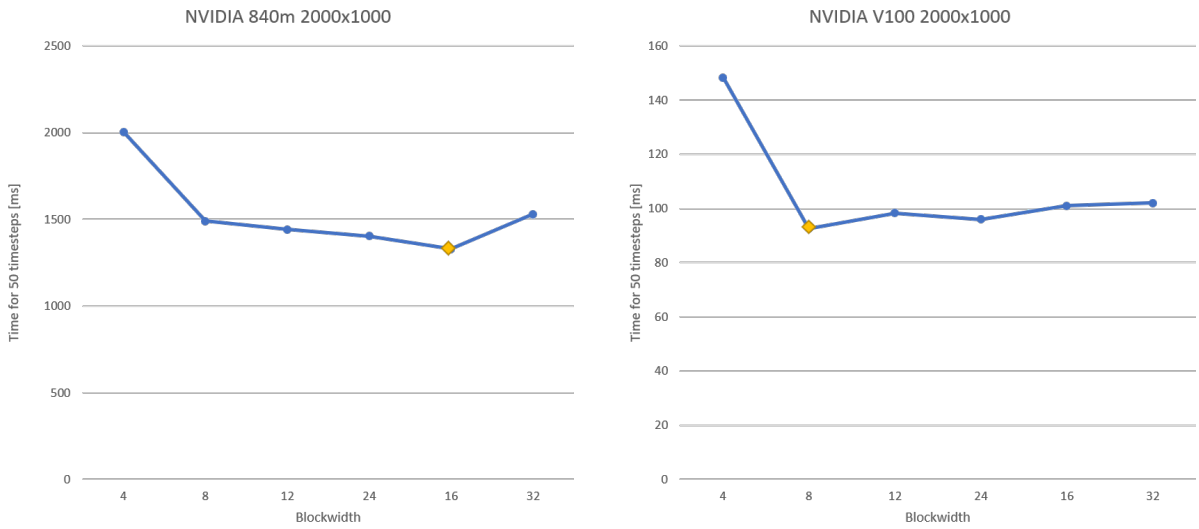


Abbildung 2.1.: Verschiedene Blockgrössen führen auf unterschiedlichen GPUs zu anderen Geschwindigkeiten

2.2.2. Algorithmus

Parallelisierung der Zellen In jedem Zeitschritt wird die Zellberechnung parallelisiert. Die Verarbeitung der neuen Richtungen pro Zellen wird dann seriell durchgeführt.

Gemittelte Turbine Eine Variante der Turbinenimplementierung wäre das Einlesen eines Modells der Turbine und Berechnen der Flügelposition anhand Wind und Drehgeschwindigkeit. Eine einfachere Version ist das Mitteln dieser wirkenden Kräfte auf einen Kreis, also die Verteilung der Gesamtkraft über die ganze Windturbine anstatt die einzelnen Flügel. Die Turbine dreht schnell, sodass an jedem Ort im Durchschnitt ständig dieselbe Kraft gilt. Grundlage für die Flügelkraft bieten Lift/Drag-Profile (Beispiel eines Flügels von Airfoil Tools unter http:

//www.airfoiltools.com/airfoil/details?airfoil=naca0012h-sa). Benutzer_innen können eigene Flügelprofile beim Ausführen des Programms angeben und es ist ein Testprofil vorgegeben.

Umfang Um die Kraft auf die einzelnen Zellen zu reduzieren muss sie durch die Anzahl Zellen der Turbine geteilt werden. Die genaue Anzahl Zellen ist jedoch nicht schnell zu berechnen, zu zeitintensiv. Was aber schnell berechnet ist, ist der Umfang der Windturbine. Die Gesamtkraft wird zur Näherung durch den Umfang anstatt durch die genaue Anzahl Zellen geteilt.

Parameterarray Damit nicht viele Inputparameter von Methode zu Methode übergeben werden müssen wird jeweils ein Array mit den Parametern überreicht. Im Array sind die Parameter an vordefinierten Indizes abgelegt. So werden die Methodensignaturen kürzerer.

Listing 2.5: Parameterarray

```
// constants.h
const int PARAM_X_SIZE = 0;

// main.cu
params[PARAM_X_SIZE] = this->params.get_xSize();

// params auf die GPU kopieren (Code weggelassen)

// lbm.cu
const int xSize = (int)params[PARAM_X_SIZE];
```

Listing 2.5 zeigt, wie wir im `main.cu` einen exemplarischen Parameter festlegen und ihn dann im `lbm.cu` wieder holen.

2.2.3. Einheiten

Eine Eigenheit von LBM ist, dass alle Grössen einheitslos sind. Die Breite einer Zelle ist 1, die Dauer eines Zeitschrittes ist 1 und die Dichte ist auch 1. Das erfordert eine Umrechnung von SI-Einheiten, um das Verfahren effektiv einsetzen zu können. Es gilt drei Grundgrössen zu bestimmen, anhand derer die restlichen Einheiten dann umgerechnet werden können: Distanz, Zeit und Masse.

Distanz Distanz-Parameter werden als Meter angegeben. Um diese umzurechnen braucht man eine Distanz als Meter und die dazugehörige Anzahl Zellen. Per Definition ist der Radius der Turbine im Grid immer $\frac{1}{16}$ der Höhe des Grids. Als Input-Parameter wird zusätzlich der Radius der Turbine in Meter und die Grösse des Grids in Anzahl Zellen benötigt. Somit lässt sich der Umrechnungsfaktor von Meter zu Zellen bestimmen.

$$r = Radius\ der\ Turbine\ [m]$$

$$ly = Anzahl\ Zellen\ in\ Y - Richtung$$

$$dx = \frac{r}{\frac{ly}{16}}$$

Zeit Für die Zeit-Umrechnung benötigt man die Schallgeschwindigkeit. Diese ist in LBM per Definition $\frac{1}{\sqrt{3}}$. Wenn man weiss, was die Schallgeschwindigkeit im zu simulierenden Medium ist,

kann man umrechnen, wievielen Sekunden ein Zeitschritt entspricht. Geschwindigkeiten sind in Meter pro Sekunde definiert, daher wird der Umrechnungsfaktor für die Distanz (dx) benötigt.

$$c_s = \frac{1}{\sqrt{3}}$$

$$luftc_s = 340\frac{m}{s}$$

$$dt = \frac{dx}{(\frac{luftc_s}{c_s})}$$

Masse Für die Masse wird die Dichte des Mediums und der Umrechnungsfaktor für Distanz (dx) benötigt.

$$\rho = 1.185\frac{kg}{m^3}$$

$$dm = \rho * dx^3$$

Restliche Einheiten Die restlichen SI-Einheiten ergeben sich nun aus dx, dt und dm.

3. Ergebnisse

Das Hauptergebnis dieser Studienarbeit ist die Software zur Turbinensimulation. Wir haben sie *alcedo* getauft. Der Code ist öffentlich einsehbar unter https://gitlab.com/alcedo-atthis/source.

3.1. Erfüllung der Anforderungen

Anforderung	Erfüllt	Kommentar
Die Software arbeitet parallel.	✓	Wir konnten CUDA wie geplant verwenden und für jeden Zeitschritt werden die Zellen miteinander parallel berechnet.
Es wird das D2Q9 Modell verwendet.	✓	Von Anfang an war nur dieses Modell im Fokus.
Die Simulation wird mit CFD validiert.	✗	Bis zum Schluss konnten die korrekten Kräfte für die Validierung nicht reproduziert werden. Siehe Abschnitt 3.3.
Es liegt ein Benchmark zur Simulation vor.	✓	Die CUDA Anwendung ist um einen Faktor 15-30 schneller, je nach Hardware. Siehe Abschnitt 3.4.
Eine Parameteränderung des Inputs verursacht keine Neukompilation der Software.	✓	Die Applikation kann mit verschiedenen Parametern aufgerufen werden.
Textbasiertes User Interface ist vorhanden.	✓	Die Applikation lässt sich mit der Konsole aufrufen. Es gibt eine Hilfefunktion. Siehe Abschnitt 3.2.
Die Arbeit wird dokumentiert.	✓	Sie halten das entsprechende Dokument gerade in Ihren Händen oder lesen es anderweitig.

Tabelle 3.1.: Erfüllung der Anforderungen

Die einzige Anforderung, die nicht erfüllt wurde, ist die Validierung. Auf diese gehen wir deshalb im Abschnitt 3.3 genauer ein. Da alle anderen Anforderungen erfüllt sind, ist die Tabelle nicht weiter zu kommentieren.

3.2. Bedienungsanleitung

Folgender Text gilt für alcedo Version 1.0.

3.2.1. Kompilieren

Linux

1. Erforderliche Software vorinstalliert: make, proprietäres NVIDIA Kernelmodul, CUDA Treiber und Runtime, CUDA Toolkit 10.0
2. Klonen Sie das Projekt und wechseln Sie in den master-Branch.
3. Navigieren Sie in das Verzeichnis `src` und führen Sie den `make` Command aus.
4. Das Kompilat befindet sich nun unter `src/bin/alcedo`.

Windows

1. Erforderliche Software vorinstalliert: Visual Studio 2017, CUDA Toolkit 10.0
2. Klonen Sie das Projekt und wechseln Sie in den master-Branch.
3. Navigieren Sie in das Verzeichnis `src` und öffnen Sie `Alcedo.sln`.
4. Wählen Sie die `Release` Konfiguration aus.
5. Builden Sie das Projekt.
6. Das Kompilat befindet sich nun unter `src/x64/Release/alcedo.exe`.

3.2.2. Bedienen

Die Bedienung ist auf allen Plattformen gleich. Eine kurze Übersicht erhält man, wenn der Help Command ausgeführt wird, wie in Listing 3.1.

Listing 3.1: `alcedo -h`

```
$ alcedo -h
usage: alcedo -d <this is where the output files go> -[program options]
   --[algorithm options]

program options:
-c        provide an algorithm config file
            format: (you can use all algorithm parameters listed below
   without the prefix dashes)
            % comment
            wind-speed: 23
            turbine-radius: 3
            wing-length: 96
-d*       output directory
-h        display this help
-s        select ouput with acronym(s)
            default: output everything
            selectors: v (veocities), d (densities), t (torque)
            example: -s vt (to omit densities outpt)
-v        display version info

algorithm options:
--count               num of timesteps to process in total
--skip                how many steps to skip before printing a file
```

```
--turbine-radius       radius of the turbine in [m]
--turbine-offset-x     offset of the turbine from the left inlet in
   [cells]
--turbine-drive        speed of the turbine in [1/s]
--wind-speed           speed of wind in [m/s]
--wing-length          length of a single wing in [m]
--wing-num             wing count
--wing-pitch           pitch of the wing in [..]
--wing-profile         wing profile as file or name
                       Available wing profile names:
                         default  examle
                       The file must be in CSV and in the following
   format:
                         alpha;CL;CD
                         alpha;CL;CD
                         alpha;...
--x-size               resolution in X direction
--y-size               resolution in Y direction

arguments marked with a star * are mandatory
```

Die Programm Parameter steuern die Software selber, mit `-v` werden aktuelle Versionshinweise ausgegeben und mit `-h` die Hilfeseite. Bei jedem Programmaufruf ist es erforderlich, den Parameter `-d` zu setzen. Mit diesem gibt man an, wo die Output Files landen sollen. Der Outputselektor `-s` wird im Abschnitt 3.2.2.1 genau erläutert.

Um den Algorithmus zu beeinflussen, werden die Algorithmusparameter verwendet. Diese bestehen aus ganzen Wörtern und beginnen mit zwei Dashes (`--parameter-name`).

3.2.2.1. Spezielle Parameter

`-c`: Konfigurationsdatei Mit dem Parameter `-c` kann eine Konfigurationsdatei (Config) angegeben werden. Die Config kann alle oder eine Auswahl der Algorithmusparameter beinhalten. Um das Programm mit einer Config auszuführen, muss folgendermassen vorgegangen werden:

Listing 3.2: alcedo mit einer Config ausführen und dabei die X-Size überschreiben. Ab Zeile 4 ein Beispiel, wie die Config aussehen könnte.

```
# Verwendung einer Config und Override des X-Size Wertes zur Laufzeit
$ alcedo -d ~/lbm -c ~/.config/lbm/myturbine.cfg --x-size 1400
...
# Aufbau der Config
$ cat ~/.config/lbm/myturbine.cfg
% Percent characters start a line comment

% Timesteps:
count: 10000
skip: 100

% Turbine:
turbine-drive: 4
turbine-offset-x: 200
turbine-radius: 90

% Wind:
wind-speed: 11
```

```
% Wing:
wing-length: 0.23
wing-num: 3
wing-pitch: 0.96
wing-profile: default

% Resolution:
x-size: 1920
y-size: 1080
$
```

Im Codelisting 3.2 ist folgendes sichtbar: Die Config kann alle Algorithmusparameter beinhalten (konsultieren Sie `alcedo -h`, falls nötig), jedoch ohne die vorangestellten Doppeldashes. Es können Kommentare beginnend mit einem Prozentzeichen gemacht werden, diese gelten dann jeweils für die gesamte restliche Linie. Gleichzeitig kann ein Configparameter beim Aufruf des Programs wieder überschrieben werden.

Config Files werden immer automatisch erstellt und in den Outputordner gelegt, sodass ein Algorithmus einfach mit den selben Parametern nochmals gestartet werden kann.

`-s`: Selektoren Da der Output viel Speicherplatz einnehmen kann, kann er eingegrenzt werden. Dafür wird der Selektor Programmparameter verwendet. Auf ihn folgt eine Auswahl des gewünschten Outputs, abgekürzt mit Buchstaben (verfügbare Selektoren können mit `alcedo -h` abgerufen werden). Es können mehrere Selektoren vermischt werden. Wird kein Selektor angegeben, wird immer alles ausgegeben. Listing 3.3 zeigt Beispiele des angewandten `-s` Parameters.

Listing 3.3: Die Anwendung von Selektoren (beachten Sie die Kommentare)

```
# Output nur mit Velocities
$ alcedo -d ~/lbm -s v
...
# Output mit Velocities und Torque
$ alcedo -d ~/lbm -s vt
...
# Kompletter Output
$ alcedo -d ~/lbm
...
$
```

`--wing-profile` Um die Simulation korrekt zu machen, benötigt der Algorithmus ein Turbinenprofil (Wing profile). Mit dem `--wing-profile` Parameter kann dieses angegeben werden. Dabei kann ein Profil verwendet werden, dass in die Software hardcodiert wurde oder es kann ein benutzerdefiniertes Profil verwendet werden. Letzteres ist eine CSV Datei in einem vordefinierten Format. Beachten Sie die Auflistung 3.4 um die korrekte Anwendungsweise des Parameters zu verstehen.

Listing 3.4: Verwendung von `--wing-profile`

```
# Verwendung des example Turbinenprofils
$ alcedo -d ~/lbm --wing-profile example
...
# Verwendung eines Turbinenprofils abgespeichert in einer CSV Datei
$ alcedo -d ~/lbm --wing-profile ~/.config/alcedo/myturbine.csv
...
```

```
# Aufbau der CSV Datei
$ cat ~/.config/alcedo/myturbine.csv
alpha;CL;CD
alpha;CL;CD
alpha;CL;CD
alpha;...
$
```

Gerne hätten wir es den Benutzern und Benutzerinnen erlaubt, direkt eine CSV Datei von Airfoiltools einzubinden. Diese sind jedoch nicht standardmässig aufgebaut, weshalb sie (wie im Listing 3.4 sichtbar) einen anderen Aufbau haben müssen. Die Struktur eines Airfoiltool CSV Files ist in 3.5 festgehalten.

Listing 3.5: CSV von Airfoiltools

```
Xfoil polar. Reynolds number fixed. Mach  number fixed
Polar key,xf-e472-il-500000
Airfoil,e472-il
Reynolds number,500000
Ncrit,9
Mach,0
Max Cl/Cd,69.4851
Max Cl/Cd alpha,9.75
Url,http://airfoiltools.com/polar/csv?polar=xf-e472-il-500000

Alpha,Cl,Cd,Cdp,Cm,Top_Xtr,Bot_Xtr
-17.750,-1.2089,0.08619,0.08243,-0.0227,1.0000,0.0395
-17.500,-1.2392,0.07839,0.07452,-0.0268,1.0000,0.0399
-17.250,-1.2722,0.07024,0.06624,-0.0311,1.0000,0.0402
-17.000,-1.3058,0.06194,0.05780,-0.0356,1.0000,0.0404
-16.750,-1.3374,0.05376,0.04945,-0.0404,1.0000,0.0404
...
```

3.2.3. Visualisierung

Als Output werden Dateien kreiert, im Binärformat, sodass sie schnell geschrieben und gelesen sind. Im mit `-d` angegebenen Ordner wird ein weiteres Verzeichnis angelegt, benannt nach dem aktuellen Datum und Uhrzeit. Darin befinden sich weitere Ordner, einen für jeden Selektor. Zusätzlich befindet sich dort eine Datei, die die gesamten Metadaten archiviert (`alcedo.cfg`), dieselben die beim Start des Programms auf der Konsole ausgedruckt werden. Um einen weiteren Durchlauf des Programms zu starten, mit den selben Attributen, reicht es, wenn man mit dem `-c` Parameter auf diese Datei verweist (siehe 3.2.2.1).

Die Dateien können mittels dem mitgelieferten oder einem eigenen Matlab Script visualisiert werden. Das vorgefertigte Skript befindet sich im `render`-Verzeichnis im Root des Projektes. Bevor es ausgeführt werden kann, muss die `input` Variable korrekt gesetzt werden, und zwar muss in diesem String das Verzeichnis angegeben werden, wo sich die berechneten Dateien befinden.

`Visualizer.m` Das Skript durchsucht den Ordner und erstellt für jeden Selektor die entsprechenden Bilder. Diese werden im selben Ordner im Unterordner `_render` abgelegt, mit der selben Verzeichnisstruktur wie schon die Outputdaten von Alcedo abgelegt wurden. Man kann das Skript ausführen, während Alcedo noch am Berechnen ist, dann werden nur die schon verfügbaren Resultate visualisiert. Wird das Script später nochmals aufgerufen, werden nur die noch nicht berechneten Dateien visualisiert. Abbildung 3.1 zeigt die Visualisierung der Strömung. Für jeden

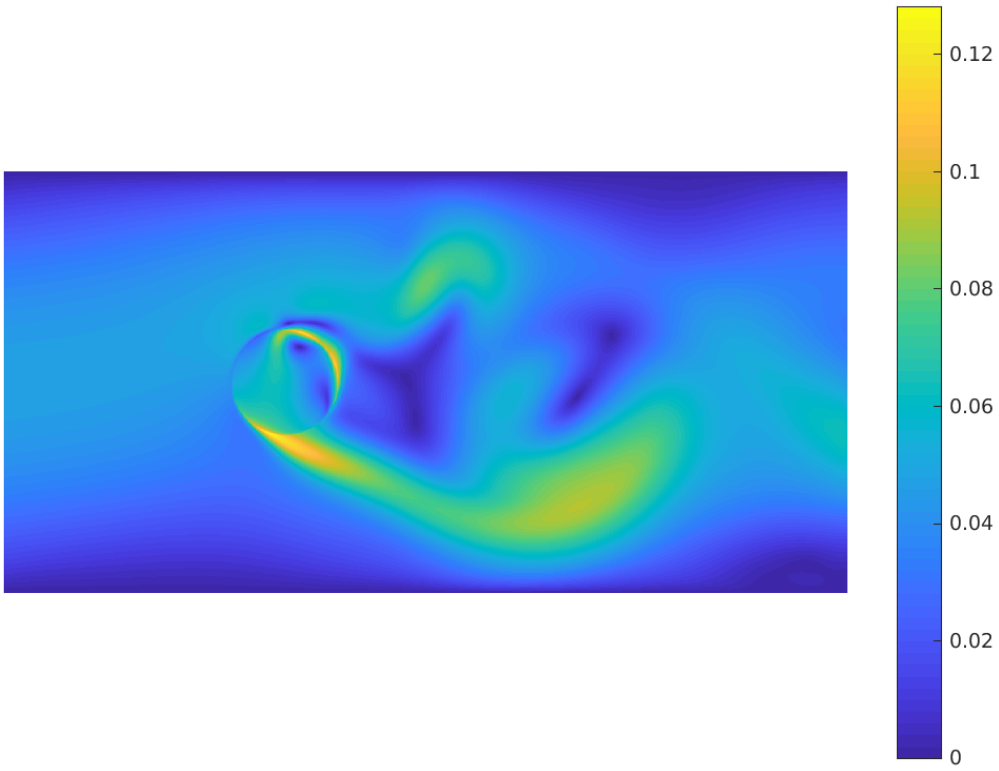


Abbildung 3.1.: Der mit Matlab visualisierte Output der Velocities einer Strömung mit dazwischenliegender Windturbine.

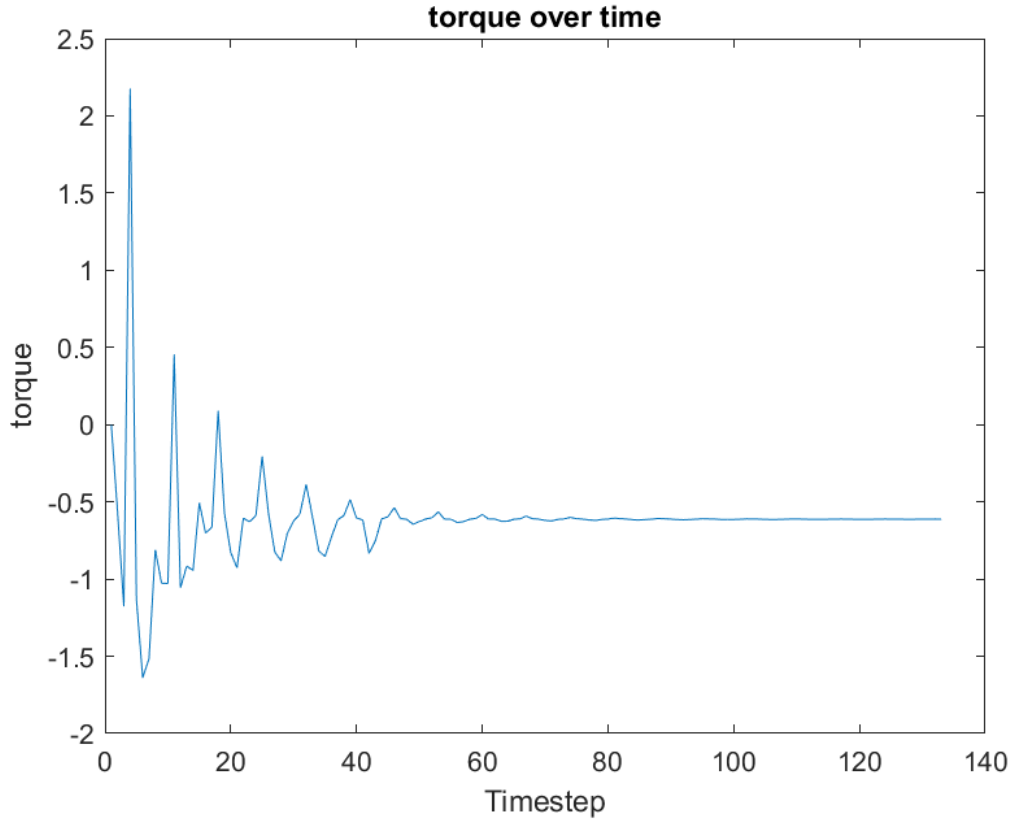


Abbildung 3.2.: Das gesamthaft wirkende Drehmoment auf die Turbine pendelt sich mit der Zeit ein. (Visualisierung mit Matlab)

Zeitschritt wird ein solches Bild erstellt. Über die gesamte Simulation kann das Drehmoment berechnet und visualisiert werden, wie in Abbildung 3.2.

Um die Bilder zu berechnen, benötigt Matlab die X- und Y-Size des resultierenden Bildes. Diese haben wir aus Bequemlichkeitsgründen für die User_innen in den Outputdaten abgelegt, jeweils als erstes Element im Array. Hätten wir das nicht getan, müsste der Benutzer oder die Benutzerin vor jedem Rendervorgang die entsprechenden Werte in das Matlab Skript einfügen.

3.3. Validierung

Um den Output der parallelen Implementierung zu testen wurde ein Hindernis in den Luftstrom eingeführt und die totale Kraft berechnet, die darauf wirkt. Dieser Output wurde mit dem von anderen CFD Tools Palabos und CFX verglichen.

Berechnung der Kraft

$$F(x,y) = 2 * \sum_{i=1}^{9} f(x,y,i) * c(i)$$

Für jede Zelle des Hindernisses wurden die Partikelverteilungen entsprechend ihrer Richtung aufsummiert und mit einem Faktor 2 multipliziert, da die Teilchen perfekt reflektiert werden. Die totale Kraft entspricht der Summe aller Teilkräfte. Diese ist noch einheitslos und muss mit der Einheit für Kraft dm/dt^2 multipliziert werden um Newton zu erhalten.

Resultat Die Kraft die von CFX errechnet wurde war $\sim 43\ N$ (Newton). Palabos berechnete ähnliche 39 Newton, während die CUDA Implementation eine Kraft von $\sim 150\ N$ errechnet hat. Diese grosse Diskrepanz konnte bis zum Ende des Projektes nicht erklärt werden. Der Code für die Kraftberechnung wurde extra für die Validierung implementiert und wird in der Anwendung für den Wirkungsgrad der Windturbine nicht verwendet. Deshalb ist nicht auszuschliessen, dass der Fehler rein in der Berechnung der Kraft liegt und der restliche Code korrekt funktioniert. Die Tatsache, dass die erreichten Geschwindigkeiten des Fluides mit denen von CFX und Palabos übereinstimmen, würde diese These unterstützen.

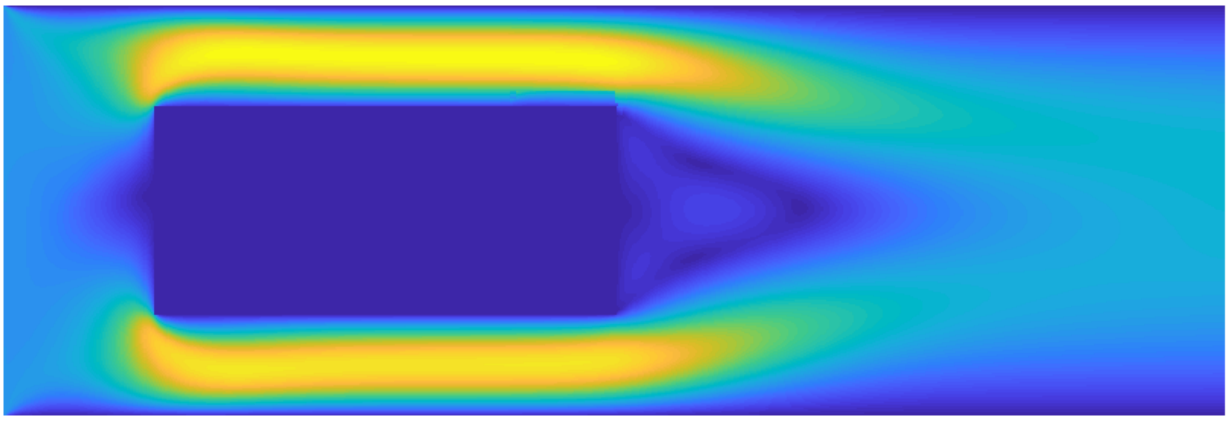


Abbildung 3.3.: Hindernis im Luftstrom mit alcedo

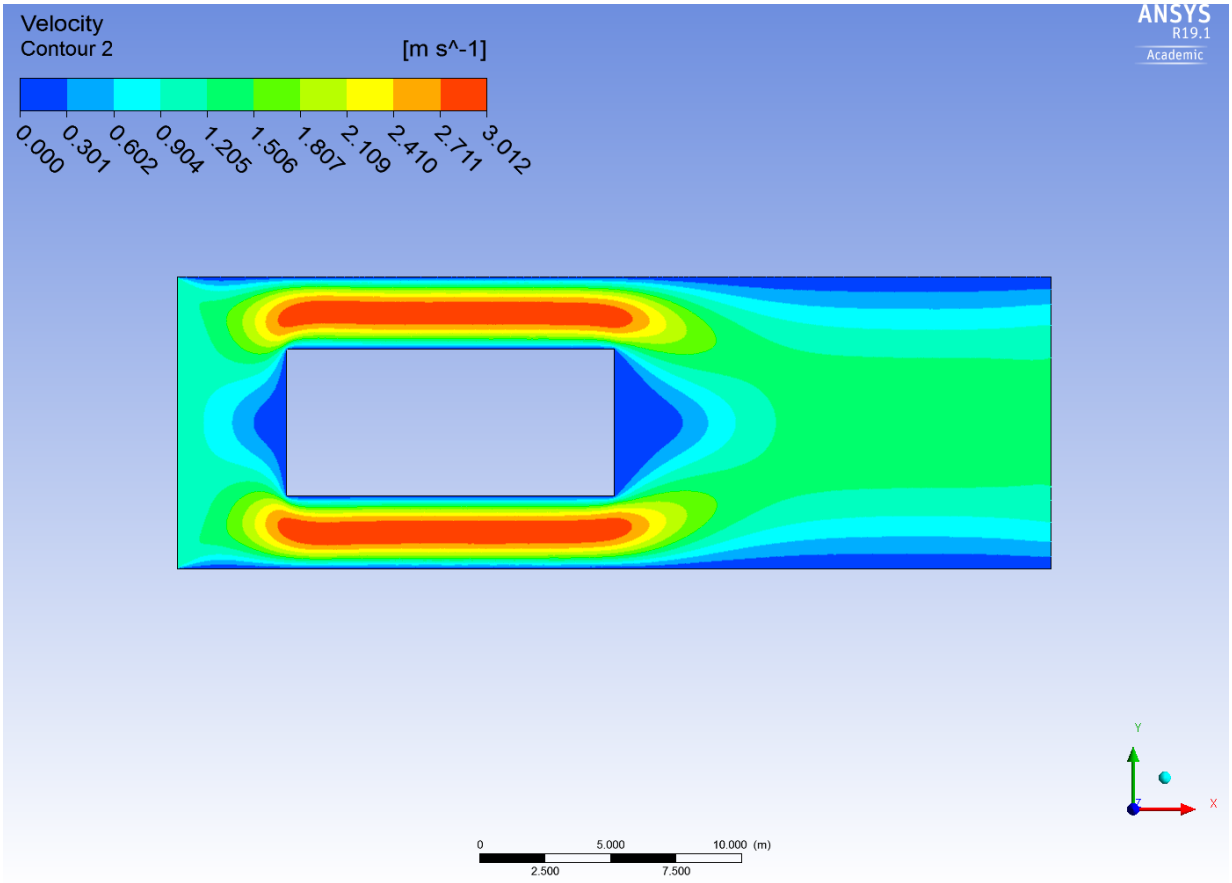


Abbildung 3.4.: Hindernis im Luftstrom mit CFX

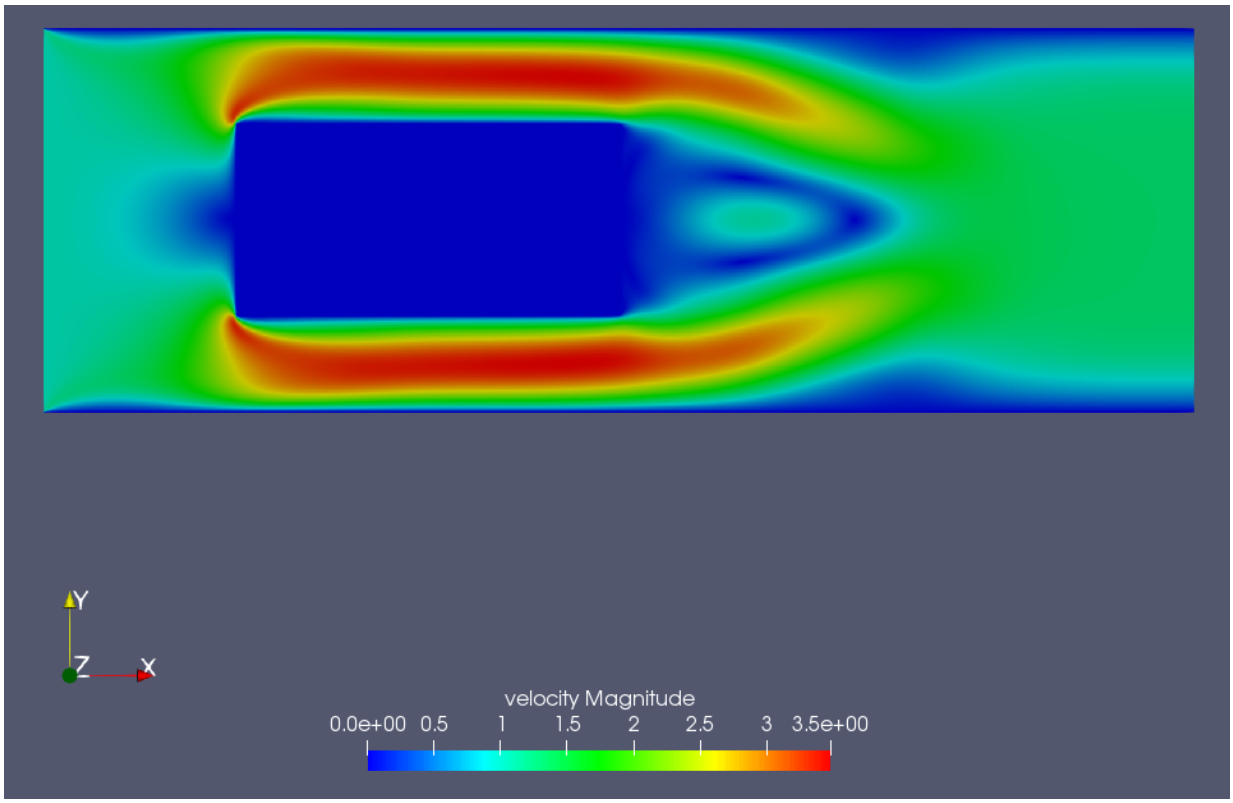


Abbildung 3.5.: Hindernis im Luftstrom mit Palabos LBM Solver

3.4. Benchmarking

Für den Performancevergleich wurden mehrere Implementationen verglichen. Der Matlab Code war mangels Implementation nicht identisch mit den anderen Kandidaten, da keine Flügelkraft berechnet wurde. Es wurde lediglich die Geschwindigkeit und Dichte pro Zeitschritt berechnet. In allen anderen wurde diese berechnet, weshalb der Vergleich ein wenig zu Gunsten der Matlab-Implementation ausfällt. Die Performancetests wurden auf einem Lenovo Thinkpad Laptop ausgeführt, sowie auf einem Cluster für den Test mit der Tesla V100 Grafikkarte.

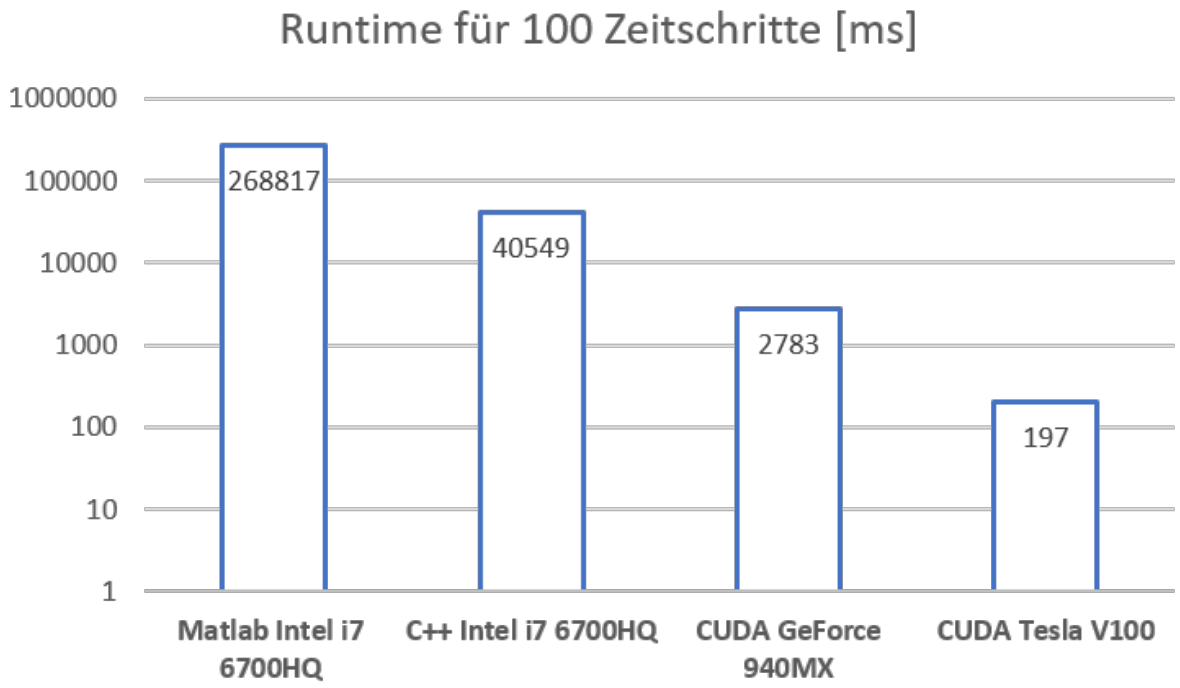


Abbildung 3.6.: Laufzeit für die Berechnung von 100 Zeitschritten in einem 2000x1000 Grid

Der aussagekräftigste Sprung ist der von der seriellen C++ zur parallelen CUDA Implementation. Diese Tests verwenden den identischen Code, einmal seriell und einmal parallel ausgeführt. Der CUDA-Code ist um einen Faktor 15 schneller. Als Vergleich: Für eine 60 Sekunden lange Simulation ist das ein Unterschied zwischen einer Laufzeit von 30 Tagen und einer von 2 Tagen. Wenn die Applikation auf einer schnelleren GPU wie der Tesla V100 ausgeführt wird, kann die Dauer um einen weiteren Faktor 14 verringert werden. Somit erreicht man eine Laufzeit von 3.5 Stunden.

Testkonfiguration

Listing 3.6: Konfiguration für Performancetests

```
% Timesteps:
count: 10000
skip: 100

% Turbine:
turbine-drive: 4.000000
turbine-offset-x: 666
turbine-radius: 63

% Wind:
wind-speed: 15.000000

% Wing:
wing-length: 0.200000
wing-num: 3
wing-pitch: 0.000000
wing-profile: default
```

```
% Resolution:
x-size: 2000
y-size: 1008
```

4. Schlussfolgerung

Im Kapitel 3 gehen wir sehr genau auf die Ergebnisse des Arbeitsinhaltes ein, deshalb werden wir das hier nicht mehr tun. Trotzdem sollen sie hiermit noch beurteilt werden.

4.1. Erreichung der Zielsetzung

Die Validierung konnte nicht zu unserer Zufriedenstellung abgeschlossen werden. CFX, sowie Palabos errechneten Kräfte, die stark von der CUDA Implementierung abweichen. Die Windgeschwindigkeiten waren jedoch beinahe identisch, was interessant ist, da die Kräfteberechnung ausschliesslich auf den gleichen Variablen beruht, wie die Berechnung der Geschwindigkeit. Trotzdem ist es aus unserer Sicht bedauernswert, dass die Studienarbeit nicht mit einem Kompletterfolg abgeschlossen werden kann. Dass wir die anderen Anforderungen alle erfüllen ist erfreulich und unserer Meinung nach haben wir die Konsolenapplikation komfortabel für den Endanwender oder die Endanwenderin programmiert (zum Beispiel mit der Einführung der Config Dateien).

4.2. Weiteres Vorgehen

Ziel dieser Dokumentation ist es unter anderem eine Weiterentwicklung zu ermöglichen. Als erstes muss sichergestellt werden, dass die Simulation genügend genau ist. Dazu könnte die gleiche Validierungsmethode verwendet werden. Eine Vermutung, wieso der bestehende Code Kräfte berechnet, die viel zu gross sind, ist die Strömung innerhalb des Hindernisses. Das Hindernis wurde einzelns für die Valdierung eingeführt und es kann gut sein, dass die Kollisionslogik mit Hindernissen nicht vollständig korrekt ist. Wenn der Output des Tools korrekt ist, sind diverse Weiterentwicklungen der Software möglich:

- Erweiterung des Gitters auf einen dreidimensionalen Raum (D3Qx)
- Grafisches User Interface
- Client-Server Architektur, wobei der Client die Software nur mit Parametern füttert und die Resultate empfängt. Die Berechnungen werden auf dem Server ausgeführt.
- Implementierung eines bewegten Flügelprofils anstatt der gemittelten Turbine

Teil III.

Anhänge

1. Persönliche Schlussberichte

1.1. Mario Tarreghetta

Die wichtigste Erkenntnis für mich war die Bedeutung von klaren Anforderungen und Abnahmekriterien. Wir sind während der Studienarbeit ziemlich ≪blindlings≫ vorgegangen, was teils in der Natur des Projektes lag, da es sich eher um eine Forschungsarbeit, als ein klar definiertes Projekt gehandelt hat. Trotzdem haben wir darunter gelitten, nicht immer zu wissen, was überhaupt genau von uns erwartet wurde und was als nächstes bearbeitet werden soll. Es war eher ein ≪drauflos≫-Arbeiten, was auch einen Grossteil der Planung verunmöglicht hat.

Mit dem Resultat bin ich grösstenteils zufrieden, da wir doch ein sehr komplexes Thema verarbeiten und implementieren konnten. Ein wenig nervt es mich aber, dass die Validierung zum Schluss um so viel daneben gelegen ist, zumal die Fluidgeschwindigkeiten sehr ähnlich mit denen von kommerziellen CFD Tools sind. Vorallem zu Beginn war ich mit der ganzen Umgebung von wissenschaftlichen Papers und der vielen mathematischen Notation ein wenig überfordert. Im Laufe des Projektes fand ich mich aber immer besser zurecht.

Weiter nehme ich für künftige Projekte mit, nie mehr GitLab zu verwenden, da die Zeiterfassung sehr dürftig war und wir am Schluss nicht einen Report über die ganze verbuchte Zeit generieren konnten.

1.2. Sebastian Hug

Noch nie habe ich vorher von der Boltzmann Gleichung und deshalb natürlich auch nicht von der Lattice-Boltzmann-Methode gehört. Sich von null auf hundert mit einem komplett neuen Thema auseinanderzusetzen war spannend. Alles war zwar nicht neu, das Kräfte und Massen im System erhalten bleiben war mir vorher schon klar. Das ist jedoch ein kleiner Teil. Zum ersten Mal habe ich auch mit CUDA gearbeitet, sowie (neben dem Schulunterricht) mit C++ und C. Dabei war es ein Vorteil, dass wir diese beide Sprachen schon im Bsys- und C++- Modul gelernt haben.

Dass alles neu für mich war, stellte mich vor ein paar Herausforderungen. Mit CUDA und den Sprachen war ich schnell vertraut, was mir am meisten Mühe bereitete, war der Algorithmus. Mittlerweile verstehe ich jeden Teil vom Code, es dauerte jedoch jeweils einen Moment, bis ich wieder einen Part verstanden habe. Viele Fragen lösten sich erst während der Implementierung. Diese nahm dadurch ein bisschen mehr Zeit in Anspruch. Obwohl wir die mathematischen und physikalischen Kenntnisse über die Lattice-Boltzmann-Methode eigentlich nicht benötigt hätten, war es schon mein Anspruch, die Methode so gut wie möglich zu verstehen. Ich war froh, dass sich die Betreuer immer und immer wieder Zeit genommen haben, die Einzelheiten zu erklären.

Dass wir kein GUI programmieren werden war mir ziemlich schnell klar. Trotzdem hat mir diese Arbeit gefehlt, und es gefiel mir deshalb, die Konsolenapplikation mit kleinen Features zu versehen, zum Beispiel die Verwendung von Konfigurationsdateien. Obwohl das auch seine Schwierigkeiten mit sich brachte, war es einiges entspannter als die Algorithmusimplementierung.

Über alles gesehen habe ich gerne an dieser Studienarbeit gearbeitet. Es gefiel mir, dass wir neues Terrain betreten haben und Dinge ausprobieren konnten, da die Arbeit keine traditionelle Softwareentwicklung beinhaltete.

2. Sitzungsprotokolle

2.1. 19. September 2018

Anwesend Henrik Nordborg, Mario Tarreghetta, Sebastian Hug

Notizen Erster Schritt: 2 dim. Lattice-Boltzmann von bestehendem Code verstehen; Einarbeitung in Thema

Bei Fragen auch an Alain Schubiger wenden

Betreuung: jeden Mittowch um 14:00 Uhr (Herr Nordborg verschickt Termin)

Der Schlussbericht:

- Bericht über das Produkt
- Lattice-Boltzmann erklären
- ≪für technischen User≫
- Anforderungen von Dokument auf Skripteserver

2.2. 26. September 2018

Anwesend Henrik Nordborg, Alain Schubiger, Mario Tarreghetta, Sebastian Hug

Protokoll

- Besprechung Stand
- Fragen zu `f` beantwortet
- Administrative Fragen geklärt (siehe unten)

Administrative Fragen

- Dokumentation? HN meint, wir sollten Mitte Semester entscheiden, da es kein traditionelles Softwareprojekt ist. Wichtig ist eine Art Tool mit GPU Parallelisierung.
- Protokolle? SH versendet ein Link zum Wiki.

2.3. 2. Oktober 2018

Anwesend Henrik Nordborg, Mario Tarreghetta, Sebastian Hug

Protokoll

- Code angeschaut, HN wird mit AS den Code/das Resultat überprüfen
- Requirements geklärt

Requirements

- Wie gross soll die Auflösung in Bild- und Zeitdimension sein? (HN klärt Systemgrösse ab, wahrscheinlich konfigurierbar)
- Reicht `float`? (HN: float sollte reichen, evtl. mit Byte direkt rechnen/ASCII)

Weiteres Vorgehen

- Weitermachen mit CUDA Implementierung
- Falschen Prototype löschen

2.4. 10. Oktober 2018

Anwesend Henrik Nordborg, Alain Schubiger (via Skype), Mario Tarreghetta, Sebastian Hug

Zu klärende Fragen

1. Wie sollen wir den Inflow simulieren?
2. Was ist Poiseuille?
3. Was ist die Reynolds Number und wieso hängt sie von der Y-Achse ab?
4. Was sind Zou/He Boundaries?

Und die Antworten dazu:

1. Collision: Komponenten, die das System verlassen wuerden, werden umgekehrt.
2. Hat sich erledigt
3. HN erklärte es auf einem Blatt. Hohe Viskosität == tiefe Reynoldszahl.
4. Für die Boundaries. Wir sollten das benutzen.

Protokoll

1. Prototyp Code angeschaut (LBM2), mit Matlab Code verglichen
2. Fragen geklärt

Nächste Schritte

- Outflow korrigieren
- Boundaries mit Zou/He machen
- HN gibt uns Matlab Files und Link zur Matlab Software

2.5. 17. Oktober 2018

Anwesend Henrik Nordborg, Alain Schubiger, Mario Tarreghetta, Sebastian Hug

Zu klärende Fragen

1. Wieso wird es bei hoher Reynoldsnumber instabil?
2. Outlet bei hoher RNr. vergleichen, Alain?
3. Sind wir so weit wie erwartet?
4. Wie soll die Turbine implementiert werden?

Und die Antworten dazu:

1. Das ist normal. Wir sollten das Turbulenzmodell einsetzen, damit kann die Reynoldsnumber erhöht werden, bei gleich bleibender Auflösung.
2. Siehe 1. Resultate wie erwartet.
3. Gut so.
4. Alain hat etwas recherchiert. Paper ≪Lattie boltzmann method for fluid dynamics - <Jen Peng>≫ Punkt 2.4.54+
   - Kraft ausrechnen
   - Zusatzterm (2.4.54) ausrechnen
   - Zusatzterm wird zu Equilibrium dazugerechnet

Weitere Gesprächsthemen

1. 3 Tracks: Flügel, Turbulenzmodell, Optimierung (profiling)
2. Henrik schreibt die Aufgabenstellung
3. Visualizer in Matlab, Parametrisierung, Programm von Matlab aufrufen

Weiteres Vorgehen

- Zusätzlichen Kraft-Term implementieren
- Outlet für hohe Reynoldszahlen flicken (hohe Reynoldszahlen sind wichtig für das Endprodukt)

2.6. 24. Oktober 2018

Anwesend Henrik Nordborg, Mario Tarreghetta, Sebastian Hug

Zu klärende Fragen

1. Wie seht es mit dem Benchmarking aus? Haben wir eine Referenzmaschine?
2. In welchem Rahmen werden sich die benutzten Reynoldszahlen bewegen?

Und die Antworten dazu:

1. Benchmarking: nicht so wichtig, evtl. Matlab Code vergleichen auf Adrians Maschine mit Cuda Code.
2. Ab 1′000′000 aufwärts

Knowhow Später wäre es gut wenn ein Flügelprofil hinterlegt werden kann, z.B. wie auf http://www.airfoiltools.com/airfoil/details?airfoil=naca0012h-sa.

Weiteres Vorgehen

- Henrik schaut mit Adrian wegen Grafikumgebung
- MT und SH versuchen, Binary Linuxfähig zu machen
- Kraftfunktion nach `u` aufgelöst von Henrik
- Henrik gibt Turbulenzmodellierung vor

2.7. 31. Oktober 2018

Ausfall Die Besprechung wurde aufgrund der Abwesenheit von Henrik und Alain abgesagt.

Zu klärende Fragen

1. Wir würden gerne mit der Dokumentation beginnen: Ist die gezeigte Struktur in Ordnung?
2. Hindernisse als Kraftterm nicht möglich/sinnvoll, da Beschleunigung. Was tun?
3. (per Mail) Wir haben soeben festgestellt, dass die maximale Auflösung die wir aufgrund GPU-Memory, Disk-space und RAM produzieren können bei 12'000 x 12'000 liegt. Ist das genügend?
4. (per Mail) Bezüglich Turbulenzmodell benötigen wir noch Input von Dir oder Alain. Die Hindernisse als Kraft zu implementieren hat nur beschränkt funktioniert, da so wie es momentan implementiert ist, die Hindernisse das Fluid beschleunigen würden, was wahrscheinlich nicht sinnvoll ist.

Die Antworten dazu per Mail von Henrik (31. Oktober 2018):

1. -
2. siehe 4.
3. -
4. (per Mail) ≪Gestern Abend habe die Funktion für die Kraft fertiggestellt. Ich werde diese noch dokumentieren und zusammen mit der Aufgabenstellung schicken. Die Turbulenz muss ich mir noch anschauen.≫

2.8. 6. November 2018

Anwesend Henrik, Mario, Sebastian

Zu klärende Fragen

1. Die maximale Auflösung, bedingt durch GPU-Memory, Disk-space und RAM, liegt momentan bei 12'000 x 6'000. Ist das ein Problem? (auf Adrian's Rechner)
2. Dürfen wir Matlab auf den Superrechnern installieren?

Und die Antworten dazu:

1. Reicht! HD ist gut.
2. Ist schon installiert `module load MATLAB/<version>`

Protokoll

1. Explizite und implizite Methoden besprochen. Wir haben die explizite Methode verwendet, sollten aber die implizite implementieren (Definition 14 im Dokument von Henrik). Somit wird das Turbulenzmodel vorübergehend hinfällig, der Algorithmus ist stabil. Das ist eine Näherung, sie verhindert die Explosion, falls Omega sehr gross wird.
2. Windturbine: Die Kraft wird zur Zeit verteilt.

Weiteres Vorgehen

- Henrik schickt uns sein Dokument und teilt den Matlab Code (Switchdrive)
- Termin morgen findet statt, falls nichts anderes kommuniziert wird (zum Beispiel für die Aufgabenstellung)

2.9. 14. November 2018

Anwesend Henrik, Mario, Sebastian

Protokoll

1. Stand
2. Fragen
3. Anforderungen angeschnitten

Zu klärende Fragen

1. Instabilität? Grössenordnung der Kräfte? Momentan Werte zwischen -40/40
2. Wirkt die Flügelkraft immer, auch wenn der Zylinder nicht angeströmt wird? Drehung?
3. Was bedeutet der NCrit Parameter auf airfoiltools.com?

Und die Antworten dazu:

1. Siehe Blatt
2. Siehe Blatt: Kraftanwendung anders machen: siehe Kommentare im Code. Geschwindigkeiten dimensionslos machen: vT wird viel kleiner.
3. Das weiss niemand so genau. Steht auf Webseite. Tiefen Wert verwenden.

Bemerkungen

- `pow(...)` nicht verwenden
- Parameter: Windgeschwindigkeit m/s (`MAX_VELOCITY`), Anzahl Flügel, Länge Flügel, Radius Windturbine, Flügelprofil (Auswahl geben, NCrit tief), Drehzahl Windturbine (konstant)
- Output in physikalischen Einheiten
- evtl. Flügel wandern lassen, anstatt gemittelt (falls Zeit)

2.10. 27. November 2018

Anwesend Alain, Henrik, Mario, Sebastian

Zu klärende Fragen

1. Ab wann (Timestep) kann man den Wirkungsgrad berechnen?
2. Wie genau Wirkungsgrad berechnen? Drehmoment? (Welche Variablen)
3. Instabilität bei Turbine
4. Kann man einfach durch den Umfang teilen? (Aufteilung der Kraft auf die Zellen, z.B. Kreis = 20 Blöcke, Umfang = 18.84)
5. Wieso müssen wir die Drehzahl durch die ≪Unit-Of-Speed≫ teilen? (Wurzel 3 * Schallgeschwindigkeit) und ist der berechnete Wert korrekt?
6. Bericht / Dokumentation?
7. Welche Dimensionen sollen wir verwenden? (Grösse der Turbine im Verhältnis zum Bild, Format, Abstand zum Rand)

Und die Antworten dazu:

1. Sollte irgendwann stabil werden.
2. Drehmoment: $Radius * F$. Tangentiale Kraft: geht mit `output[0/1]` (siehe Blatt Henrik) ($M = xfy - yfx$). Die einzelnen dann aufsummieren. Sollte eine normalverteilte Kurve (Graph Drehgeschwindigkeit, M) geben.
3. (normal)
4. ja
5. x
6. Kurz diskutiert. Es ist ein Forschungsbericht, keep that in mind. Tests: Validierung. Machen wir mit dem Klotz. Benchmarking.
7. Selber ausrechnen

Next steps

- Einfaches Hindernis einbauen (Würfel)
- Units (siehe Blatt)
- Versuchen, normalverteilte Kurve hinzukriegen

2.11. 28. November 2018

Anwesend Alain, Henrik, Mario, Sebastian

Fragen / Besprechungspunkte

1. Wingdrive in m/s oder Winkelgeschwindigkeit?
2. Schallgeschwindigkeit parametrisiert? Für unit-of-time verwenden?
3. Validierung: Gemeinsamer Termin oder hin-und-her-senden von Daten?
4. Validierung: Einigung auf Parameter & Impulsänderung in welcher Form?

Antworten/Resultate:

1. Tangentialgeschwindigkeit der Turbine ((Winkelgeschwindigkeit omega (rad/s) * Radius) / Einlasswindgeschwindigkeit). Müsste etwa 3-4 sein.
2. Über die Viskosität. Viskosität hat Einheit kg/m/s . masterThesis.pdf Seite 11 (19). Windgeschwindigkeit. Viskosität dynamisch auch fix (Luft = 1.831e-5)
3. Nettokraft. Richtungswechsel aufsummieren und multiplizieren mit ... (Impulsübertrag: Masse * 2*v), Kraft ist Impulsänderung pro Zeit, Vektorsubtraktion. Überall: vorne, oben, unten und vor allem hinten. Nettokraft, Summe der y-Kräfte müsste 0 sein. Wir schicken ihm das.
4. Wir machen etwas und Henrik stellt es in CFD nach (Force per length)

Info

- Drehmoment in Nm

Weitere Schritte

- Reynoldszahl rausnehmen (siehe 2.)
- Bilderoutput: Dichte + Geschwindigkeit optional machen
- Validierung: Nettokraft berechnen und an Henrik senden

2.12. 5. Dezember 2018

Anwesend Alain, Henrik, Mario, Sebastian

Protokoll

1. Mail nicht angekommen, besprochen während Sitzung: Outputs unterschiedlich um Faktor 12.

Weiteres Vorgehen

- Wir machen erneut eine Simulation mit hoher Viskosität und ohne Omegakorrektur
- Termin sobald neue Resultate vorhanden

2.13. 12. Dezember 2018

Anwesend Alain, Mario, Sebastian

Protokoll

1. Validierung: Da die Zeit schon fortgeschritten ist, geben wir uns mit dem Resultat vorläufig zufrieden. Die Abweichung zwischen Palabos und Alcedo beträgt momentan etwa 10 Nm. Die Geschwindigkeiten stimmen überrein.
2. Benchmarking: Matlabvergleich machen
3. Fragen geklärt

Zu klärende Fragen

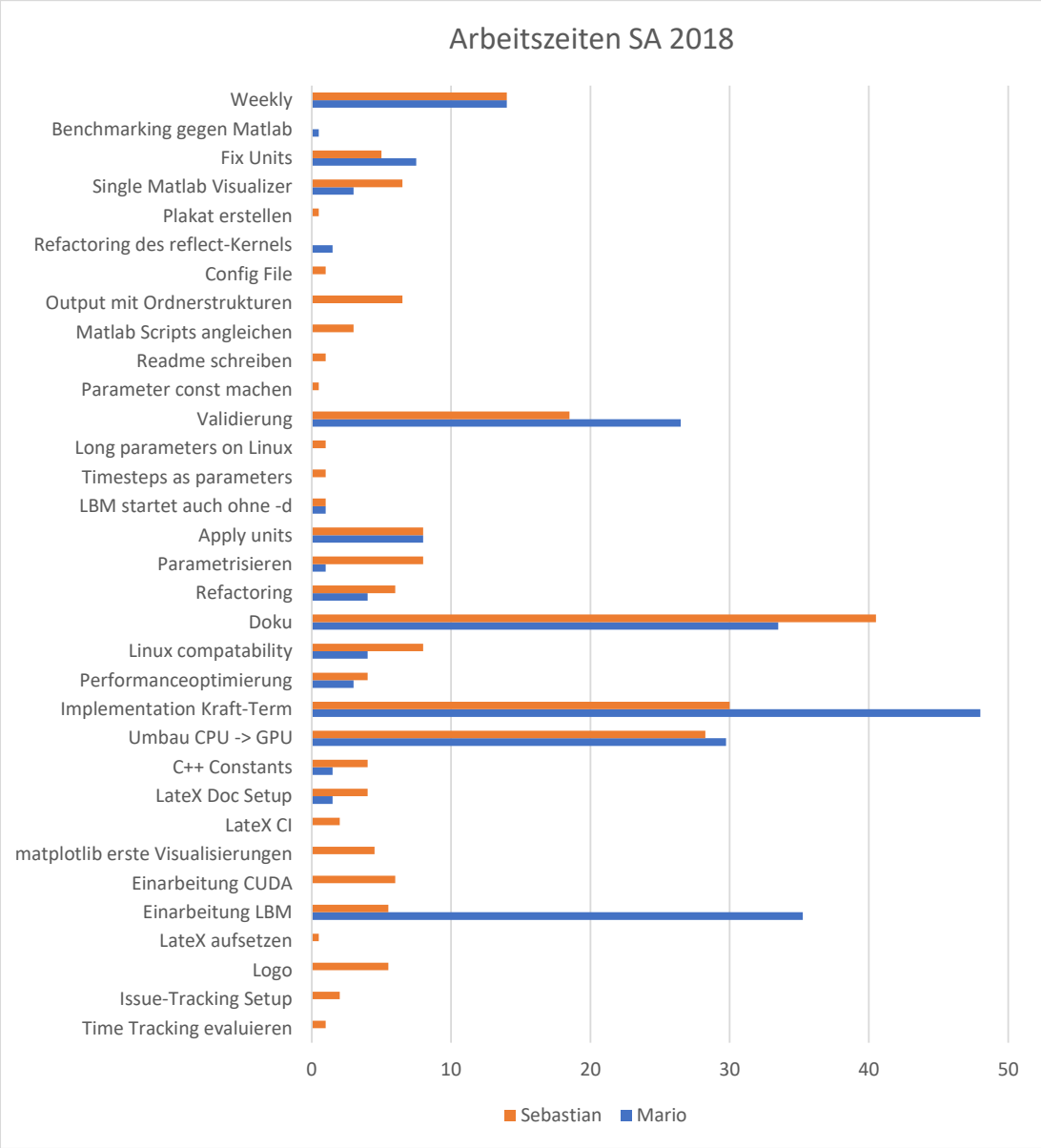
1. Unterschriebene Aufgabenstellung → Scan
2. Default Flügelprofil
3. Benchmarking gegen was? Palabos?

Und die Antworten dazu:

1. Machen wir am nächsten Mittwoch
2. Lieber schauen, dass CSV von Airfoiltools direkt importierbar ist
3. Evtl. noch gegen Matlab

3. Zeiterfassung

Arbeitspaket	Mario	Sebastian
Time Tracking evaluieren	0	1
Issue-Tracking Setup	0	2
Logo	0	5,5
LateX aufsetzen	0	0,5
Einarbeitung LBM	35,25	5,5
Einarbeitung CUDA	0	6
matplotlib erste Visualisierungen	0	4,5
LateX CI	0	2
LateX Doc Setup	1,5	4
C++ Constants	1,5	4
Umbau CPU -> GPU	29,75	28,25
Implementation Kraft-Term	48	30
Performanceoptimierung	3	4
Linux compatability	4	8
Doku	33,5	40,5
Refactoring	4	6
Parametrisieren	1	8
Apply units	8	8
LBM startet auch ohne -d	1	1
Timesteps as parameters	0	1
Long parameters on Linux	0	1
Validierung	26,5	18,5
Parameter const machen	0	0,5
Readme schreiben	0	1
Matlab Scripts angleichen	0	3
Output mit Ordnerstrukturen	0	6,5
Config File	0	1
Refactoring des reflect-Kernels	1,5	0
Plakat erstellen	0	0,5
Single Matlab Visualizer	3	6,5
Fix Units	7,5	5
Benchmarking gegen Matlab	0,5	0
Weekly	14	14
	223,5	**227,25**


Arbeitszeiten SA 2018
Weekly
Benchmarking gegen Matlab
Fix Units
Single Matlab Visualizer
Plakat erstellen
Refactoring des reflect-Kernels
Config File
Output mit Ordnerstrukturen
Matlab Scripts angleichen
Readme schreiben
Parameter const machen
Validierung
Long parameters on Linux
Timesteps as parameters
LBM startet auch ohne -d
Apply units
Parametrisieren
Refactoring
Doku
Linux compatability
Performanceoptimierung
Implementation Kraft-Term
Umbau CPU -> GPU
C++ Constants
LateX Doc Setup
LateX CI
matplotlib erste Visualisierungen
Einarbeitung CUDA
Einarbeitung LBM
LateX aufsetzen
Logo
Issue-Tracking Setup
Time Tracking evaluieren
0
10
20
30
40
50
Sebastian
Mario

4. Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Semesterarbeit HS 2018

GPU-optimiertes Simulationstool für die Auslegung von vertikalachsigen Windturbinen

Studierende: Sebastian Hug, Mario Tarreghetta

Betreuer: Henrik Nordborg, Alain Schubiger

1 Ausganslage

Die Auslegung vertikalachsiger Windturbinen ist im Moment sehr anspruchsvoll. Vereinfachte Berechnungstools funktionieren zwar, sind aber sehr ungenau und Computational Fluid Dynamics (CFD) sehr zeitintensiv. Mit der Arbeit soll ein neues Berechnungstool geschaffen werden, das eine schnelle Optimierung der Windturbine ermöglicht.

Kern des Berechnungstools ist die Lattice Boltzmann Methode zur schnellen Berechnung der Strömung. Da sich diese auf GPUs hervorragend parallelisieren lässt, soll die Implementierung gleich mit CUDA gemacht werden.

2 Aufgabenstellung

Folgende Arbeiten sind zu bewerkstelligen:

1. Einarbeitung in den Lattice Boltzmann Algorithmus (LBM)
2. Implementierung und Parallelisierung eines 2D-Strömungslösers mit LBM
3. Validierung mit einem 2D Objekt in CFD
4. Implementierung der vertikalachsigen Windturbine (als Kraft auf die Strömung)
5. Validierung der Simulation
6. Benchmarking
7. Einfaches User Interface (textbasiert oder in Matlab)
8. Dokumentation

3 Bewertung

Das Ergebnis der Arbeit und der Bericht werden bewertet.

Henrik Nordborg

Erneuerbare Energien und Umwelttechnik
Prof. Dr. Henrik Nordborg ■ henrik.nordborg@hsr.ch ■ T +41 55 222 4370
Hochschule für Technik Rapperswil ■ IET ■ Oberseestrasse 10 ■ CH 8640 Rapperswil
www.hsr.ch/eeu

5. Matlab

5.1. channelWithWall.m

Listing 5.1: channelWithWall.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% channelWithWall.m: Channel flow past a wall
%                    , using a LB method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lattice Boltzmann sample in Matlab
% Copyright (C) 2006-2008 Jonas Latt
% Address: EPFL, 1015 Lausanne, Switzerland
% E-mail: jonas@lbmethod.org
% Get the most recent version of this file on LBMethod.org:
%   http://www.lbmethod.org/_media/numerics:cylinder.m
%
% Original implementaion of Zou/He boundary condition by
% Adriano Sciacovelli (see example "cavity.m")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation; either version 2
% of the License, or (at your option) any later version.
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
% You should have received a copy of the GNU General Public
% License along with this program; if not, write to the Free
% Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
% Boston, MA  02110-1301, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edited: 2018-10-03 by aschubig@hsr.ch
% Based of cylinder.m by JLatt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear

% GENERAL FLOW CONSTANTS
lx      = 400;       % number of cells in x-direction
ly      = 400;       % number of cells in y-direction
% obst_x = lx/5+1;   % position of the cylinder; (exact
% obst_y = ly/2+3;   % y-symmetry is avoided)
% obst_r = ly/10+1;  % radius of the cylinder
obst_l = ly/4*3;
uMax    = 0.05;      % maximum velocity of Poiseuille inflow
Re      = 100000;      % Reynolds number
nu      = uMax * obst_l / Re;  % kinematic viscosity
omega   = 1. / (3*nu+1./2.);      % relaxation parameter
maxT    = 400000;  % total number of iterations
```

```
46  tPlot   = 50;       % cycles
47
48  % Relaxation factor
49  wt = 1 / ( 1 + omega );
50
51
52  % D2Q9 LATTICE CONSTANTS
53  t  = [4/9, 1/9,1/9,1/9,1/9, 1/36,1/36,1/36,1/36];
54  cx = [  0,   1,  0, -1,  0,    1,  -1,  -1,   1];
55  cy = [  0,   0,  1,  0, -1,    1,   1,  -1,  -1];
56  opp = [ 1,   4,  5,  2,  3,    8,   9,   6,   7];
57  col = [2:(ly-1)];
58  in  = 1;   % position of inlet
59  out = lx;  % position of outlet
60
61  [y,x] = meshgrid(1:ly,1:lx); % get coordinate of matrix indices
62  obst = zeros(lx,ly);
63
64  % Specify the obstacle
65  %for i=ly-obst_l:obst_l
66  %    obst(lx/10,i) = 1;
67  %end
68  % obst = ...                    % Location of cylinder
69  %     (x-obst_x).^2 + (y-obst_y).^2 <= obst_r.^2;
70  obst(:,[1,ly]) = 1;     % Location of top/bottom boundary
71  bbRegion = find(obst); % Boolean mask for bounce-back cells
72
73  % INITIAL CONDITION: Poiseuille profile at equilibrium
74  L = ly-2; y_phys = y-1.5;
75  ux = 4 * uMax / (L*L) * (y_phys.*L-y_phys.*y_phys);
76  uy = zeros(lx,ly);
77  rho = 1;
78  for i=1:9
79      cu = 3*(cx(i)*ux+cy(i)*uy);
80      fIn(i,:,:) = rho .* t(i) .* ...
81                      ( 1 + cu + 1/2*(cu.*cu) - 3/2*(ux.^2+uy.^2) );
82  end
83
84  % Specify the force
85  gx = zeros(1,lx,ly);
86  gy = zeros(1,lx,ly);
87  gx(:,151:200,:) = -0.01;
88
89  % MAIN LOOP (TIME CYCLES)
90  for cycle = 1:maxT
91
92      % MACROSCOPIC VARIABLES
93      rho = sum(fIn);
94      ux  = reshape ( (cx * reshape(fIn,9,lx*ly)), 1,lx,ly) ./rho;
95      uy  = reshape ( (cy * reshape(fIn,9,lx*ly)), 1,lx,ly) ./rho;
96
97      % MACROSCOPIC (DIRICHLET) BOUNDARY CONDITIONS
98        % Inlet: Poiseuille profile
99      y_phys = col-1.5;
100     %ux(:,in,col) = 4 * uMax / (L*L) * (y_phys.*L-y_phys.*y_phys);
101     ux(:,in,col) = 0.05;
102     uy(:,in,col) = 0;
```

```
103     rho(:,in,col) = 1 ./ (1-ux(:,in,col)) .* ( ...
104         sum(fIn([1,3,5],in,col)) + 2*sum(fIn([4,7,8],in,col)) );
105       % Outlet: Constant pressure
106     rho(:,out,col) = 1;
107     ux(:,out,col) = -1 + 1 ./ (rho(:,out,col)) .* ( ...
108         sum(fIn([1,3,5],out,col)) + 2*sum(fIn([2,6,9],out,col)) );
109     uy(:,out,col)  = 0;
110
111     % MICROSCOPIC BOUNDARY CONDITIONS: INLET (Zou/He BC)
112     fIn(2,in,col) = fIn(4,in,col) + 2/3*rho(:,in,col).*ux(:,in,col);
113     fIn(6,in,col) = fIn(8,in,col) + 1/2*(fIn(5,in,col)-fIn(3,in,col)) ...
114                                   + 1/2*rho(:,in,col).*uy(:,in,col) ...
115                                   + 1/6*rho(:,in,col).*ux(:,in,col);
116     fIn(9,in,col) = fIn(7,in,col) + 1/2*(fIn(3,in,col)-fIn(5,in,col)) ...
117                                   - 1/2*rho(:,in,col).*uy(:,in,col) ...
118                                   + 1/6*rho(:,in,col).*ux(:,in,col);
119
120     % MICROSCOPIC BOUNDARY CONDITIONS: OUTLET (Zou/He BC)
121     fIn(4,out,col) = fIn(2,out,col) - 2/3*rho(:,out,col).*ux(:,out,col);
122     fIn(8,out,col) = fIn(6,out,col) +
    1/2*(fIn(3,out,col)-fIn(5,out,col)) ...
123                                     -
    1/2*rho(:,out,col).*uy(:,out,col) ...
124                                     -
    1/6*rho(:,out,col).*ux(:,out,col);
125     fIn(7,out,col) = fIn(9,out,col) +
    1/2*(fIn(5,out,col)-fIn(3,out,col)) ...
126                                     +
    1/2*rho(:,out,col).*uy(:,out,col) ...
127                                     -
    1/6*rho(:,out,col).*ux(:,out,col);
128
129     % COLLISION STEP
130     for i=1:9
131         cu = 3*(cx(i)*ux+cy(i)*uy);
132         cg = 3*(cx(i)*gx+cy(i)*gy);
133
134         fEq(i,:,:)  = rho .* t(i) .* ...
135                         ( 1 + cu + 1/2*(cu.*cu)  - 3/2*(ux.^2+uy.^2) );
136         % fOut(i,:,:) = fIn(i,:,:) - omega .* (fIn(i,:,:)-fEq(i,:,:));
137         force = rho .* t(i) .* ...
138                         ( cg + (cu.*cg) - 3*(ux.*gx +uy.*gy) );
139
140         fOut(i,:,:) = fIn(i,:,:) + omega*wt .* ( fEq(i,:,:)-fIn(i,:,:) )
    ...
141           + wt*force;
142     end
143
144     % OBSTACLE (BOUNCE-BACK)
145     for i=1:9
146           fOut(i,bbRegion) = fIn(opp(i),bbRegion);
147     end
148
149     % STREAMING STEP
150     for i=1:9
151         fIn(i,:,:) = circshift(fOut(i,:,:), [0,cx(i),cy(i)]);
152     end
```

```
153
154     % VISUALIZATION
155     if (mod(cycle,tPlot)==1)
156         u = reshape(sqrt(ux.^2+uy.^2),lx,ly);
157         u(bbRegion) = nan;
158
159         ux = reshape(ux,lx,ly);
160         uy = reshape(uy,lx,ly);
161         ux(bbRegion) = nan;
162         uy(bbRegion) = nan;
163         P = rho * 1/3;
164         P = reshape(P, lx, ly);
165
166 %         [yP, xP] = meshgrid(1:5:lx,1:5:ly);
167
168
169 %         subplot(1, 3, 1)
170 %         imagesc(u');
171 %         axis equal off;
172 %         subplot(1, 3, 2)
173 %         contour(u')
174 %         hold on
175 %         quiver(ux', uy');
176 %         axis equal off;
177 %         hold offremore
178 %         subplot(1, 3, 3)
179 %         contourf(P')
180 %         axis equal off;
181                 subplot(1, 2, 1)
182         imagesc(u');
183         colorbar
184         axis equal off;
185         subplot(1, 2, 2)
186         contourf(P')
187         colorbar
188         axis equal off;
189 %         drawnow
190         print(sprintf('lbm_matlab_%d',cycle),'-dpng')
191     end
192 end
```

5.2. bladeforce.m

Listing 5.2: bladeforce.m

```
1  %function F = bladeforce( u0, omega, phi )
2  function F = bladeforce()
3  npts = 36;
4  u0 = 10;
5  omega = 2*pi;
6  phi = 2*pi*(0:(npts-1))/npts;
7
8  R = 2;
9  rho = 1.185;
10 L = 0.3;
```

```
11 pitch = 0;
12
13 u = [u0;0];
14
15 r = zeros(length(phi),2);
16 fD = zeros(length(phi),2);
17 fL = zeros(length(phi),2);
18 F = zeros(size(r));
19
20 M = zeros(size(phi));
21
22 for k = 1:length(phi)
23
24   pt = phi(k);
25   cw = cos(pt);
26   sw = sin(pt);
27
28   er = [ -sw; cw ];
29   et = [ cw; sw ];
30   vt = -(R*omega)*et;
31
32   % Compute relative velocity
33   w = u - vt;
34
35   eD = w / norm(w);
36   w2 = w'*w;
37
38   tmp = eD'*er;
39   eL = [ eD(2); -eD(1) ];
40   if tmp > 0
41     % eL = [ -eD(2); eD(1) ];
42     alpha = -acos(eD'*et) - pitch;
43   else
44     alpha = acos(eD'*et) - pitch;
45   end
46
47   C0 = 0.5*rho*w2*L;
48   [cL,cD] = liftanddrag( alpha );
49
50   xv(k) = -R*sw;
51   yv(k) = R*cw;
52   r(k,1) = -R*sw;
53   r(k,2) = R*cw;
54
55   fD(k,1) = cD*C0*eD(1);
56   fD(k,2) = cD*C0*eD(2);
57
58   fL(k,1) = cL*C0*eL(1);
59   fL(k,2) = cL*C0*eL(2);
60
61   M(k) = -R*( fD(k,:)*et + fL(k,:)*et );
62
63   F(k,1) = fD(k,1) + fL(k,1);
64   F(k,2) = fD(k,2) + fL(k,2);
65
66 %   uvD(k) = eD(1);
67 %   vvD(k) = eD(2);
```

```
%    uvL(k) = eL(1);
%    vvL(k) = eL(2);
%
%    fDu(k) = cD*C0*eD(1);
%    fDv(k) = cD*C0*eD(2);
%
%    fLu(k) = cL*C0*eL(1);
%    fLv(k) = cL*C0*eL(2);

end

sc = 0.02;

phif = (0:0.01:2*pi)';
rf = R*[ -sin(phif), cos(phif) ];


figure(1)
quiver(r(:,1),r(:,2),sc*fD(:,1),sc*fD(:,2),0)
axis equal
hold on
quiver(r(:,1),r(:,2),sc*fL(:,1),sc*fL(:,2),0)
plot(rf(:,1),rf(:,2))
hold off

figure(2)
plot(phi*180/pi,M)

figure(3)
quiver(r(:,1),r(:,2),sc*F(:,1),sc*F(:,2),0)
axis equal
hold on
plot(rf(:,1),rf(:,2))
hold off


end

\section{\texttt{liftanddrag.m}}
\begin{lstlisting}[language=matlab, caption={liftanddrag.m}]
function [cL, cD] = liftanddrag( alpha )
%   alpha     CL         CD        CDp        CM      Top_Xtr   Bot_Xtr
%  ------ -------- --------- --------- -------- -------- --------
NACA0012 = [ ...
 -18.750  -1.2412   0.10481   0.10127  -0.0051   1.0000   0.0385
-18.500   -1.2663   0.09658   0.09290  -0.0099   1.0000   0.0389
-18.250   -1.2893   0.08886   0.08503  -0.0144   1.0000   0.0392
-18.000   -1.3103   0.08150   0.07752  -0.0186   1.0000   0.0395
-17.750   -1.3292   0.07464   0.07050  -0.0226   1.0000   0.0398
-17.500   -1.3457   0.06835   0.06405  -0.0263   1.0000   0.0402
-17.250   -1.3593   0.06267   0.05820  -0.0295   1.0000   0.0405
-17.000   -1.3694   0.05770   0.05306  -0.0323   1.0000   0.0408
-16.750   -1.3798   0.05312   0.04836  -0.0342   1.0000   0.0413
-16.500   -1.3871   0.04917   0.04434  -0.0356   1.0000   0.0419
-16.250   -1.3904   0.04576   0.04085  -0.0368   1.0000   0.0424
-16.000   -1.3910   0.04280   0.03781  -0.0376   1.0000   0.0429
-15.750   -1.3897   0.04022   0.03514  -0.0381   1.0000   0.0435
```

```
125 -15.500   -1.3868    0.03795    0.03278   -0.0382    1.0000    0.0442
126 -15.250   -1.3827    0.03594    0.03067   -0.0380    1.0000    0.0449
127 -15.000   -1.3773    0.03417    0.02877   -0.0375    1.0000    0.0456
128 -14.750   -1.3706    0.03262    0.02709   -0.0367    1.0000    0.0463
129 -14.500   -1.3630    0.03123    0.02555   -0.0356    1.0000    0.0469
130 -14.250   -1.3614    0.02935    0.02364   -0.0338    1.0000    0.0479
131 -14.000   -1.3541    0.02810    0.02235   -0.0322    1.0000    0.0487
132 -13.750   -1.3438    0.02712    0.02132   -0.0305    1.0000    0.0496
133 -13.500   -1.3321    0.02628    0.02042   -0.0288    1.0000    0.0506
134 -13.250   -1.3194    0.02552    0.01958   -0.0271    1.0000    0.0517
135 -13.000   -1.3055    0.02484    0.01880   -0.0253    1.0000    0.0527
136 -12.750   -1.2897    0.02415    0.01800   -0.0238    1.0000    0.0536
137 -1.2778    0.02298    0.01683   -0.0219    1.0000    0.0551
138 -1.2605    0.02228    0.01611   -0.0206    1.0000    0.0563
139 -1.2412    0.02171    0.01549   -0.0195    1.0000    0.0577
140 -1.2210    0.02118    0.01490   -0.0185    1.0000    0.0592
141 -1.1996    0.02073    0.01435   -0.0176    1.0000    0.0606
142 -1.1823    0.01983    0.01344   -0.0162    1.0000    0.0624
143 -1.1622    0.01923    0.01283   -0.0151    1.0000    0.0641
144 -1.1403    0.01877    0.01234   -0.0142    1.0000    0.0659
145 -1.1178    0.01837    0.01188   -0.0133    1.0000    0.0678
146 -1.0953    0.01794    0.01138   -0.0124    1.0000    0.0695
147 -1.0758    0.01723    0.01070   -0.0112    1.0000    0.0721
148 -1.0534    0.01682    0.01028   -0.0103    1.0000    0.0742
149 -1.0305    0.01647    0.00989   -0.0094    1.0000    0.0765
150 -1.0068    0.01620    0.00955   -0.0086    1.0000    0.0784
151 -0.9872    0.01554    0.00894   -0.0072    1.0000    0.0817
152 -0.9649    0.01520    0.00860   -0.0061    1.0000    0.0844
153 -0.9423    0.01491    0.00827   -0.0051    1.0000    0.0871
154 -0.9205    0.01455    0.00788   -0.0039    1.0000    0.0898
155 -0.8996    0.01412    0.00750   -0.0026    1.0000    0.0934
156 -0.8772    0.01386    0.00723   -0.0015    1.0000    0.0967
157 -0.8545    0.01365    0.00697   -0.0003    1.0000    0.0995
158 -0.8341    0.01321    0.00657    0.0011    1.0000    0.1037
159 -0.8122    0.01295    0.00634    0.0023    1.0000    0.1077
160 -0.7897    0.01277    0.00612    0.0035    1.0000    0.1113
161 -0.7688    0.01242    0.00581    0.0048    1.0000    0.1157
162 -0.7471    0.01219    0.00561    0.0061    1.0000    0.1204
163 -0.7246    0.01204    0.00543    0.0072    1.0000    0.1246
164 -0.7033    0.01174    0.00518    0.0085    1.0000    0.1299
165 -0.6812    0.01156    0.00503    0.0096    1.0000    0.1353
166 -0.6587    0.01143    0.00488    0.0107    1.0000    0.1401
167 -0.6282    0.01116    0.00469    0.0101    0.9986    0.1475
168 -0.5894    0.01101    0.00454    0.0078    0.9958    0.1547
169 -0.5487    0.01074    0.00436    0.0051    0.9930    0.1643
170 -0.5108    0.01052    0.00417    0.0031    0.9887    0.1729
171 -0.4714    0.01031    0.00402    0.0008    0.9848    0.1830
172 -0.4307    0.01006    0.00386   -0.0019    0.9819    0.1937
173 -0.3931    0.00985    0.00369   -0.0037    0.9765    0.2044
174 -0.3537    0.00962    0.00353   -0.0060    0.9716    0.2166
175 -0.3172    0.00938    0.00337   -0.0076    0.9657    0.2292
176 -0.2843    0.00916    0.00321   -0.0084    0.9565    0.2416
177 -0.2538    0.00898    0.00308   -0.0086    0.9451    0.2541
178 -0.2243    0.00879    0.00295   -0.0085    0.9326    0.2681
179 -0.1964    0.00861    0.00281   -0.0081    0.9173    0.2824
180 -0.1699    0.00843    0.00266   -0.0074    0.8986    0.2968
181 -0.1439    0.00825    0.00251   -0.0066    0.8769    0.3138
```

```
-0.1184   0.00803   0.00235  -0.0058   0.8535   0.3382
-0.0946   0.00766   0.00221  -0.0048   0.8260   0.4076
-0.0719   0.00727   0.00211  -0.0035   0.7958   0.5090
-0.0482   0.00704   0.00206  -0.0023   0.7639   0.5821
-0.0243   0.00692   0.00203  -0.0011   0.7274   0.6395
.0000   0.00688   0.00202   0.0000   0.6873   0.6873
.0243   0.00692   0.00203   0.0011   0.6397   0.7274
.0483   0.00704   0.00206   0.0023   0.5821   0.7638
.0720   0.00727   0.00211   0.0035   0.5089   0.7958
.0946   0.00766   0.00222   0.0047   0.4075   0.8261
.1185   0.00803   0.00235   0.0058   0.3382   0.8534
.1439   0.00825   0.00251   0.0066   0.3138   0.8769
.1699   0.00843   0.00266   0.0074   0.2968   0.8986
.1965   0.00861   0.00281   0.0081   0.2824   0.9174
.2243   0.00879   0.00295   0.0085   0.2681   0.9326
.2538   0.00898   0.00308   0.0086   0.2541   0.9452
.2843   0.00916   0.00321   0.0084   0.2416   0.9565
.3172   0.00938   0.00337   0.0076   0.2292   0.9657
.3537   0.00962   0.00353   0.0060   0.2166   0.9716
.3931   0.00985   0.00369   0.0037   0.2044   0.9765
.4307   0.01006   0.00386   0.0019   0.1937   0.9820
.4714   0.01031   0.00402  -0.0008   0.1829   0.9849
.5107   0.01052   0.00417  -0.0031   0.1729   0.9887
.5488   0.01074   0.00436  -0.0051   0.1643   0.9930
.5894   0.01101   0.00454  -0.0078   0.1547   0.9958
.6282   0.01116   0.00469  -0.0101   0.1475   0.9986
.6585   0.01143   0.00488  -0.0107   0.1401   1.0000
.6811   0.01156   0.00503  -0.0096   0.1353   1.0000
.7032   0.01174   0.00518  -0.0085   0.1299   1.0000
.7244   0.01204   0.00543  -0.0072   0.1246   1.0000
.7469   0.01219   0.00561  -0.0060   0.1204   1.0000
.7687   0.01242   0.00581  -0.0048   0.1157   1.0000
.7896   0.01277   0.00612  -0.0034   0.1113   1.0000
.8120   0.01295   0.00634  -0.0023   0.1077   1.0000
.8339   0.01321   0.00657  -0.0011   0.1037   1.0000
.8543   0.01365   0.00697   0.0004   0.0995   1.0000
.8771   0.01386   0.00722   0.0015   0.0967   1.0000
.8995   0.01412   0.00749   0.0026   0.0934   1.0000
.9204   0.01455   0.00788   0.0039   0.0899   1.0000
.9422   0.01491   0.00827   0.0051   0.0871   1.0000
.9649   0.01520   0.00860   0.0061   0.0844   1.0000
.9872   0.01554   0.00893   0.0072   0.0817   1.0000
.0068   0.01621   0.00955   0.0086   0.0784   1.0000
.0305   0.01647   0.00989   0.0094   0.0765   1.0000
.0535   0.01682   0.01028   0.0103   0.0742   1.0000
.0759   0.01723   0.01070   0.0112   0.0721   1.0000
.0955   0.01794   0.01138   0.0124   0.0695   1.0000
.1180   0.01836   0.01188   0.0133   0.0678   1.0000
.1405   0.01877   0.01234   0.0141   0.0659   1.0000
.1624   0.01923   0.01283   0.0150   0.0641   1.0000
.1826   0.01983   0.01344   0.0161   0.0624   1.0000
.1999   0.02073   0.01435   0.0175   0.0606   1.0000
.2214   0.02118   0.01490   0.0184   0.0592   1.0000
.2416   0.02171   0.01549   0.0194   0.0577   1.0000
.2609   0.02228   0.01611   0.0205   0.0563   1.0000
.2783   0.02298   0.01683   0.0218   0.0550   1.0000
.2902   0.02416   0.01801   0.0237   0.0536   1.0000
```

```
.3062   0.02484   0.01880   0.0252   0.0527   1.0000
.3202   0.02552   0.01958   0.0269   0.0517   1.0000
.3329   0.02628   0.02043   0.0287   0.0506   1.0000
.3447   0.02712   0.02132   0.0304   0.0496   1.0000
.3551   0.02809   0.02235   0.0320   0.0487   1.0000
.3625   0.02936   0.02364   0.0337   0.0478   1.0000
.3638   0.03126   0.02558   0.0354   0.0469   1.0000
.3719   0.03260   0.02707   0.0365   0.0463   1.0000
.3787   0.03414   0.02875   0.0373   0.0456   1.0000
.3842   0.03592   0.03064   0.0378   0.0449   1.0000
.3884   0.03792   0.03275   0.0380   0.0441   1.0000
.3915   0.04018   0.03510   0.0379   0.0435   1.0000
.3929   0.04275   0.03776   0.0374   0.0429   1.0000
.3923   0.04571   0.04080   0.0365   0.0424   1.0000
.3890   0.04913   0.04429   0.0353   0.0418   1.0000
.3817   0.05308   0.04831   0.0340   0.0412   1.0000
.3716   0.05766   0.05301   0.0320   0.0408   1.0000
.3616   0.06263   0.05816   0.0292   0.0405   1.0000
.3480   0.06831   0.06402   0.0260   0.0401   1.0000
.3316   0.07461   0.07047   0.0223   0.0398   1.0000
.3127   0.08146   0.07748   0.0183   0.0395   1.0000
.2918   0.08882   0.08500   0.0140   0.0391   1.0000
.2689   0.09653   0.09286   0.0096   0.0388   1.0000
.2440   0.10474   0.10120   0.0048   0.0385   1.0000 ];

d = 180*alpha/pi;
angle = NACA0012(:,1);
lift = NACA0012(:,2);
drag = NACA0012(:,3);

if d < angle(1)
  d = angle(1);
elseif d > angle(end)
  d = angle(end);
end

cL = interp1(angle,lift,d);
cD = interp1(angle,drag,d);
```

6. Glossar

alcedo In dieser Studienarbeit produzierte Software

BGK Bhatnagar-Gross-Krook Näherung für die Kollision

CFD Computational Fluid Dynamics

gcc GNU Compiler Collection

GPU Graphics Processing Unit, Grafikkarte in einem Computer

GPGPU General-Purpose computing on Graphics Processing Units

NVIDIA Nvidia Corporation, von Kalifornien, USA, produziert GPUs für Gaming und professionelle Anwendungen sowie System On A Chip Units (SoCs).

SDK Software Development Kit

Streaming Multiprocessor Prozessor auf der GPU. Eine Grafikkarte hat mehrere SMs.

Literatur

[1] Jonas Walker Alain Schubiger. ≪Lattice-Boltzmann Codeentwicklung≫. In: *Hochschule für Technik Rapperswil* Projektarbeit II (2016).

[2] N. A. Adams E. Riegel T. Indinger. ≪Implementation of a Lattice-Boltzmann method for numerical fluid mechanics using the nVIDIA CUDA technology≫. In: *Springer-Verlag* CSRD (2009) 23.DOI 10.1007/s00450-009-0087-3 (2009).

[3] Chen Peng. ≪The Lattice Boltzmann Method for Fluid Dynamics: Theory and Applications≫. In: *École Polytechnique Fédérale de Lausanne* Master of Mathematics Thesis (2011).

[4] M. Cathelain S. Rullaud F.Blondel. ≪Actuator-Line Model in a Lattice Boltzmann Framework for Wind Turbine Simulations≫. In: *Jounral of Physics: Conference Series* 1037.022023 (2018). URL: https://doi.org/10.1088/1742-6596/1037/2/022023.

[5] Alain Schubiger. ≪Thermal Simulations at low to modest Reynolds numbers using the lattice Boltzmann method. A comparison to the Finite-Volume Method(CFD)≫. In: *Hochschule für Technik Rapperswil* Master Thesis (2017).

[6] Wikipedia. *Schematische Darstellung eines Zeitschrittes in einem D2Q9-Modell*. URL: https://upload.wikimedia.org/wikipedia/commons/c/cb/Lattice_boltzmann_3steps.svg. (Zugriff: 2018-12-3).