TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Computation in Engineering

Interaktive Strömungssimulation auf Hochleistungsrechnern unter Anwendung der Lattice-Boltzmann Methode

Michael Pfaffinger

Vollständiger Abdruck der von der Fakultät für Bauingenieur- und Vermessungswesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. habil. M. Manhart

Prüfer der Dissertation:

1. Univ.-Prof. Dr.rer.nat. E. Rank
2. Univ.-Prof. Dr.-Ing. G. Hausladen
3. Univ.-Prof. Dr.-Ing. habil. Chr. van Treeck,
   Rheinisch-Westfälische Technische Hochschule Aachen

Die Dissertation wurde am 19.10.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Bauingenieur- und Vermessungswesen am 18.04.2012 angenommen.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit interaktiver Simulation von Raumluftströmungen auf Hochleistungsrechnern. Konkret werden thermische Strömungen untersucht, die als Grundlage für eine Behaglichkeitsanalyse dienen können. Hierbei werden zuerst das Thema Computational Steering und anschließend die Lattice-Boltzmann Methode (LBM) als Basis der Strömungssimulation erörtert.
Einen weiteren Kern der Arbeit bildet die Visualisierung der gewonnenen Daten aus der interaktiven Strömungssimulation. An dieser Stelle wird auf verschiedene Themen, u.a. auch die Nutzung von Virtual-Reality Techniken, eingegangen.
In einem weiteren Schritt wird die für die Simulation notwendige Parallelisierungsstrategie näher untersucht. Hierbei wird auch auf das Kommunikationskonzept eingegangen. Anschließend werden Optimierungsansätze für die Nutzung auf dem Höchstleistungsrechner Bayern II (HLRB II) untersucht und die Leistungsfähigkeit des Simulationscodes mit Hilfe von Benchmark Beispielen in verschiedenen Konfigurationen untersucht. Hierbei wurde auch die Möglichkeit einer hybriden Parallelisierungsstrategie mit OpenMP bewertet.
Einen weiteren Kernbaustein der Arbeit bildet die Validierung des LBM basierten Simulationscodes anhand von Referenzsimulationen mit einer kommerziellen Simulationsumgebung. Dies geschieht an zwei klar definierten Beispielen. Bei dem ersten Beispiel handelt es sich um ein Modell mit freier Konvektion und bei dem zweiten Beispiel wird ein Modell mit gemischter Konvektion betrachtet.
Abschließend werden zwei mögliche industrielle Anwendungsbeispiele gezeigt.

Abstract

This thesis examines the interactive simulation of indoor air flow using high-performance computers. Particular focus is given to thermal fluid flow, which can serve as a basis for thermal comfort analysis. The dissertation begins by addressing the subject of Computational Steering before examining the Lattice-Boltzmann Method (LBM) as the basis for the computational fluid flow simulation.
Another key aspect of this work is the visualization of the data obtained from interactive fluid flow simulation. Here, various issues, including the use of Virtual-Reality techniques, are investigated.
The parallelization strategy necessary for the simulation is discussed and questions concerning the communication concept for the parallelization are evaluated. Different optimization strategies for use with the high-performance computer HLRB II (Höchstleistungsrechner Bayern II) are also examined and the performance of the simulation code tested using benchmark examples in various configurations. The possibility of using hybrid OpenMP parallelization is also addressed.
Another key aspect of this work is the validation of the LBM-based simulation code based on reference simulations with a commercial simulation environment. This is achieved using two clearly defined examples: the first example is a model with free convection and the second example a model with mixed convection.
Finally, two potential industrial application examples are shown.

Vorwort

Diese Arbeit entstand als Teil meiner Arbeit am Lehrstuhl für Computation in Engineering (vormals Bauinformatik) in den Jahren 2005 bis 2011 im Rahmen des Forschungsprojektes *ComfSim* und des IGSSE Projekts *Buildings, users, climate*. Die Arbeit wurde durch ein durch die Siemens AG gewährtes Industriestipendium gefördert.

Ich möchte mich an dieser Stelle bei all jenen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben. Ganz besonderer Dank geht an meinen Doktorvater Herrn Prof. Dr.rer.nat. Ernst Rank, der mich zu jeder Zeit unterstützt hat und somit maßgeblich zum Gelingen beigetragen hat.

Ebenso möchte ich Herrn Prof. Dr.-Ing. Gerhard Hausladen und Herrn Univ.-Prof. Dr.-Ing. habil. Christoph van Treeck für die Übernahme des Zweit- bzw. Drittgutachtens danken. Bei der Siemens AG bedanke ich mich an dieser Stelle für die finanzielle Unterstützung, ohne die diese Arbeit nicht möglich gewesen wäre.

Des Weiteren möchte ich mich bei Herrn Dr. rer.nat. Ralf-Peter Mundani, Herrn Martin Schlaffer und nochmal bei Herrn Univ.-Prof. Dr.-Ing. habil. van Treeck bedanken. Sie standen mir bei fachlichen Fragen immer mit Rat und Tat zur Seite.

Stellvertretend für alle Kollegen am Lehrstuhl möchte ich hier der guten Seele des Lehrstuhls, Frau Hanne Cornils, und meinem Zimmerkollegen Herrn David Franke meinen herzlichen Dank aussprechen. Ich wusste das außerordentlich angenehme Arbeitsklima und die familiäre Atmosphäre am Lehrstuhl ganz besonders zu schätzen.

Meinem Vater danke ich für die fortwährende Unterstützung während meines Studiums und meiner Promotionszeit. Er war jederzeit für mich da.

Zum Schluss möchte ich mich noch bei meiner Frau Marjolaine bedanken. Sie gab mir immer Halt und Unterstützung, auch wenn ich während meiner Promotionszeit wieder einmal gestresst war und nicht genug Zeit für sie hatte.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Ein zunehmender Zeit- und Kostendruck in der Planung und Konstruktion von Gebäuden, aber auch von Fahrzeugen aller Art, Schiffen und Flugzeugen, sowie die Forderung nach nachhaltigen Energiekonzepten vor allem bei Bauwerken machen neue Werkzeuge z.B. für die TGA[1]- oder Energieplanung für den Planer erforderlich.
In der TGA-Planung kommen zur Strömungssimulation bereits seit einiger Zeit Computational Fluid Dynamics (CFD) Programme, d.h. Werkzeuge zur numerischen Strömungssimulation, zur Anwendung. Bei diesen Programmen bleibt jedoch die Erstellung des Modells, die numerische Simulation selbst und die Auswertung der Ergebnisse immer noch eine zeitaufwendige und somit teure Aufgabe.
Bei den am Markt hauptsächlich verbreiteten CFD Werkzeugen handelt es sich um Programme, welche versuchen, die Navier-Stokes Gleichungen direkt mit numerischen Mitteln anzunähern. Der Ansatz, wie er in dieser Arbeit besprochen wird, setzt hierbei auf die Lattice-Boltzmann Methode (siehe 3.2), eine zu herkömmlichen Verfahren alternative Methode, die sich beispielsweise durch die Verwendung von uniformen kartesischen Gittern auszeichnet. Hierdurch ergeben sich insbesondere Vorteile bei der Gittergenerierung, welcher bei dem Streben nach einer interaktiven Simulation, bei der der Nutzer direkt mit der Simulation interagieren kann (siehe Kapitel 2), eine bedeutende Rolle zukommt.
In vielen Anwendungsbereichen, wie z.B. bei der Planung von Belüftungsanlagen in Bürogebäuden, Krankenhäusern, Zügen, Flugzeugkabinen oder Maschinenräumen, ist neben einer CFD Simulation auch die Betrachtung des thermischen Komforts in Innenräumen von wachsender Bedeutung. Dies stellt jedoch immer noch eine komplexe Aufgabe dar. Hierfür sind zwar mittlerweile auch kommerzielle Softwareprodukte verfügbar, beispielsweise THESEUS-FE der P+Z Engineering GmbH [54] oder auch WUFI Plus Therm des Fraunhofer Instituts für Bauphysik, jedoch bieten diese Produkte bisher keine Integration in eine interaktive Entwicklungsumgebung und sind teilweise auch auf den Fahrzeugbau (THESEUS-FE) spezialisiert. Des Weiteren handelt es sich zum Beispiel bei den üblicherweise im Gebäudebereich eingesetzten Systemen um zonale Verfahren. Dies bedeutet, dass das Bauwerk bzw. dessen Luftvolumen in einzelne Zonen unterteilt wird. Für jede Zone wird dann beispielsweise nur eine gemittelte Raumlufttemperatur berechnet, weshalb bei diesem Ansatz keine Informationen über das

[1] Technische Gebäudeausrüstung.

genaue Strömugsverhalten der Luft in einem Raum und damit einhergehend auch keine Informationen über den Einfluss der Strömung auf den thermischen Komfort vorhanden sind.
Zur Verbesserung dieser Situation wird in dieser Arbeit ein Projekt beschrieben, welches die einzelnen Bestandteile in einer Computational Steering Umgebung (CSE[2]) zu vereinen versucht. Die CSE `iFluids` besteht aus einer thermischen Strömungssimulation, welche speziell auf die Simulation von Innenraumluftströmungen ausgerichtet ist.
Es soll versucht werden, dem Planer in frühen Planungsphasen ein Werkzeug an die Hand zu geben, um damit einen Entwurf zu untersuchen und zu optimieren. Er soll die Möglichkeit erhalten, interaktiv eine Strömungssimulation durchführen zu können. Die Ergebnisse werden direkt zusammen mit dem zu Grunde liegenden Geometriemodell visualisiert.
Zur Beschleunigung des Designprozesses ist es außerdem wünschenswert, zur Laufzeit der Simulation Änderungen an den Parametern, den Randbedingungen und sogar an der Geometrie des Simulationsmodells durchführen zu können und die Auswirkungen auf die (Komfort)-Bedingungen in dem simulierten Raum direkt zu sehen.
Die Fokussierung auf die frühe Planungsphase wurde gewählt, da in diesem Stadium das größte Einsparpotential durch eine intelligente Planung gegeben ist. Der qualitative Verlauf der Einsparmöglichkeiten bei der Planung bzw. beim Bau eines typischen Gebäudes ist in Abbildung 1.1 dargestellt.

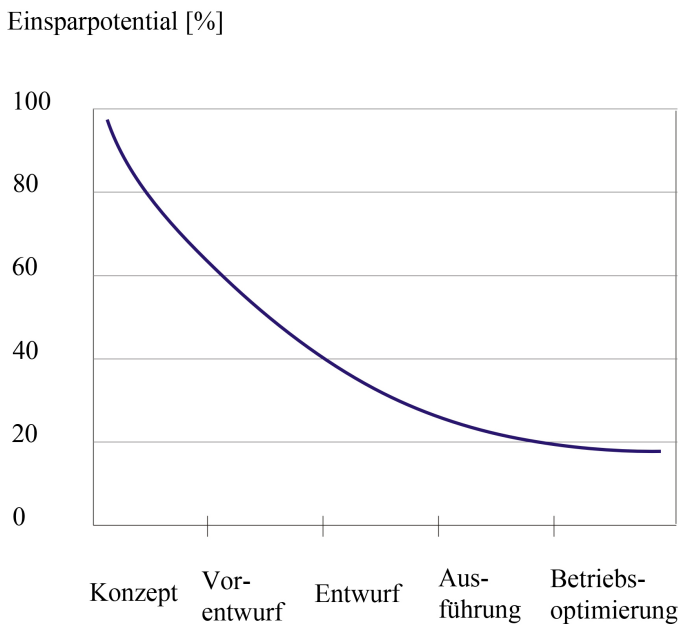


Abbildung 1.1: Einsparpotential während der Planung bzw. dem Bau eines Gebäudes (nach [23]).

Die hier vorgestellten Modelle und Simulationen werden dreidimensional betrachtet, weshalb eine entsprechende Visualisierungstechnik für ein intuitives Arbeiten wünschenswert ist. Entsprechende Ansätze werden in Kapitel 5 untersucht. Ziel muss es sein, dem Benutzer eine einfach zu bedienende und flexible Benutzeroberfläche zur Verfügung zu stellen.
Strömungssimulationen, v.a. von größeren Gebieten und mit feineren, für physikalisch realistische Analysen erforderlichen Auflösungen, benötigen viel Rechenleistung und Arbeitsspeicher.

[2]Computational Steering Environment.

Dies trifft vor allem dann zu, wenn man sich im Kontext einer interaktiven Strömungssimulation, wie sie in `iFluids` umgesetzt ist, bewegt. Hier ist es nicht praktikabel, auf ein Simulationsergebnis mehrere Stunden oder sogar Tage zu warten. Techniken des wissenschaftlichen Höchstleistungsrechnens gewinnen somit für diese Anwendungsfelder immens an Bedeutung, weshalb sie ebenfalls in einem Teil dieser Arbeit betrachtet werden.
Hinsichtlich der für den Anwender verfügbaren Rechenleistung hat sich in den letzten Jahren viel getan. Somit sind Anwendungen, die vor wenigen Jahren noch mangels Rechenleistung unmöglich waren, jetzt auch im normalen Planungsalltag zunehmend möglich. Durch die Entwicklung hin zu Multi- und Many-Core Prozessoren wird neben der Parallelisierung mittels MPI (siehe Abschnitt 6.4.1) auch die Parallelisierung mittels OpenMP (siehe Abschnitt 6.4.2) bzw. einer hybriden Parallelisierungslösung zunehmend interessant und notwendig; dies gilt auch für den Einsatz auf hochperformanten Workstations. Bei modernen Multikern-Prozessoren wird die Leistung der einzelnen Kerne durch eine Senkung des Prozessortaktes reduziert. Die dadurch frei werdenden Energieressourcen werden zum Betrieb von weiteren Kernen genutzt. Da jedoch die Halbierung der Stromaufnahme eines Prozessorkerns nur zu einer Leistungseinbuße um ein Viertel führt, ist beispielsweise die Gesamtleistung eines Zweikernprozessor bei gleicher Stromaufnahme um etwa das $1,5$-fache höher als nur bei einem Kern ähnlicher Bauart. Es ist jedoch zu beachten, dass zur Nutzung der gesamten Leistung eines Mehrkernprozessors auch die verwendeten Programme entsprechend angepasst werden müssen, um tatsächlich alle Kerne zu nutzen.
Auch bei der Leistungsfähigkeit der Visualisierungshardware wurden große Fortschritte gemacht, sodass die Nutzung von Virtual Reality Techniken nicht mehr nur in Rechenzentren, sondern auch bei Anwendern mit kleinem Budget möglich wird.
Durch die Kombination von interaktiven Planungswerkzeugen mit entsprechender Rechenleistung, ggf. auch als on-demand Leistung in einem Rechenzentrum mit zusätzlicher remote Visualisierung, und Virtual Reality Umgebungen ergeben sich völlig neue Möglichkeiten für den Planungsablauf. Die einzelnen Komponenten, die für eine solche Anwendung in der Strömungssimulation von Innenraumluftströmungen nötig sind, werden in dieser Arbeit beleuchtet.

1.2 Aufbau der Arbeit

Die hier vorgestellte Arbeit zeigt diverse Teilaspekte von Innenraumluftströmungen, deren Simulation und Auswertung. Im Anschluss an diese Einführung wird in Kapitel 2 das Thema Computational Steering betrachtet. Hier wird von der Definition ausgegangen, die geschichtliche Entwicklung betrachtet und auf verwandte Projekte eingegangen. Abschließend wird das Computational Steering in `iFluids` vorgestellt.
Das darauf folgende Kapitel 3 widmet sich den methodischen Grundlagen der numerischen Simulation innerhalb der Computational Steering Umgebung. Es wird die Lattice-Boltzmann Methode (LBM) als Alternative zu Navier-Stokes basierten Strömungssimulationen vorgestellt. Da zur Abbildung von thermischen Problemen in dieser Simulationsumgebung ein hybrider Ansatz gewählt wurde, wird auch auf die Methode der Finiten-Differenzen, welche zur Lösung der Advektions/Diffusions-Gleichung verwendet wird, kurz eingegangen. Abschließend folgt in diesem Kapitel noch ein Exkurs zu den verschiedenen Rechengittern in der CFD und insbesondere in `iFluids`.

Für die hier vorgestellte Simulationsumgebung wurde ein parametrisches, numerisches Manikinmodell entwickelt und integriert. Dieses stellt die Schnittstelle zur Anbindung an Methoden für Komfortbewertung in Innenräumen dar und wird in Kapitel 4 erläutert.
Ein Kernbaustein dieser Arbeit, die Visualisierung in der interaktiven Strömungssimulation, wird in Kapitel 5 untersucht. Auch hier wird mit einem Überblick über das Thema begonnen, wobei auch auf die speziellen Anforderungen und Virtual-Reality (VR) Umgebungen eingegangen wird. Anschließend wird die Umsetzung in einer Forschungsumgebung, d.h. in `iFluids`, vorgestellt. Darauf folgt die Darstellung einer möglichen Umsetzung in industriellem Umfeld und die Evaluierung zusätzlicher Nutzungsmöglichkeiten des implementierten VR-Systems für Fremddaten.
Da es sich bei numerischen Strömungssimulationen um sehr rechenzeitintensive Anwendungen handelt und gerade hinsichtlich einer interaktiven Simulation hohe Anforderungen an die Leistungsfähigkeit gestellt werden, ist die Betrachtung von Möglichkeiten zur Parallelisierung des Programms unumgänglich (siehe Kapitel 6).
Diese Arbeit entstand im Kontext von zwei Forschungsprojekten, wobei bei dem Projekt ComfSim[3] [66] eine enge Partnerschaft mit dem Leibniz Rechenzentrum (LRZ) in Garching bei München bestand und somit eine Fokussierung auf den dort installierten, leistungsstarken Bundeshöchstleistungsrechner HLRB II gegeben war. Das Kapitel 7 stellt den Höchstleistungsrechner vor und beschreibt die Arbeiten zur Leistungsoptimierung der Simulationsumgebung im Allgemeinen und insbesondere für diese Hardware. Hierbei wurden entsprechende Benchmark Rechnungen durchgeführt und ausgewertet.
Im Kapitel 8 werden zwei Referenzsimulationen vorgestellt, mit deren Hilfe die Simulationsumgebung `iFluids` evaluiert worden ist. Bei dem ersten Modell handelt es sich um einen beheizten horizontalen Zylinder und bei dem zweiten Modell um die Modellierung des Testraums aus einem IGSSE[4] Projekt. Bei beiden Modellen wurde das Problem zusätzlich zu Vergleichszwecken in einem validierten, kommerziellen CFD Programm (Ansys CFX) modelliert und berechnet.
Anschließend werden in Kapitel 9 noch zwei industrielle Anwendungsbeispiele für eine Innenraumluftströmungssimulation gezeigt.
Den Abschluss bildet eine Zusammenfassung der gewonnenen Ergebnisse und ein Ausblick (siehe Kapitel 10).

[3] ComfSim: Computergestützte Strömungssimulation und Komfortanalyse in Innenräumen (ein durch die Bayerische Forschungsstiftung gefördertes Projekt).
[4] International Graduate School for Science and Engineering der Technischen Universität München.

Kapitel 2

Computational Steering in der CFD

Dieses Kapitel betrachtet das Thema Computational Steering. Hier wird von der Definition derselben ausgegangen, die geschichtliche Entwicklung betrachtet und auf verwandte Projekte eingegangen. Abschließend wird das Computational Steering in `iFluids` vorgestellt.

2.1 Definition

Eine klassische numerische Simulation in der Struktur- oder Strömungsmechanik besteht aus mehreren Schritten. Am Anfang steht die Geometriemodellierung und die Übernahme derselben in das gewählte Simulationsprogramm. Hieran schließt sich die Netzgenerierung an, welche sich vor allem für komplexe Probleme sehr aufwendig gestalten kann und einen signifikanten Anteil am Gesamtaufwand der Simulation hat. Im Zuge der Netzgenerierung werden üblicherweise auch die Randbedingungen des Modells, etwaige Ausgangsbedingungen und die Simulationsparameter festgelegt. Die vorgenannten Arbeitsschritte werden oft unter dem Namen Preprocessing zusammengefasst.
Darauf folgt die eigentliche Berechnung. Diese kann, je nach Umfang, Komplexität und verfügbarer Rechenleistung, durchaus mehrere Tage oder Wochen in Anspruch nehmen. Gerade bei Simulationen von dreidimensionalen Problemen, beispielsweise in der CFD, nimmt die benötigte Rechenzeit bei einer höheren Genauigkeit, d.h. bei feineren Gittern bzw. Netzen, sehr rasch zu. Eine Halbierung der Gitterweite führt beispielsweise zu einem achtfachen Aufwand.
An den Berechnungslauf schließt sich nun das sogenannte Postprocessing an. Hierbei werden die gewonnenen Datenmengen gesichtet und ausgewertet. Da die Datenmenge durchaus mehrere Gigabyte (GB) oder Terabyte (TB) betragen kann, muss auch hierfür entsprechende Rechenleistung und Arbeitszeit aufgewendet werden.
Sollten durch die Auswertung der Simulationsergebnisse Modifikationen am ursprünglichen Design des simulierten Problems nötig werden, gelangt man zu einem iterativen Prozess, welcher wieder im Preprocessing beginnt. Dieser Ablauf ist in Abbildung 2.1 dargestellt.
Johnson und Parker [27] definieren nun Computational Steering als die „capacity to control all aspects of the computational science and engineering pipeline“. Dies bedeutet, dass durch eine Computational Steering Umgebung die oben genannten Arbeitsschritte zusammengefasst werden und der Anwender die Möglichkeit zur direkten Interaktion mit der Simulation erhält. Hierdurch können notwendige Modifikationen am Modell schon während der noch laufenden Simulation erkannt und direkt mit sofortiger Wirkung auf die laufende Rechnung umgesetzt

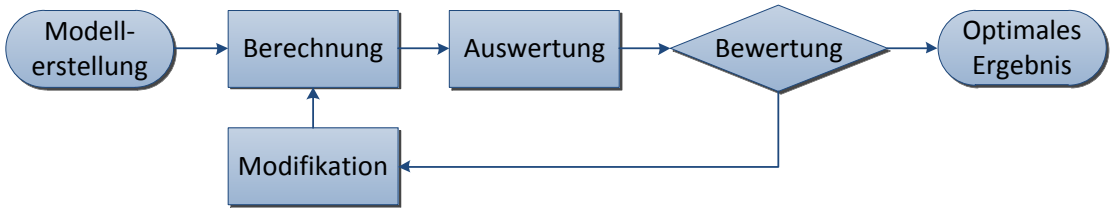


Abbildung 2.1: Ablauf einer numerischen Simulation.

werden. Dies bietet dem Ingenieur eine effiziente Möglichkeit, den Designprozess zu beschleunigen.

2.2 Geschichte des Computational Steering

Die ersten veröffentlichten Ideen zum Computational Steering tauchten bereits in den späten 80er Jahren des 20. Jahrhunderts auf ([28]). Beispielsweise stellt McCormick et al. 1987 fest, dass sich Ingenieure die Möglichkeit wünschen, mit Simulationen in Echtzeit zu interagieren ([40]).
Der Begriff *Computational Steering* wurde dann von van Liere et al. [56] eingeführt. In den Anfängen wurde die Interaktion mit den Simulationsdaten nur auf die Postprocessingschritte beschränkt, das heißt, man versuchte nur durch Interaktion mit der Visualisierung (beispielsweise durch die Anwendung von verschiedenen Filtern, etc.) einen besseren Einblick in die generierten Daten zu bekommen [56]. Erst durch die zunehmende verfügbare Rechenleistung und Netzwerkbandbreite konnten Ansätze verwirklicht werden, die eine Interaktion mit der gesamten oben beschriebenen Simulationskette ermöglichen. Computational Steering hat somit oft eine enge Verbindung zum Höchstleistungsrechnen (HPC[1]) und schneller Netzwerktechnologie, welche niedrige Latenzzeiten ermöglicht. Dies trifft vor allem dann zu, wenn mit großen Datenmengen, wie sie unter anderem in der Strömungssimulation vorkommen, gearbeitet wird.
Zusammen mit van Liere prägten auch Mulder und van Wijk [41] ganz entscheidend die frühe Entwicklung des Computational Steering. Hierbei wurde bereits eine Benutzeroberfläche entwickelt, welche sowohl zur Visualisierung der Simulationsdaten als auch zur Modifikation von Steering Parametern genutzt wurde. Dabei wurde ein Konzept mit parametrisierten Geometrieobjekten für verschiedene, einfache Simulationen umgesetzt.

2.3 Verwandte Projekte

Nachfolgend sollen in der Literatur beschriebene Projekte im Bereich des Computational Steering vorgestellt werden. Die meisten haben jedoch eine wesentliche Einschränkung verglichen mit dem in dieser Arbeit vorgestellten Ansatz; sie ermöglichen zwar eine Interaktion mit der Simulation, jedoch erlauben sie dabei nur die Veränderung einzelner Parameter.

[1] High performance computing.

gViz Projekt

Bei der gViz-Bibliohek handelt es sich um eine Middleware, die es ihren Anwendern erlaubt Daten in einem Postprocessing-Schritt oder direkt während der laufenden Simulation zu visualisieren [8]. Bei beiden Anwendungsszenarien werden Grid-Computing Ressourcen und kollaborative Techniken unterstützt. Zusätzlich zur Visualisierung besteht die Möglichkeit, zur Laufzeit Parameter an den zu Grunde liegenden Simulationskernel zu senden, soweit dieser dies unterstützt.

RealityGrid

Hierbei ([9], [47]) handelt es sich um eine Initiative, welche es sich zum Ziel gesetzt hat, durch einen Steering Ansatz die Entwicklung neuer Materialien sowie das Verständnis von komplexen physikalischen Systemen zu vereinfachen. Bei diesem Projekt ist eine Applikation typischerweise in einen Clientteil, einen Simulationsteil und die Visualisierung unterteilt, welche mittels der Steering Bibliothek miteinander kommunizieren. Die Bibliothek wurde so entworfen, dass der Aufwand, eine vorhandene Applikation „steerable“ zu machen, möglichst gering ist. Eine Grundvoraussetzung, um eine Applikation über diesen Ansatz steeringfähig zu machen, ist die Einführung von Check- und Stopp-Punkten. An diesen Punkten können geänderte Parameter übernommen und die Simulation entsprechend neu gestartet werden. RealityGrid erlaubt sowohl das reine online Monitoring einer laufenden Simulation als auch die Modifikation von bereits initial definierten Parametern. So wurde beispielsweise basierend auf der RealityGrid-Bibliothek ein Lattice-Boltzmann-Strömungssimulator mit Steeringfunktionalität entwickelt [12], welcher die Modifikation von gewissen Parametern (u.a. Relaxationszeit und Dichte des Fluids) zur Laufzeit der Simulation erlaubt. Eine Manipulation der Geometrie wie in der hier vorgestellten Arbeit ist jedoch nicht möglich.

CAVEstudy

Eine weitere Computational Steering Bibliothek ist als Teil des CAVEstudy Projekts entwickelt worden [48]. Hierbei handelt es sich um ein speziell auf den Einsatz in Virtual Reality Umgebungen ausgerichtetes System. Besonders fällt hier auf, dass im Gegensatz zu den bereits vorgestellten Ansätzen hier der ursprüngliche Simulationscode nicht angepasst werden muss. Die Kommunikation mit der Simulation erfolgt über definierbare Eingabe- und Ausgabe-Dateien.

CUMULVS

Auch bei CUMULVS (**C**ollaborative **U**ser **M**igration **U**ser **L**ibrary for **V**isualisation and **S**teering) handelt es sich um eine Middleware, die für Computational Steering Anwendungen eingesetzt werden kann [31]. Sie ermöglicht die dynamische Anbindung von Betrachtern an eine laufende Simulation, um vorläufige Daten zu extrahieren und zu sammeln. Hierbei spielt es keine Rolle, ob es sich bei der Simulation um ein serielles oder paralleles Problem handelt. Die Daten können anschließend in einer Vielzahl von Visualisierungsapplikationen (z.B. VTK oder AVS) mittels einer zur Verfügung gestellten Bibliothek dargestellt, analysiert oder animiert werden. Basierend auf der Auswertung der Zwischenergebnisse kann der Anwender verschiedene Parameter der Simulation zur Laufzeit verändern. Allerdings beschränkt sich

dies wieder auf einfache Parameter des Algorithmus oder des Modells und ermöglicht nicht die Modifikation von beispielsweise Geometrieobjekten oder Netzen. Ein Vorteil von CUMULVS besteht in der Fähigkeit, die Datenschnittstellen auch zur Kommunikation von gekoppelten Simulationen oder zum Erstellen von Checkpoints nutzen zu können. Mittels dieser Checkpoints kann eine Simulation nach einem Hardware- oder Softwarefehler automatisch von einem Checkpoint aus fortgesetzt werden.

COVISE

Der Name leitet sich von **CO**llaborative **VI**sualisation and **S**imulation **E**nvironment ab und wurde am Höchstleistungsrechenzentrum der Universität Stuttgart (HLRS) entwickelt. Es handelt sich hierbei um eine Softwareumgebung, welche Berechnung, Postprocessing und Visualisierungsfunktionen nahtlos vereint. In dieser Softwareumgebung wird eine Applikation in verschiedene Arbeitsschritte unterteilt, welche durch COVISE Module repräsentiert werden. Diese Module lassen sich auch in einer heterogenen Hardwareumgebung auf verschiedene Maschinen verteilen. Besonderes Augenmerk wurde vom HLRS auf die Nutzung auf Höchstleistungsrechnern und Virtual Reality Umgebungen für die Visualisierung gelegt [26].

EPSN

Bei dem französischen Projekt EPSN handelt es sich ebenfalls um eine Computational Steering Umgebung für parallele und verteilte numerische Simulationen. Auch hier sollen die Möglichkeiten einer VR Visualisierung mit der Leistungsfähigkeit von Hochleistungssimulationen kombiniert werden. Es besteht die Möglichkeit, dass sich n Visualisierungsprozesse an m Simulationsprozesse ankoppeln, es handelt sich also um ein sogenanntes *m-by-n* (auch *MxN*) Computational Steering. Bei diesem Projekt wurde zudem besonderer Wert auf die Einhaltung der zeitlichen Kohärenz bei der Manipulation von Daten der Simulation durch den Benutzer gelegt. Hierfür wurde ein sogenanntes *Hierarchical Task Model* (HTM), welches den Steuerungsverlauf der Simulation wiedergeben soll, eingeführt.

pV3

pV3 (parallel Visual3) wurde hauptsächlich für CFD Anwendungen am MIT[2] entwickelt. Bei dem System handelt es sich ebenfalls um eine Client-Server-Architektur, welche mit bestehenden Simulationscodes zusammenarbeitet. Bei diesem sehr frühen Ansatz beschränkt man sich vornehmlich auf ein Online-Monitoring. Hierbei ist es möglich, den Visualisierungsclient dynamisch ohne Beeinflussung der Simulation an diese anzukoppeln und wieder zu trennen. Ein Nachteil von pv3 ist, dass nur eine vorkompilierte Version des Servers und kein offener Quellcode verfügbar ist [24].

VISiT

VISiT (Virtual Intuitive Simulation Testbed) basiert auf der bereits beschriebenen CoVISE-Umgebung und stellt eine Steering Umgebung dar, welche der in dieser Arbeit beschriebenen

[2] Massachusetts Institute of Technology, USA

Auffassung des Computational Steering sehr nahe kommt. Sie integriert kommerzielle Applikationen zur Gittergenerierung und Strömungssimulation mit einer VR-Schnittstelle [30]. Hierbei kann der Anwender neben den Randbedingungen auch Geometrieobjekte innerhalb der Strömung manipulieren. In VISit kommt jedoch im Gegensatz zum Lattice-Boltzmann-Rechenkern in dieser Arbeit ein Navier-Stokes-basierter Strömungslöser zur Anwendung.

ViSiCADE

Das Ziel des europäischen ViSiCADE (**Vi**rtual **Si**mulation Environment for a Seamless Integration of **CAD**/CA**E** into VR) Projekts ist, ein intuitives Simulations Framework zu schaffen, bei dem CAD (Computer Aided Design) und CAE (Computer Aided Engineering) in eine VR-Umgebung intergriert werden mit dem Zweck, den Zeitbedarf für die Modellierung und Auswertung von Simulationen zu reduzieren [37]. Das System stellt dem Ingenieur eine VR-Umgebung zur Verfügung, welche es ihm ermöglicht, ein CAD Modell zu importieren, davon automatisch und interaktiv ein Analysemodell abzuleiten und dieses zu modifizieren. Dies beinhaltet eine automatische Netzgenerierung (mit Methoden zur interaktiven Verfeinerung) sowie Steering Möglichkeiten während der Simulation. Ergänzt wird dies durch speziell entwickelte Visualisierungstechniken, die den Anwender beim Design/Simulations-Zyklus unterstützen. Das System unterstützt eine Arbeitsweise, die eine interaktive Überarbeitung eines bestehenden Modells und einen Neustart der Simulation mit modifizierten Parametern erlaubt.

Interaktive Strömungssimulationen

Einen frühen Ansatz zum Computational Steering in der (Raum)Luftströmungssimulation stellte Kühner 2003 mit VFReal vor [29]. Hierbei handelt es sich um einen Vorläufer der in dieser Arbeit vorgestellten CSE. Bei VFReal handelt es sich um eine monolithische Applikation, welche einen Lattice-Boltzmann basierten Simulationskern mit einer Visualisierungsumgebung koppelt. Auch hier ist es bereits möglich, diverse Geometrieobjekte interaktiv zur Laufzeit der Simulation als Hindernisse im Strömungsgebiet zu platzieren, jedoch besteht die Beschränkung auf eine Kombination von parametrisierten geometrischen Grundkörpern.
In der Arbeit von Wenisch [68, 71] wurde der Ansatz von [29] dann u.a. um die Möglichkeit der vollautomatischen Gittergenerierung erweitert, wodurch die Platzierung von beliebigen facettierten Geometrieobjekten ermöglicht wurde. Ferner wurde von Borrmann [4, 5] dieses Konzept zu einer kooperativen Version des Computational Steering erweitert, welches mehrere Visualisierungs Clients und eigenständige Simulationsprozesse erlaubt.
Die Arbeit wurde anschließend als Teil des ComfSim Projekts von van Treeck et al. [66] weitergeführt.

2.4 Computational Steering in iFluids

Beschreibung von iFluids

Bei der am Lehrstuhl für Computation in Engineering entwickelten interaktiven Software zur Strömungssimulation *iFluids* handelt es sich um eine echtzeitfähige, interaktive 3D-Simulationsumgebung (Computational Steering Environment, kurz CSE), welche eine unmittelbare Auswertung von Ergebnissen unter Verwendung von Virtual-Reality (VR) Techniken (siehe

Kapitel 5) zur Laufzeit erlaubt. Das Strömungssimulationstool basiert im Gegensatz zu anderen gängigen CFD Programmen nicht auf dem Lösen der Navier-Stokes Gleichungen, sondern auf der Lattice-Boltzmann Methode (LBM) (siehe Abschnitt 3.2). Des Weiteren wurde bei der Entwicklung besonderer Wert auf eine modulare Architektur gelegt, die es erlaubt, verschiedene Simulationskerne und Visualisierungs-Frontends zu kombinieren.

Die Applikation setzt sich in der einfachsten Version aus zwei Teilen, dem Simulationskernel und dem Visualisierungsinterface, zusammen, die durch eine MPI-Schnittstelle (Message Passing Interface) verbunden sind. Diese Schnittstelle wurden von Wenisch [69, 71, 68] angelegt und dann im Kontext dieser Arbeit erweitert, um sie für eine thermische Strömungssimulation (siehe van Treeck [65]) zu nutzen und ein numerisches Dummy-Modell integrieren zu können. Sie wird dazu genutzt, um Daten in beide Richtungen zwischen den beiden Programmteilen auszutauschen. Dabei ist es möglich, dass diese Bestandteile auf völlig unterschiedlichen Rechnern, welche über ein Computernetzwerk verbunden sind, ausgeführt werden. Ergänzend ist auch eine kooperative Version, welche auf der Arbeit von Borrmann [6] basiert, verfügbar, die das Arbeiten von mehreren Benutzer an verschiedenen Standorten am selben Modell erlaubt.

Die Visualisierungsapplikation lässt sich auf unterschiedlichen Plattformen, beispielsweise einer Desktop Workstation oder auch einer Virtual Reality Umgebung, betreiben. Diese Anwendung ermöglicht dem Benutzer einerseits die Darstellung und Auswertung der durch die Simulation gewonnenen Daten, andererseits dient sie auch zum Laden und Verändern der Geometrieobjekte und gestattet das Definieren und Modifizieren der Randbedingungen der Simulation. Sämtliche Modifikationen an Geometrie und Randbedingungen lassen sich interaktiv zur Laufzeit der Simulation vornehmen. Das System akzeptiert industrieübliche STL[3]-Dateien als Eingangsdatenformat für die Geometrieobjekte. Bei der Visualisierung von Simulationsdaten stehen dem Anwender verschiedenste Techniken zur Verfügung. In dieser Arbeit wurde das System dahingehend erweitert, dass nun Oberflächentemperaturen und Behaglichkeitsdaten, wie sie z.B. in [61] gewonnen werden, auf der Ausgangsgeometrie dargestellt werden können - gemeinsam mit den Strömungsdaten.

Die Simulationsapplikation ist die zweite Kernkomponente des Systems. Sie bezieht sämtliche Informationen vom Visualisierungsinterface und liefert die Ergebnisdaten auch an dieses zurück. Hierbei handelt es sich um eine transiente Simulation und Änderungen im Strömungsfeld werden direkt und unverzüglich an die Visualisierung übertragen. Die Simulationsapplikation untergliedert sich in verschiedene Komponenten: Das Kommunikationsinterface, den Gittergenerator, den parallelisierten Rechenkernel und die Schnittstellen zur Strahlungs- und Komfortberechnung. Durch die MPI-basierte Parallelisierung des Rechenkernels lässt sich die Applikation auf massiv parallelen Höchstleistungsrechnern, wie beispielsweise dem HLRB II, nutzen.

Dem Gittergenerator, entwickelt von Wenisch [71], kommt im Kontext der CSE eine besondere Bedeutung zu. Es handelt sich hierbei um einen Oktalbaum-basierten Algorithmus, welcher eine extrem hohe Leistung zeigt und eine vollautomatische Generierung des kartesischen Rechengitters in Sekundenbruchteilen ermöglicht (siehe 3.6.3). Erst hierdurch wird die vollständige Interaktion des Benutzers mit der Simulation hinsichtlich Geometrie und Randbedingungen möglich, da Modifikationen in der Regel eine Neugenerierung des Gitters erforderlich machen.

[3]Hierbei handelt es sich um das Stereolithography Dateiformat, bei welchem dreidimensionalen Flächen durch Dreiecksfacetten beschrieben werden.

Interaktives Setzen von Randbedingungen

Wie in Abbildung 2.2 dargestellt, gestattet die CSE ferner das interaktive Setzen bzw. Modifizieren von Randbedingungen. Hierfür kann ein Benutzer zur Laufzeit mit einzelnen Geometrieobjekten interagieren.
In der Berechnung stehen verschiedene Randbedingungen zur Auswahl, welche alle über die grafische Oberfläche angewählt werden können. Jedes Geometrieobjekt, welches keine andere Randbedingung zugewiesen bekommen hat, wird automatisch mit dem Typ *Wall* versehen. Dieser bezeichnet in der Simulation eine *bounce back* Randbedingung. In der Strömungsmechanik werden an Berandungen bzw. Wänden, deshalb *Wall*, Haftrandbedingungen (auch *no-slip* Randbedingungen genannt) definiert. Die Geschwindigkeit des Fluids an der Berandung ist hier durch die Geschwindigkeit dieser bestimmt. Dies bedeutet beispielsweise, dass bei einer ruhenden Berandung die Geschwindigkeit des Fluid direkt an der Wand null ist. Als weitere Randbedingungen für Berandungen der Fluiddomäne sind *slip* Randbedingungen für die drei Raumrichtungen u, v und w vorgesehen. Bei diesen Randbedingungen sind wandparallele Geschwindigkeiten erlaubt, d.h. es wird eine reibungsfreie Wand modelliert. Für die Geschwindigkeitsrandbedingung setzt der Anwender die u-, v- und w-Geschwindigkeitskomponenten einer Strömung direkt. Neben einer Einflussrandbedingung mit vorgegebener Geschwindigkeit lässt sich eine Druckrandbedingung definieren. Weiter kann für thermische Simulationen entweder eine Temperatur oder ein Temperaturgradient definiert werden. Die Auswahl *Fluid* erlaubt es dem Nutzer, temporär Objekte für den Simulationskern auszublenden, ohne sie aus der CSE löschen zu müssen.
Eine weitere Auswahlmöglichkeit im Dialogfeld für Randbedingungen ist die *Hull* Option. Diese beeinflusst jedoch nicht die Simulation, sondern hat nur eine Auswirkung auf die Visualisierung von Geometrieobjekten. Konkret bewirkt diese Option, dass Facetten eines Objekts, für welches diese Option aktiviert wurde, automatisch ausgeblendet werden, wenn die Facettennormale von der virtuellen Kameraposition weg zeigt. Dies bewirkt zum Beispiel bei Wänden, dass sie automatisch ausgeblendet werden und der Benutzer ins Innere eines Raumes blicken kann. Dreht der Anwender die Szene, so wird das Objekt wieder eingeblendet und ggf. ein anderer Teil ausgeblendet.

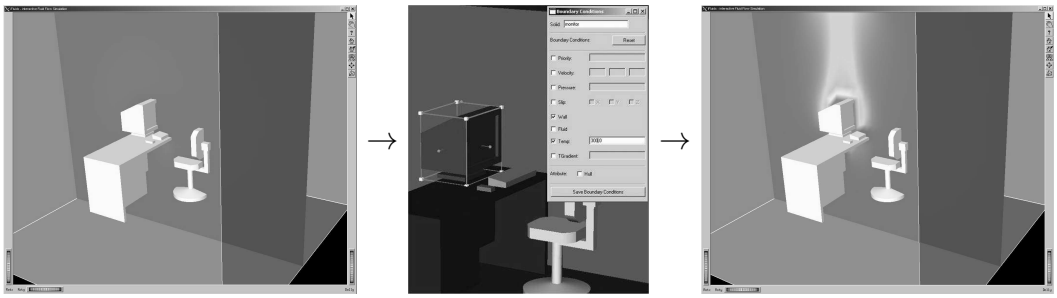


Abbildung 2.2: Interaktives Setzen von Randbedingungen zur Laufzeit. Beispiel: Setzen einer Temperatur-Randbedingung bei einem Bildschirm. Rechts: Auftriebsströmung (aus [63])

Der Benutzer selektiert mit dem Eingabegerät, beispielsweise mit einer Maus oder SpaceMouse, ein Geometrieobjekt im Darstellungsbereich der Visualisierungsapplikation und wechselt anschließend in den Subobjektmodus. In diesem Modus sind die einem Geometrieobjekt zugeordneten Subobjekte einzeln auswählbar. Zur besseren Unterscheidbarkeit der Subobjek-

te des selektierten Geometrieobjekts werden diese in verschiedenen Farben dargestellt. Jedes Subobjekt kann nun separat selektiert und das zugehörige Fenster der Benutzeroberfläche zur Bearbeitung der Randbedingungen geöffnet werden. Dies geschieht durch einen Doppelklick mit dem Eingabegerät bzw. die Nutzung eines Tastaturkurzbefehls. Der Randbedingungsdialog bietet dem Benutzer sämtliche verfügbaren Randbedingungen zur Auswahl an. Nach Bestätigung der Auswahl wird das Dialogfenster geschlossen und die Änderungen im Datensatz des Geometrieobjekts bzw. seines Subobjekts aktualisiert. Die Änderungen werden dabei auch unmittelbar dem Rechenkern kommuniziert.

Integration des parametrischen Modells

Ein parametrisches Manikinmodell (siehe Abschnitt 4 und [61], [77]) wurde im Rahmen dieser Arbeit in die Visualisierungsapplikation integriert und mit den vorhandenen Geometrieschnittstellen verbunden. Hierbei werden die facettierten Oberflächen der Körperteile des Manikins an die CSE übergeben. Dadurch lässt sich das Manikin in verschiedenen Detaillierungsstufen und Oberflächennetzauflösungen interaktiv in eine laufende Simulation laden. Hierzu wurde eine Eingabemaske integriert. Über diese Maske lässt sich auch die Körperhaltung des Manikins kontrollieren und verändern, beispielsweise durch eine Transformation von stehender zu sitzender Position. Neben der manuellen Bewegung der Gliedmaßen des Manikins durch den Anwender wurden auch Algorithmen implementiert, die ein Gehen bzw. Laufen des Manikins simulieren [77]. Hierbei werden die nötigen Transformationen der Körperteile des Manikins automatisch generiert und die Transformationsmatrizen für die facettierten Geometrieobjekte der Körperteile automatisch in das System eingespeist. Die Funktionen hierfür wurden in eigene Threads[4] ausgegliedert, um die gleichzeitige Darstellung der Bewegung und der Simulationsergebnisse im Ausgabefenster der Applikation nicht zu beeinträchtigen. Die Geh- bzw. Laufbewegung des Dummys hat jedoch keine unmittelbare Auswirkung auf die laufende Strömungssimulation. Die Animation wir nur in der Visualisierungsapplikation dargestellt. Eine Übermittlung der Geometrieinformation, d.h Position der Gliedmaßen etc., an den Rechenkern erfolgt erst nach stoppen der Animation.
Abbildung 2.3 zeigt ein im Laufmodus animiertes parametrisches Manikinmodell in einer `iFluids` CFD Simulation. Das Manikin wurde interaktiv auf einem Laufband platziert und anschließend die Laufbewegung initiiert. Die farbigen Stromlinien visualisieren die Umströmung der Objekte in der Simulation (jeweils stationäre Zustände). Wie bei der Modifikation von anderen geometrischen Objekten durch den Anwender, beispielsweise Einrichtungsgegenständen, werden auch bei Modifikationen des Manikinmodells die entsprechenden Informationen über die geometrischen Verschiebungen über die vorhanden Kommunikationsschnittstellen an die Simulation übertragen und entsprechend weiter verarbeitet. Während die Geh- bzw. Laufbewegung des Dummys aktiv ist, erfolgt aktuell keine Übermittlung der zugehörigen Transformationen an die Simulation. Dies geschieht erst nach dem Stoppen selbiger.

[4] Unter einem Thread, bzw. in diesem Kontext genauer einem User Thread oder auf deutsch „leichtgewichtigem Prozessfaden“, versteht man einen vom Hauptprogramm zur Laufzeit gestarteten zusätzlichen Subprozess, welcher eine bestimmte Aufgabe abarbeitet, ohne das Hauptprogramm zu blockieren.

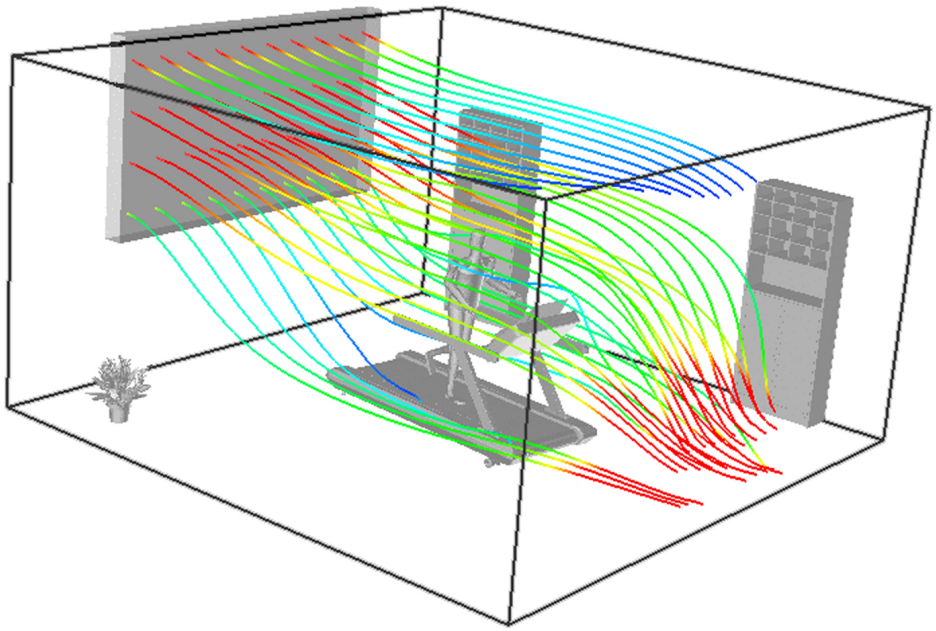


Abbildung 2.3: Animiertes parametrisches Manikinmodell in einer `iFluids` CFD Simulation auf einem Laufband platziert. Die Luftströmung wird durch Stromlinien visualisiert (aus [61]).

Kapitel 3

Methodische Grundlagen

Dieses Grundlagenkapitel widmet sich den methodischen Grundlagen der numerischen Simulation innerhalb der Computational Steering Umgebung. Es wird die Lattice-Boltzmann Methode (LBM) als Alternative zu Navier-Stokes basierten Strömungssimulationen vorgestellt. Da zur Abbildung von thermischen Problemen in dieser Simulationsumgebung ein hybrider Ansatz gewählt wurde, wird auch auf die Methode der Finiten-Differenzen, welche zur Lösung der Advektions/Diffusions-Gleichung verwendet wird, kurz eingegangen. Abschließend folgt in diesem Kapitel noch ein Exkurs zu den verschiedenen Rechengittern in der CFD und insbesondere in `iFluids`.

3.1 Grundlagen der inkompressiblen und kompressiblen Strömungsmechanik

Die Dynamik von Fluiden kann mathematisch durch einen Satz partieller Differentialgleichungen beschrieben werden, die sogenannten Navier-Stokes Gleichungen[1]. Das zugrunde liegende Materialgesetz beschränkt sich dabei auf Newtonsche Flüssigkeiten und Gase, d.h. dass keine erste Normalspannung im Fluid existiert. Somit ist ein linearer Zusammenhang zwischen Geschwindigkeitsgradient und Scherspannungen vorhanden. Des Weiteren ist in einem solchen Fluid die Viskosität konstant [43].

Die Gleichungen basieren auf integraler Erhaltung von Masse, Impuls und Energie in einem infinitesimalen Kontrollvolumen. Die integrale Formulierung kann unter Voraussetzung stetiger Strömungsgrößen über den Satz von Gauß in eine differentielle Form gebracht werden, die auch im Folgenden verwendet wird.

Die Lösung der Navier-Stokes-Gleichungen ergibt ein Strömungsfeld, welches die Geschwindigkeit, Druck und Dichte des Fluids an einem bestimmten Ort zu einem bestimmten Zeitpunkt beschreibt. Ausgehend von diesem Strömungsfeld können anschließend durch Ankopplung weiterer Feldgleichungen auch weitere Größen, etwa die Temperatur des Fluids, berechnet werden.

Im Folgenden wird die Indexschreibweise mit der Einsteinschen Summenkonvention verwendet.

[1] Benannt nach Claude Louis Marie Henri Navier (1785–1836) und George Gabriel Stokes (1819–1903). Üblicherweise wird nur der Impulserhaltungssatz als Navier-Stokes Gleichung bezeichnet.

Für die Normal- und Scherspannungen in Newtonschen Fluiden gilt folgende Beziehung:

$$\sigma_{ij} = -p\delta_{ij} - \frac{2}{3}\mu\frac{\delta u_i}{x_i}\delta_{ij} + \mu\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] \tag{3.1}$$

Hierbei bezeichnet μ die (dynamische) Viskosität des Fluids, $\vec{u}$ dessen Geschwindigkeit am Ort $\vec{x}$. Der Druck im Fluid lässt sich als isotroper Anteil aus der Spur des Spannungstensors berechnen:

$$p = \frac{\text{spur}(\sigma)}{3}. \tag{3.2}$$

Dies gestattet eine Aufteilung des Spannungstensors in Druck und Reibspannungen:

$$\sigma_{ij} = p \cdot \delta_{ij} + \tau_{ij} \tag{3.3}$$

mit

$$\tau_{ij} = \mu\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\delta_{ij}\frac{\partial u_i}{\partial x_i}\right]. \tag{3.4}$$

Werden nun inkompressible Fluide betrachtet, d.h. Dichte $\rho = const.$, ergeben Massen- und Impulserhaltung zusammen ein System von partiellen nichtlinearen Differenzialgleichungen zweiter Ordnung für vier Unbekannte, die drei Komponenten des Geschwindigkeitsvektors $\vec{u}$ und z.B. den Druck p. Dies führt zu den inkompressiben Navier-Stokes-Gleichungen. Auf eine Herleitung wird an dieser Stelle verzichtet, sie kann jedoch in der entsprechenden Literatur, wie beispielsweise [43] oder [25], nachgelesen werden.
Die sogenannte Kontinuitätsgleichung für die Betrachtung der Massenerhaltung lautet für den inkompressiblen Fall:

$$\frac{\partial u_i}{\partial x_i} = 0. \tag{3.5}$$

Für den Impuls gilt unter Verwendung des zuvor formulierten Materialgesetzes (siehe Gleichung 3.3) und der inkompressiblen Kontinuitätsgleichung:

$$\rho\left(\frac{\partial u_j}{\partial t} + u_i\frac{\partial u_j}{\partial x_i}\right) = F_j - \frac{\partial p}{\partial x_j} + \mu\left(\frac{\partial^2 u_j}{\partial x_i^2}\right). \tag{3.6}$$

Die auf das Fluid einwirkenden Kräfte werden dabei mit $\vec{F}$ bezeichnet.
Für Fluide mit kleiner Strömungsgeschwindigkeit, d.h. $|\vec{u}| \ll c_s$, wobei c_s die Schallgeschwindigkeit im Fluid bezeichnet, kann näherungsweise angenommen werden, dass das Fluid sich inkompressibel oder schwach kompressibel verhält. In schwach kompressiblen Fluiden kann

näherungsweise angenommen werden, dass der Druck p und die Dichte ρ linear voneinander abhängig sind.

$$\rho = c_s^2 \cdot p \text{ mit } \frac{\partial(c_s)}{\partial \rho} \sim 0 \tag{3.7}$$

Die Gleichungen 3.5, 3.6 und 3.7 reichen aus, um ein inkompressibles Strömungsfeld zu beschreiben.
Bei kompressiblen Fluiden und den daraus resultierenden nicht vernachlässigbaren Temperatureinflüssen wird zur Bestimmung der dann nicht mehr abhängigen Größen ρ und p eine zusätzliche Gleichung benötigt. Diese ergibt sich aus der Energieerhaltung. Auch hier wird für die Herleitung auf [43] oder [25] verwiesen. Die Kontinuitätsgleichung für den kompressiblen Fall lautet:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0. \tag{3.8}$$

Für die kompressible Impulsbilanz erhält man:

$$\rho \left(\frac{\partial u_j}{\partial t} + u_i \frac{\partial u_j}{\partial x_i} \right) = F_j - \frac{\partial p}{\partial x_j} + \frac{\partial}{\partial x_j} \left(\mu \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right) - \frac{2}{3} \mu \delta_{ij} \frac{\partial u_i}{\partial x_i} \right). \tag{3.9}$$

Des Weiteren gilt für ein ideales kalorisches Gas die folgende Energiegleichung:

$$\rho \cdot c_p \left(\frac{\partial T}{\partial t} + u_i \frac{\partial T}{\partial x_i} \right) = \left(\frac{\partial p}{\partial t} + u_i \frac{\partial p}{\partial x_i} \right) + \frac{\partial}{\partial x_i} \left(\lambda \frac{\partial T}{\partial x_j} \right) + \mu \cdot \Phi. \tag{3.10}$$

λ steht für die Wärmeleitfähigkeit und T für die Temperatur. Innere Wärmequellen sind in Gleichung 3.10 vernachlässigt. c_p bezeichnet die spezifische Wärmekapazität des Fluids und Φ die Dissipationsrate. Sie muss an jedem Ort innerhalb des Strömungsfeldes größer als Null sein.

$$\Phi = 2 \left(\frac{\partial u_i}{\partial x_i} \right)^2 + \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)^2 + \left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i} \right)^2 + \left(\frac{\partial u_k}{\partial x_j} + \frac{\partial u_j}{\partial x_k} \right)^2 - \frac{2}{3} \cdot (\nabla \cdot \vec{u})^2 \tag{3.11}$$

Zur eindeutigen Lösung des Gleichungssystems sind zusätzlich Anfangs- und Randbedingungen notwendig. Hierbei sind Dirichlet-, Neumann- und gemischte (sog. Cauchy-) Randbedingungen möglich. Bei Dirichlet-Bedingungen wird für eine Größe, beispielsweise Strömungsgeschwindigkeit oder Temperatur, ein konstanter Wert entlang des Randes vorgegeben. Bei Neumann-Bedingungen wird die erste Ableitung einer Größe am Rand in Normalenrichtung angegeben. Ein Beispiel hierfür ist ein gegebener Wärmestrom durch eine Wand. Bei Cauchy-Bedingungen sind an einem Teil der Berandung Dirichlet- und an einem anderen Teil der Berandung Neumann-Bedingungen gegeben.
Bei Anfangsbedingungen handelt es sich immer um Dirichlet-Bedingungen.

3.2 Die Lattice-Boltzmann Methode in der Strömungssimulation

Der CFD[2]-Simulationskern der in dieser Arbeit beschriebenen Computational Steering Umgebung basiert nicht auf einem Ansatz, der die Navier-Stokes-Gleichungen direkt löst, sondern auf der Lattice-Boltzmann Methode (oder „Gitter-Boltzmann-Methode“). Dieses Kapitel soll einen Einblick in die dahinter stehende Theorie liefern. Die Darstellung orientiert sich dabei an [33].

3.2.1 Top-Down- vs. Bottom-Up-Ansätze

Die konventionelle Simulation von Fluidströmungen basiert auf (üblicherweise nichtlinearen) partiellen Differentialgleichungen bzw. Integro-Differentialgleichungen. Diese werden anschließend mit Finite-Differenzen (FD), Finite-Volumen (FV), Finite-Elemente (FE) oder Spektralmethoden diskretisiert. Die dadurch entstehenden algebraischen Gleichungen oder gewöhnliche Differentialgleichungssysteme werden mit Hilfe von numerischen Standardmethoden gelöst ([14], [58]).
Die Gittergas-Verfahren (LGCA[3]) und die (historisch) daraus abgeleitete Lattice-Boltzmann Methode (LBM) sind hingegen Vertreter eines Bottom-Up-Ansatzes, bei dem der Ausgangspunkt ein diskretes gasdynamisches Modell ist, welches durch Bildung von (statistischen) Momenten die gewünschten Größen (Masse, Impuls, Schubspannungen, etc. für die Navier-Stokes-Gleichungen) liefert ([14], [58]).
Für den in dieser Arbeit vorgestellten Simulationscode kommt die Lattice-Boltzmann Methode und somit ein Bottom-Up-Ansatz zur Anwendung.

3.2.2 Die Boltzmann-Gleichung

Die Lattice-Boltzmann Methode ist eine aus der statistischen Physik abgeleitete, auf einer vereinfachten kinetischen Theorie basierende mesoskopische Formulierung, durch welche asymptotisch (Chapman-Enskog-Entwicklung) die inkompressiblen Navier-Stokes-Gleichungen (siehe Gleichungen 3.5, 3.6 und 3.7) gelöst werden. Hierunter versteht man, dass unter Einführung eines formalen Skalierungsparameters ϵ die Zeit in einer Multiskalenentwicklung dargestellt werden kann, sodass sich mit einer entsprechenden Skalierung im Grenzfall $\epsilon \to 0$ die Navier-Stokes Gleichungen ergeben (siehe hierzu [53]).

Bei der zugrunde liegenden Boltzmann-Gleichung[4] handelt es sich um eine Integro-Differentialgleichung. Sie beschreibt eine Veränderung der statistischen Verteilung von Teilchen f in einem Fluid. Die Verteilungsfunktion $f(t, \vec{x}, \vec{\xi})$ gibt die Massendichte von Partikeln im Phasenraumvolumen $(d\vec{x}, d\vec{\xi})$ an, die sich zum Zeitpunkt t am Ort $\vec{x}$ mit der Propagationsgeschwindigkeit $\vec{\xi}$ befinden. Die Massendichte kann als relative Häufigkeit und daher als Aufenthaltswahrscheinlichkeit interpretiert werden. Die Boltzmann-Gleichung erhält man aus deren Veränderung

[2] Computaional Fluid Dynamics.
[3] Lattice-Gas Cellular Automata.
[4] Benannt nach dem Physiker Ludwig Boltzmann (1844-1906).

nach der Zeit, die ausschließlich durch einen Kollisionsterm $\Omega(f)$ bestimmt wird:

$$\frac{Df}{Dt} = \frac{\partial f}{\partial t} + \vec{\xi} \cdot \frac{\partial f}{\partial \vec{x}} + \frac{\vec{F}}{m} \cdot \frac{\partial f}{\partial \vec{\xi}} = \Omega(f). \tag{3.12}$$

Der erste Term des partiellen Differentials gibt die ortsfeste Veränderung an, der zweite den advektiven Teilchentransport im Ortsraum. Der dritte Term stellt einen advektiven Teilchentransport im Phasenraum dar, kann jedoch einfacher als Quellterm unter Einfluss einer externen Kraft $\vec{F}$ interpretiert werden.
Der Kollisionsoperator $\Omega(f)$ modelliert den Kollisionsprozess im Kontrollvolumen von je zwei Partikelverteilungen mit den Geschwindigkeiten $\left\{\vec{\xi}, \vec{\xi_1}\right\}$ vor bzw. $\left\{\vec{\xi'}, \vec{\xi_1'}\right\}$ nach der Kollision mit dem differentiellen Wirkungsquerschnitt σ ([75]):

$$\Omega(f) = \int \int \sigma(\Omega) \left|\vec{\xi} - \vec{\xi_1}\right| \left[f(\vec{\xi'}) f(\vec{\xi_1'}) - f(\vec{\xi}) f(\vec{\xi_1})\right] d\Omega d\vec{\xi_1}. \tag{3.13}$$

Der Kollisionsoperator $\Omega(f)$ macht die Boltzmann-Gleichung zu einer aufwändig zu lösenden Integro-Differential-Gleichung. Unter bestimmten Annahmen an das Strömungsfeld kann er vereinfacht werden, muss jedoch trotzdem noch in der Lage sein, die betrachteten physikalischen Eigenschaften abzubilden.
Der einfachste, sogenannte *BGK*-Ansatz wurde von Bhatnagar, Gross und Krook im Jahr 1954 eingeführt ([3]). Er linearisiert den Kollisionsterm aufgrund der Annahme, dass sich das Gas als Kontinuum verhält und die Abweichungen vom Gleichgewichtszustand sehr gering sind. Somit ergibt sich folgender, vereinfachte Ansatz mit der Gleichgewichtsverteilung $f^{(0)}$:

$$\Omega(f) \approx \Omega_{BGK}(f) = -\frac{1}{\tau}(f - f^{(0)}). \tag{3.14}$$

τ bezeichnet die Relaxationszeit, welche als gemitteltes Zeitintervall zwischen zwei Stoßvorgängen zweier Teilchen interpretiert werden kann. Sie beschreibt die Tendenz der Verteilungsfunktion f, sich dem Zustand einer einheitlichen Dichte- und Geschwindigkeitsverteilung im Fluid anzunähern. Die Gleichgewichtsverteilung $f^{(0)}(\rho, \vec{u})$ erhält man als analytische Lösung der Boltzmann-Gleichung zu:

$$f^{(0)}(\rho, \vec{u}) = \frac{\rho}{(2\pi RT)^{3/2}} \exp\left(-\frac{(\vec{\xi} - \vec{u})^2}{2RT}\right). \tag{3.15}$$

R bezeichnet die universelle Gaskonstante, T die Temperatur des Fluids.
Durch die Chapman-Enskog-Entwicklung ([11]) ist es möglich, die Navier-Stokes-Gleichungen und ihre Materialparameter direkt aus der Boltzmann-Gleichung abzuleiten. Es handelt sich hierbei um eine aufwändige Multiskalenanalyse. Sie ermöglicht den Nachweis der Gleichwertigkeit der beiden Ansätze unter bestimmten Voraussetzungen wie kleinen Knudsenzahlen.
Die Knudsen-Zahl ist definiert als

$$Kn = \frac{\lambda_r}{L_r}, \tag{3.16}$$

wobei λ_r für den mittleren freien Weg[5] $[m]$ und L_r für die charakteristische Länge $[m]$ steht. Für $Kn \ll 1$ verhält sich das Fluid wie ein Kontinuum und die Abweichungen vom Gleichgewichtszustand sind klein. Die Knudsen-Zahl ist ein Maß für die Dichte einer Strömung:

$$Kn \sim \frac{1}{\rho} \tag{3.17}$$

Der Gültigkeitsbereich des BGK-Ansatzes ist weiterhin eingeschränkt durch folgende Annahmen:

- Es finden nur binäre Kollisionen Berücksichtigung. Daraus ergibt sich insbesondere eine Gültigkeit für Gase, bei denen die Partikelgröße sehr viel kleiner ist als der mittlere Abstand zwischen zwei Teilchen.
- Es existiert molekulares Chaos. Dies bedeutet, dass sich die Wahrscheinlichkeitsdichten von Mehrteilchenkonfigurationen in Produkte von Einteilchenwahrscheinlichkeitsdichten faktorisieren lassen, da die Teilchengeschwindigkeiten vor und nach der Kollision nicht miteinander korrelieren.

  Dies kann angenommen werden, wenn die Zeitdauer zwischen zwei Stößen sehr viel größer als die Zeitdauer der Kollision selbst ist.
- Von außen einwirkende Kräfte müssen wesentlich kleiner sein als die, die bei der Kollision selbst wirken, um auszuschließen, dass diese durch die äußeren Kräfte wesentlich beeinflusst wird.

Die makroskopischen Größen wie Dichte (3.18) und Impuls (3.19) lassen sich als Momente der Verteilungsfunktionen bezüglich der mikroskopischen Geschwindigkeiten $\vec{\xi}$ ausdrücken. Die Dichte stellt ein Moment nullter Ordnung und die Impulsdichte ein Moment erster Ordnung dar.

$$\rho(t, \vec{x}) = \int_{-\infty}^{\infty} f(t, \vec{\xi}, \vec{x}) d\vec{\xi} \tag{3.18}$$

$$\rho(t, \vec{x})\vec{u}(t, \vec{x}) = \int_{-\infty}^{\infty} f(t, \vec{\xi}, \vec{x}) \cdot \vec{\xi} d\vec{\xi} \tag{3.19}$$

Dies bedeutet also, dass die makroskopische Geschwindigkeit $\vec{u}(t, \vec{x})$ den Mittelwert der Teilchengeschwindigkeit $\vec{\xi}$ darstellt. Der Spannungstensor ergibt sich als Moment zweiter Ordnung mit $\vec{\zeta} = (\vec{\xi} - \vec{u})$ zu:

$$D_{\alpha\beta}(t, \vec{x}) = \int_{-\infty}^{\infty} \zeta_\alpha \zeta_\beta f(t, \vec{\xi}, \vec{x}) d\vec{\xi}. \tag{3.20}$$

[5] Die mittlere freie Weglänge ist die durchschnittliche Weglänge, die ein Teilchen ohne Wechselwirkung mit anderen Atomen, Molekülen oder Berandungen zurücklegt.

Hierin lässt sich $\zeta_\beta f(t,\vec{\xi},\vec{x})d\vec{\xi}$ als differentieller Impulsanteil in β-Richtung und $\zeta_\alpha\zeta_\beta f(t,\vec{\xi},\vec{x})d\vec{\xi}$ als Fluss in α-Richtung deuten [33].
Eine weitere wichtige Größe für die LBM stellt der direkt aus der Teilchengeschwindigkeit hervorgehende Impulsfluss- und Spannungstensor Π dar.

$$\Pi_{\alpha\beta}(t,\vec{x}) = \int_{-\infty}^{\infty} \xi_\alpha\xi_\beta f(t,\vec{xi},\vec{x})d\vec{\xi} = u_\alpha u_\beta\rho + D_{\alpha\beta}(t,\vec{x}) \tag{3.21}$$

Für $\vec{u} = 0$ stimmen der Spannungstensor und der Impulsstromtensor überein.

Wird die Boltzmann-Gleichung (Gleichung 3.12) unter Substitution von $f\left(t,\vec{x},\vec{\xi}\right)$ durch $f\left(t,\vec{x},\vec{e_i}\right)$ mit einem Satz aus N diskreten mikroskopischen Geschwindigkeiten e_i anstelle der kontinuierlichen Geschwindigkeiten ξ diskretisiert, lässt sich die diskrete Boltzmann-Gleichung (Gleichung 3.22) herleiten, hier mit BGK-Modell:

$$\frac{\partial f_i}{\partial t} + \vec{e_i}\frac{\partial f_i}{\partial \vec{x}} = -\frac{1}{\tau}\left(f_i - f_i^{(0)}\right) - \vec{F}\cdot\frac{\partial f_i}{\partial \vec{e_i}}. \tag{3.22}$$

Die Gleichgewichtsfunktion $f_i^{(0)}$ erhält man aus einer Reihenentwicklung der kontinuierlichen Gleichgewichtsfunktion 3.15 nach der Geschwindigkeit $\vec{u}$, die nach der zweiten Ordnung abgebrochen und numerisch integriert wird (siehe z. B. [33]). Als Konsequenz der Reihenentwicklung ist jedoch die Gitter-Boltzmann-Methode mit diesem Ansatz nur für kleine Machzahlen gültig. Die diskretisierte Gleichgewichtsverteilung lautet [21]:

$$f_i^{(0)} = t_i\varrho\left[1 + 3\frac{(\vec{e_i}\cdot\vec{v})}{c_0^2} + \frac{9}{2}\frac{(\vec{e_i}\cdot\vec{v})^2}{c_0^4} - \frac{3}{2}\frac{\vec{v}^2}{c_0^2}\right]. \tag{3.23}$$

mit den Gauß-Gewichten t_i, die vom Diskretisierungsmodell abhängen. c_0 bezeichnet die dem Diskretisierungsmodell zugrunde liegende Einheitspropagationsgeschwindigkeit. Diskretisierungsmodelle sind das D1Q3 Modell für 1D, das D2Q9 Modell für 2D oder die D3Q15, D3Q19 oder D3Q27 Modelle für 3D. Hierbei gibt die erste Zahl die Dimension und die zweite Zahl die Anzahl der diskreten Richtungen im Modell an.
Als Nächstes wird nun ein regelmäßiges Gitter eingeführt und Gleichung 3.22 räumlich und zeitlich durch ein Finite-Differenzen Verfahren diskretisiert. Hierdurch erhält man die Gitter-Boltzmann-Gleichung (Gleichung 3.24) unter Vernachlässigung externer Kräfte:

$$f_i\left(t+\Delta t,\vec{x}+\vec{e_i}\Delta t\right) = f_i(t,\vec{x}) - \frac{\Delta t}{\tau}(f_i(t,\vec{x}) - f_i^{(0)}(t,\vec{x})). \tag{3.24}$$

Die Einheitspropagationsgeschwindigkeit ergibt sich dadurch zu $c_0 = \frac{\Delta x}{\Delta t}$.

3.3 Lösung der Advektions/Diffusions-Gleichungen mit der Methode der Finiten-Differenzen

Im Folgenden soll auf die theoretischen Grundlagen zur Lösung der Gleichungen für den konvektiven Wärmeübergang (siehe Gleichung 3.25) mit Hilfe der Finite-Differenzen Methode eingegangen werden. Bei Gleichung 3.25 wurde von Inkompressibilität und vernachlässigbarer Dissipation ausgegangen, c_p ist die spezifische Wärmekapazität. Die Darstellungen orientieren sich an [13] und [46].

$$\rho c_p \left(\frac{\partial T}{\partial t} + u_i \frac{\partial T}{\partial x_i} \right) = \lambda \frac{\partial^2 T}{\partial x_i^2} \tag{3.25}$$

Konvektiver Wärmeübergang findet dann statt, wenn ganze Molekülgruppen von einem Ort zu einem anderen mit unterschiedlicher Temperatur bewegt werden. Konvektion ist somit auf Flüssigkeiten und Gase beschränkt.
In den Fällen, bei denen die Strömung des Fluids durch Auftriebskräfte induziert wird, beispielsweise bei einem von Luft umgebenen heißen Zylinder, bei dem es durch Erwärmung der Luft zu Dichteänderungen kommt, spricht man von *freier Konvektion*. Der Wärmeübergang wird hierbei durch die spezifischen Fluideigenschaften bestimmt.
Wenn die Strömung des Fluids hingegen hauptsächlich durch äußere Einflüsse induziert wird, beispielsweise durch ein Gebläse, spricht man von *erzwungener Konvektion*.
Wenn äußere Einflüsse vorhanden sind, aber das Verhältnis von Auftriebskräften zu diesen nicht vernachlässigt werden kann, spricht man von *gemischter Konvektion*.
Die Entdeckung der Abhängigkeit des konvektiven Wärmeübergangs von der Temperaturdifferenz zwischen Fluid T_f und Festkörper T_s und der Größe der Kontaktfläche A geht auf Newton zurück. Es gilt:

$$Q = hA(T_s - T_f). \tag{3.26}$$

Hierbei ist Q der Wärmestrom in [W] und h der sogenannte Wärmeübergangskoeffizient in [W/m^{2}K]. Werte von h für verschiedenste Strömungsparameter wurden in unzähligen experimentellen Versuchen gewonnen. Die meisten Untersuchungen wurden mit dimensionslosen Größen in Beziehung gesetzt. Hierzu wird die *Nusselt*-Zahl aus dem Ähnlichkeitsgesetz der Wärmeübertragung verwendet:

$$Nu = \frac{hL}{\lambda}. \tag{3.27}$$

L bezeichnet die charakteristische Länge in [m] und λ die Wärmeleitfähigkeit des Fluids in [W/m K].

Bei der Finite-Differenzen Methode handelt es sich um das einfachste numerische Verfahren zur Lösung von partiellen Differenzialgleichungen wie der Wärmeleitungsgleichung (Energiegleichung).

Bei diesem Ansatz wird die ursprüngliche Gleichung dadurch angenähert, dass abgeleitete Ausdrücke durch Differenzenquotienten ersetzt werden und das Gebiet mit einer endlichen Anzahl von Gitterpunkten diskretisiert wird. An diesen wird dann die genäherte Lösung der ursprünglichen Gleichung durch Auswertung der Differenzengleichungen bestimmt.

3.4 Die hybride thermische LB-Methode

Da thermische Simulationen, die auf der thermischen Lattice-Boltzmann Methode ([55]) basieren, unter numerischen Stabilitätsproblemen leiden, wird für den Simulationskern, der in dieser Arbeit zum Einsatz kommt, ein anderer Ansatz gewählt.
Anstelle einer thermischen LBM kommt eine hybride thermische LB-Methode entsprechend der Arbeit von Lallemand und Lou [34] zur Anwendung, wie sie von van Treeck in [58] vorgestellt und weiterentwickelt wurde. Hierbei kommt für die Strömungsseite das *multiple-relaxation-time* Lattice-Boltzmann Schema (MRT), welches von d'Humières et al. [15] vorgeschlagen wurde, zur Anwendung. Ausgehend von der Diskretisierung der diskreten Boltzmann Gleichung hinsichtlich ihres mikroskopischen Geschwindigkeitsraums und durch Anwendung eines Finite-Differenzen Verfahrens erster Ordnung in Raum und Zeit, wie z.B. in [34], findet die Advektion im Phasenraum $F = \mathbb{R}^B$ statt, während die Kollision in einem physikalisch gleichwertigem Momentenraum $M = \mathbb{R}^B$ durchgeführt wird; B bezeichnet die Anzahl der (semi)-diskreten Geschwindigkeiten im Modell [15, 34].

$$\vec{f}(t + \Delta t, \vec{x} + \vec{\xi}_i \Delta t) - \vec{f}(t, \vec{x}) = -\mathbf{M}^{-1} \hat{S} \left[\vec{m}(t, \vec{x}) - \vec{m}^{(0)}(t, \vec{x}) \right] \tag{3.28}$$

Der Phasenraum ist als ein D-dimensionales Gitter durch eine Menge von B Verteilungsfunktionen $\{f_i(t, \vec{x}) : i = 0, ..., b\}$ definiert, welches zu B Geschwindigkeitsvektoren $\left\{ \vec{\xi}_i : i = 0, ..., b \right\}$ mit $b = (B - 1)$ korrespondiert. Die Herleitung der Transformationsmatrix $\mathbf{M}$ ist in [15] gegeben. $\mathbf{M}$ transformiert einen Vektor des Phasenraums $\vec{f}$ linear in einen Vektor im orthogonalisierten Momentenraum $\vec{m}$: $\vec{m} = \mathbf{M}\vec{f}$. Die Rücktransformation geschieht mit $\mathbf{M}^{-1}$.
Das Modell erlaubt die Optimierung der Relaxationskoeffizienten hinsichtlich der Stabilität und Dispersionseigenschaften (Lallemand [34]). Dadurch lassen sich Feldverteilungen, die durch instabile oder sehr schwach gedämpfte Moden des Rechenverfahrens erzeugt wurden und keine physikalischen Eigenschaften des Fluids modellieren, sondern nur reine Artefakte des mathematischen Modells darstellen, von den „physikalischen“ Moden trennen und stärker dämpfen [74].
Alle nicht erhaltenen kinetischen Momente werden zu ihrem Gleichgewicht $m_i^{(0)}(t, \vec{x})$ relaxiert. Zur Diskretisierung der Temperaturgleichung wird ein Finite-Differenzen Verfahren im Raum und ein explizites Eulerverfahren in der Zeit angewandt:

$$\frac{T_{i,j,k}(t + \Delta t^{FD}) - T_{i,j,k}(t)}{\Delta t^{FD}} = -\vec{j}_{i,j,k}(t) \nabla^{(h)}_{i,j,k} T_{i,j,k}(t) + \alpha \Delta^{(h)}_{i,j,k} T_{i,j,k}(t). \tag{3.29}$$

Hierbei beziehen sich die Indizes i, j, k auf die Position im dreidimensionalen Rechengitter. Der Gradient $\nabla^{(h)}_{i,j,k}$ und der Laplace Operator $\Delta^{(h)}_{i,j,k}$ sind die diskretisierten Finite-Differenzen

Operatoren auf dem Rechengitter mit Gitterweite h mit den Raumrichtungen i, j und k (siehe [58]). α ist die Temperaturleitfähigkeit.
Die beiden Verfahren sind explizit gekoppelt, d.h. das durch das MRT Verfahren gewonnene Geschwindigkeitsfeld wird in die Energiegleichung eingesetzt, während deren Lösung zur Berechnung der Auftriebskräfte $F_z(\vec{x}, t)$ (siehe Gleichung 3.30) in Sinne einer Boussinesq Approximation[6] genutzt wird. Die Kraft wird als Änderung des Impulses $\vec{j}$ in z-Richtung aufgebracht; jeweils zur Hälfte vor und nach der Relaxation der kinetischen Momente [58].

$$F_z(\vec{x}, t) = g_z \beta \Delta T(\vec{x}, t) \tag{3.30}$$

Da die numerische Methode nur für kleine Mach-Zahlen gültig ist, muss die Amplitude von $F_z(\vec{x}, t)$ ausreichend klein gewählt werden. Sie wird vom Faktor $g_z\beta$ bestimmt, wenn $T \in [-1; +1]$ und $\Delta t = 1$. β ist der Wärmeausdehnungskoeffizient und g_z die Beschleunigung aufgrund von Gravitation.

$$Ra = \frac{Pr g_z \beta 2 T_0 L^3}{\nu^2} \tag{3.31}$$

Für eine vorgegebene Rayleigh Zahl Ra (siehe Gleichung 3.31) und eine ebenfalls gegebene Prandtl Zahl $Pr = \nu/\alpha$ und durch Setzen von $\beta = 1$ können die Parameter Viskosität ν und Diffusionskoeffizient α berechnet werden. Die Relaxationskoeffizienten können dann mit den Formeln aus [34] bestimmt werden, wobei ν die Stabilitätskriterien des MRT Schemas erfüllen muss. L ist die charakteristische Länge des dimensionslosen Systems.
Um die numerische Stabilität des hybriden Schemas zu untersuchen, müssen die Eigenwerte der resultierenden linearisierten Iterationsmatrix berechnet werden (siehe [34]). Dies kann zwar nicht mehr analytisch erfolgen, aber man kann Abschätzungen treffen, entweder bestimmt durch den LBM-Teil oder von der Konvektions-Diffusions Gleichung für die Temperatur ([34]). Für den LBM Teil ist die Auflösung in Zeit und Raum gekoppelt, sodass für eine gegebene Auflösung der Zeitschritt Δt festgelegt ist. Die Stabilität wird durch die Kollisionskoeffizienten bestimmt. Aus Stabilitätsgründen müssen diese kleiner als 2 sein. Mit dem MRT Schema können mit Werten für die Kollisionskoeffizienten von bis zu ca. $1,999$ – abhängig von der Geometrie der Strömung und den Randbedingungen – stabile Simulationen erzielt werden.
Für die Advektions-Diffusions Gleichung mit einer Advektionsgeschwindigkeit u und einem Diffusionskoeffizienten α existieren mehrere Restriktionen für die Zeitschrittweite Δt^{FD}.
Die erste Einschränkung ergibt sich aus der CFL-Zahl[7]:

$$\Delta t^{FD} < \Delta x / u. \tag{3.32}$$

[6] Bei der Boussinesq Approximation handelt es sich um eine Vereinfachung der Navier-Stokes-Gleichungen. Ihr liegen die Annahme einer vernachlässigbaren Dichtevariation und nicht zu großer Temperaturgradienten zu Grunde.

[7] Courant-Friedrichs-Lewy- oder Courant-Zahl. Sie ist definiert als: $c = \frac{u \Delta t}{\Delta x}$. Hierbei bezeichnet u die Geschwindigkeit, Δt die Zeitschrittlänge und Δx das Längenintervall, d.h. die Gitterweite. Für Stabilität beim Rückwärts-Euler-Verfahren ist $c < 1$ gefordert. Dies bedeutet, dass der Zeitschritt und die Gitterweite so gewählt werden müssen, dass ein Teilchen bzw. eine Welle während eines Zeitschrittes nur maximal die Entfernung Δx zurücklegen.

Für den diffusiven Teil gilt weiter

$$\Delta t^{FD} < \Delta x^2/(2\alpha), \tag{3.33}$$

wodurch sich schlussendlich nach [65] ergibt:

$$\Delta t^{FD} < 2\alpha/u^2. \tag{3.34}$$

Demnach ist es nach van Treeck [65] für gewisse α Werte entweder vorteilhaft oder sogar notwendig, verschiedene Zeitschrittweiten für den LBM und den FD Teil zu wählen. In dem in dieser Arbeit verwendeten Rechenkern (siehe hierzu van Treeck [58]) wird allerdings auf eine unterschiedliche Wahl der Zeitschrittweite verzichtet.

3.5 Turbulenzmodelle

Die nachfolgende Darstellung der Turbulenzmodelle folgt im Wesentlichen der Darstellung von van Treeck in [58].
Die Abbildung von Turbulenzen stellt ein besonderes Problem in der Stömungsmechanik dar. Turbulenz tritt bei hohen Reynoldszahlen auf und lässt sich als instationär, nichtlinear, mischungsintensiv, unregelmäßig, dreidimensional und dissipativ charakterisieren [67, 58]. Anders als bei laminaren Strömungen findet bei turbulenten Strömungen auch eine Vermischung von Fluidteilchen quer zur Strömung statt.
Theoretisch sind die Navier-Stokes-Gleichungen in der Lage das gesamte Spektrum einer turbulenten Strömung abzubilden, wobei dann alle charakteristischen Skalen der Strömung räumlich und zeitlich aufgelöst werden müssen. Die sogenannten Mikroskalen von *Kolmogorov* dienen zur Abschätzung der kleinsten charakteristischen Längeneinheit l_K und Zeiteinheit t_K einer Strömung (siehe beispielsweise [35] oder [58]). Sie sind abhängig von der Viskosität ν und der Dissipationsrate ϵ eines Fluids.
Wenn man dies auf eine Direkte Numerische Simulation[8] (DNS) anwendet, kommt man zu der Erkenntnis, dass selbst bei Nutzung von Höchstleistungsrechnern die Lösung der Navier-Stokes-Gleichungen auf Reynoldszahlen $< \mathcal{O}(10^4)$ beschränkt ist, wie in [62] beschrieben.
Zur Lösung dieses Problems wurden nun Ansätze entwickelt, bei denen nicht mehr das gesamte turbulente Spektrum der Strömung raumzeitlich aufgelöst wird, sondern die turbulenten Skalen modelliert und als Einfluss auf die simulierten Skalen der Strömung aufgebracht werden. Innerhalb dieser Ansätze gibt es wiederum verschiedene Abstufungen. Im Gegensatz zur DNS, in der alle Skalen der Strömung simuliert werden, existiert die sogenannte *Reynolds averaged Navier-Stokes Simulation* (RANS). Bei dieser handelt es sich um eine statistische Betrachtungsweise. Hierbei wird das gesamte Turbulenzspektrum modelliert. Die turbulenten Scheinspannungen sind dabei im voll ausgebildeten turbulenten Bereich vorherrschend und verschwinden im wandnahen Bereich, welcher von viskosen Kräften geprägt ist. Die Suche

[8] Bei der Direkten Numerischen Simulation werden alle relevanten Raum- und Zeitskalen direkt aufgelöst. Somit handelt es sich hierbei um die genaueste Methode zur Strömungsberechnung. Da die Betrachtung einer turbulenten Strömung sowohl eine sehr feine räumliche Auflösung als auch eine sehr feine zeitliche Diskretisierung erfordert, ist die Anwendbarkeit auf reale Probleme sehr stark eingeschränkt.

nach Lösungen für die unbekannten Spannungen wird als Schließungsproblem der Turbulenz bezeichnet.
Zur Schließung dieses Gleichungssystems sind verschiedene Modelle verfügbar, wie beispielsweise das Zwei-Gleichungs-k-ϵ-Modell [36]. Eine Übersicht über die verschiedenen Modelle wird in [7] gegeben.
Ein Mittelweg zwischen DNS und RANS ist die *Large-Eddy* Simulation (LES). Hierbei werden nur die kleinen Skalen der Strömung modelliert, während die großen Skalen durch das Verfahren direkt aufgelöst und simuliert werden. Ein Vorteil dieses Verfahrens ist, dass nur noch Turbulenzstrukturen, welche aufgrund des gewählten Gitterabstandes nicht mehr aufgelöst werden können, modelliert werden müssen und die dafür nötigen Feinstrukturmodelle einfacher sein können als solche, die das gesamte Spektrum abdecken müssen [7, 49, 67].
Dieses Verfahren ist zwar hinsichtlich der nötigen Rechenleistung deutlich teurer als ein RANS Verfahren, jedoch wird es mit steigender verfügbarer Rechenleistung zunehmend interessant, da sich hiermit bessere Einblicke in den Charakter einer Strömung gewinnen lassen, die darauf beruhen, dass bei der LES eine räumliche und nicht wie bei RANS eine zeitliche Mittelung vorgenommen wird.
Im LBM Rechenkern der hier vorgestellten CS Umgebung `iFluids` wurde von van Treeck [65] ein auf Smagorinsky basierendes LES Modell implementiert.

3.6 Rechengitter in der CFD

In der numerischen Strömungssimulation nimmt die Generierung des Rechengitters eine bedeutende Rolle ein, da das erzeugte Gitter Grundlage aller weiteren Simulationsschritte ist und maßgeblich die Genauigkeit der gewonnenen Ergebnisse beeinflusst. In klassischen Ansätzen in der Strömungssimulation kann die Gittergenerierung insbesondere bei sehr komplexen Geometrien zu einer sehr aufwendigen und langwierigen Aufgabe werden.
Generell wird in der CFD zwischen strukturierten und unstrukturierten Gittern unterschieden (siehe Abbildung 3.1), wobei nach [25] im Allgemeinen strukturierte Gitter, verglichen mit unstrukturierten Gittern, als effizienter hinsichtlich Genauigkeit, Rechenzeit und Speicherbedarf anzusehen sind.
In der in dieser Arbeit vorgestellten Simulationsumgebung, welche auf der Lattice-Boltzmann Methode basiert, kommen ausschließlich strukturierte, kartesische Gitter zur Anwendung.

3.6.1 Unstrukturierte Gitter

Bei unstrukturierten Gittern wird das zu vernetzende Gebiet mit Punkten gefüllt, welche anschließend durch Dreiecke, Vierecke oder Polygone (in 2D) bzw. Tetraeder, Prismen, Pyramiden, Hexaeder oder beliebige Polyeder (in 3D) verbunden werden.
Im industriellen Umfeld hat sich dieser Ansatz zunehmend durchgesetzt, da sich beliebige Geometrien mit unstrukturierten Gittern besser abbilden lassen. Bei Verwendung von strukturierten Gittern ist unter Umständen ein sehr feines Gitter zur guten Abbildung von z. B. gekrümmten Objekten erforderlich. Dies führt jedoch zu einem gesteigerten Rechenaufwand. In der Industrie wird die geringere Genauigkeit bei der Verwendung von unstrukturierten Gittern gegenüber strukturierten Gittern aufgrund der besseren und genaueren Abbildungsmöglichkeit von beliebigen Geometrien mit weniger Freiheitsgraden oft vernachlässigt. Allerdings birgt die

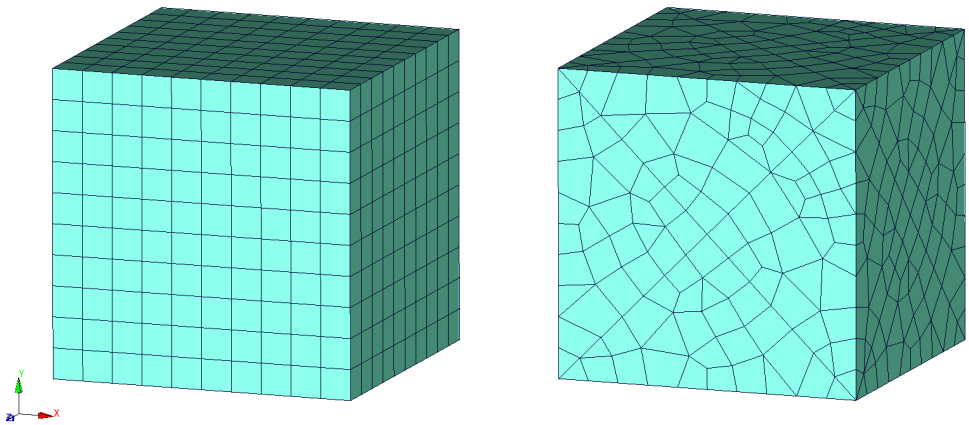


Abbildung 3.1: Strukturiertes (links) und unstrukturiertes (rechts) Rechengitter.

genauere Vernetzung von Geometrieobjekten mit unstrukturierten Gittern auch einen höheren Vernetzungsaufwand.
Ein Vorteil von unstrukturierten Gittern besteht jedoch in der Möglichkeit, lokale Gebiete mit einer veränderten Genauigkeit aufzulösen, ohne dass die Vernetzung des übrigen Rechengebiets dadurch beeinflusst wird. Dadurch sind lokale Netzadaptionen durch lokale Verfeinerung oder Vergröberung möglich.

3.6.2 Strukturierte Gitter

Für die Vernetzung eines Gebiets mit strukturierten Gittern wird dieses durch Gruppen von sich schneidenden Linien, eine Gruppe je Raumrichtung, unterteilt. Die Gitterpunkte ergeben sich folglich an den Schnittpunkten der einzelnen Linien.
Bei strukturierten Gittern wird nochmals in Kartesische Gitter, nicht-uniforme Kartesische Gitter, körperangepasste (*body-fitted*) strukturierte Gitter sowie multiblockstrukturierte Gitter unterschieden [25]. In der an dieser Stelle beschriebenen Implementation eines numerischen Strömungslösers kommen ausschließlich uniforme Kartesische Gitter zur Anwendung. Bei diesen Gittern sind die Abstände zwischen den Gitterpunkten in alle Raumrichtungen äquidistant und für die durch die Gitterknoten gebildeten Quadrate bzw. Quader gilt $\Delta x = \Delta y$ bzw. $\Delta x = \Delta y = \Delta z$.

3.6.3 Gittergenerierung in `iFluids`

Für die Gittergenerierung des uniformen Kartesischen Rechengitters kommt ein auf Baumdatenstrukturen basierender Algorithmus zum Einsatz (siehe Abbildung 3.2). Hierbei wurde von Wenisch [71] ein vollautomatischer Gittergenerierungsprozess umgesetzt.
Dieser transformiert die eingelesenen facettierten Geometrieinformationen in ein uniformes Kartesisches Gitter.
Hierzu wird zuerst ein leeres Gitter ohne Geometrieinformationen oder Randbedingungen erstellt. Zur richtigen und effizienten Abbildung der Geometrieinformationen und der dazu-

gehörigen Randbedingungen wird für jedes Dreieck[9] ausgehend vom Wurzeloktant (root octant) ein Oktant berechnet. Abhängig von Größe und Orientierung der Facette sind diese Oktanten für gewöhnlich eher klein. Die notwendige Tiefe des Oktalbaums wird durch die Größe des Wurzeloktanten und die Auflösung des Gitters bestimmt. Über den Oktalbaum kann nun bestimmt werden, in welcher bzw. in welchen Gitterzelle(n) eine Facette liegt.
Diese Gitterzellen, die sogenannten Voxel, erhalten eine Markierung je nachdem, ob sie entweder innerhalb, außerhalb oder genau auf der Begrenzung eines Hindernisses liegen. Sollte das Voxel auf der Berandung eines Objekts liegen, wird die für diesen Rand definierte Randbedingung ebenfalls mit dem Voxel abgespeichert. Das genaue Vorgehen der Gittergenerierung mit Hilfe eines Oktalbaums ist in [71] beschrieben.
Durch die effiziente Programmierung ist der Gittergenerator in der Lage, selbst aufwändige STL-Geometrien in wenigen Sekunden oder sogar Sekundenbruchteilen zu verarbeiten. Dies ist für die Anwendung in der hier beschriebenen CSE von essenzieller Bedeutung, da nur so ein interaktives Arbeiten mit der Simulation möglich ist.

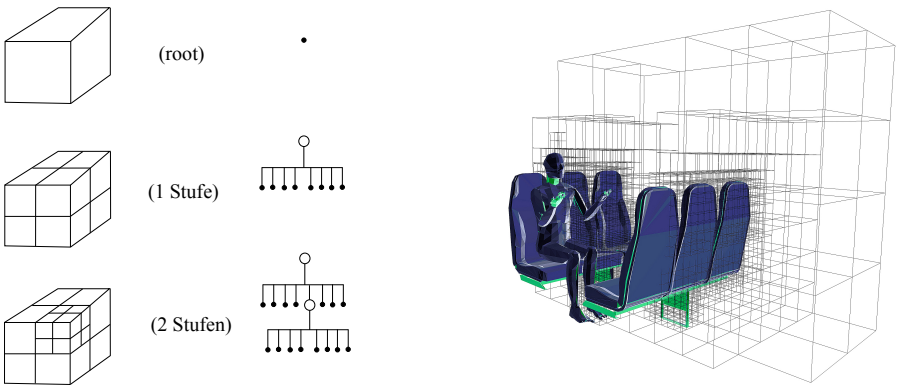


Abbildung 3.2: Visualisierung der Baumdatenstruktur zur Generierung des uniformen Kartesischen Rechengitters (nach [66]).

[9] Als Dateiformat für die geometrischen Objekte wir das STL-Format verwendet. In diesem Dateiformat werden die Oberflächen bereits über Dreiecksfacetten abgebildet.

Kapitel 4

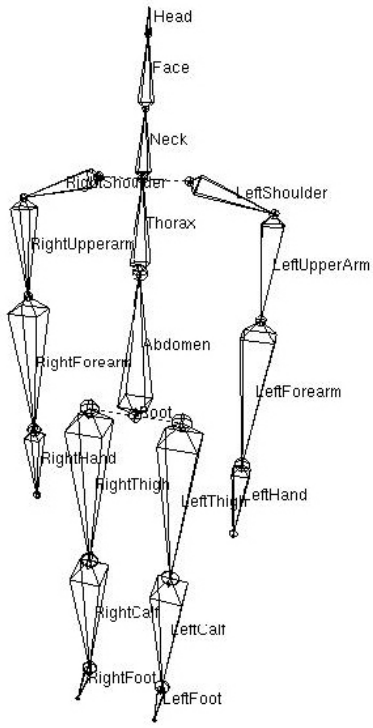
Implementierung eines parametrischen numerischen Dummys

Zur Vorbereitung der Ankopplung einen thermischen Manikins[1] wurde ein parametrischer numerischer Dummy in die CSE integriert. Hierbei wurde besonderer Wert auf die Kompatibilität mit dem Modell von Fiala [17] gelegt. Die Implementierung geschah im Rahmen des Projekts ComfSim durch die Beteiligten Yue, van Treeck und Pfaffinger [63].

4.1 Aufbau des parametrischen Dummys

Das Manikin ist in zehn Körperelemente unterteilt, wobei diese teilweise nochmals in Sektoren untergliedert sind (siehe nachfolgende Tabelle 4.1). Die Geometrie des Manikins wurde von Yue [77] in einem CAD-Programm konstruiert und dann in einem facettierten Format bereit gestellt, jeweils separat für die einzelnen Sektoren. Hierbei wurden verschiedene Detaillierungsgrade und Netzfeinheiten berücksichtigt. So liegt der Dummy einmal als vereinfachtes Zylindermodell und einmal als Modell mit ausgeformten Körperteilen vor, beides jeweils in einer groben, mittleren und feinen Triangulation der Oberfläche. Dies geschah, um ein flüssig benutzbares Modell auf Visualisierungsrechnern mit unterschiedlicher Leistungsfähigkeit zur Verfügung stellen zu können. Das entsprechende Modell kann vom Benutzer über Optionen in einem Kontrollfenster gewählt werden.
Zusätzlich wurde zum Facettennetz der Oberfläche des Manikins auch ein Skelettmodell des Dummys angelegt, das als Gerüst dient, auf welches die facettierte Oberfläche gelegt wird. Auf dem Skelettmodell sind die möglichen Bewegungen des Manikins und die sich daraus ableitenden Transformationsmatrizen für das Oberflächennetz definiert [63]. Das Skelettmodell wurde mit dem Programm *Blender* generiert.
Der numerische Dummy kann als Ganzes bewegt, skaliert oder rotiert werden. Hierbei verhält er sich wie ein anderes, gewöhnliches Geometrieobjekt innerhalb der CSE. Die Besonderheit dieses parametrischen Modells liegt jedoch darin, dass sich die Positionen der einzelnen Körperteile über die Modifikation eines Parameters ändern, d.h. „bewegen“ lassen; konkret wird ein Drehwinkel an den Gelenken vorgegeben. Hierbei werden die Abhängigkeiten der einzelnen Körperteile zueinander, wie sie durch das Skelettmodell vorgegeben sind, eingehalten.

[1] Manikin bezeichnet ein (CAD-)Modell eines Menschen.



Körperteil	Anzahl der Unterelemente	Namen der Unterelemente
Kopf	3	Stirn, Kopf, Unterseite
Gesicht	3	Gesicht, Oberseite, Unterseite
Nacken	4	Anterior, Posterior, Oberseite, Unterseite
Schultern	1	Schultern
Thorax	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Abdomen	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Oberarme	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Unterarme	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Hände	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Oberschenkel	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Unterschenkel	5	Anterior, Posterior, Inferior, Oberseite, Unterseite
Füße	4	Rist, Sohle, Oberseite, Unterseite

Abbildung 4.1: Körperteile des parametrischen Modells (siehe Yue [77]).

4.2 Implementierung des parametrischen Dummys

Das parametrische Manikin wurde im objektorientierten Sinne (siehe [76]) als Klassenstruktur implementiert. Hierbei dient eine einzelne Root-Klasse als Container für den „Körper“ des parametrischen Dummys. Für die Körperteile wurde ebenfalls eine Root-Klasse definiert, von der die einzelnen Körperteile – als member-Klasse der Körper-Klasse – abgeleitet sind. Ein Diagramm der Klassenstruktur ist in Abbildung 4.2 gegeben.
Die Oberflächennetze und Koordinaten der Knoten der einzelnen Körperteile sind wiederum in einer weiteren member-Klassen abgespeichert.

Um die Bewegungen der einzelnen Körperteile korrekt abzubilden, werden Zeiger zu einem übergeordneten „Knochen“ sowie zu bis zu drei nachgeordneten „Knochen“ gemäß dem entworfenen Skelettmodell angelegt. Des Weiteren werden hier lokal die nötigen Transformationsmatrizen vorgehalten.
Die Bewegungen der Körperteile werden durch Rotationen an den Gelenken erzeugt. Die dafür notwendigen Funktionen sind ebenso in den betreffenden Körperteil-Klassen angelegt und modifizieren die entsprechenden Transformationsmatrizen, d.h. die Transformationsmatrix des direkt bewegten Körperteils und auch die Transformationsmatrizen evtl. untergeord-

neter Körperteile.
Jeder Dummy wird zur Laufzeit unter Anwendung der vom Benutzer vorgegebenen Parameter, wie Modelltyp und Netzauflösung, dynamisch aus den im CAD-System erzeugten Einzelteilen zusammengesetzt. Diese Bausteine liegen in einem *LS-DYNA*-Format als Dateien in einer entsprechenden Ordnerstruktur im Installationsverzeichnis der CSE-Umgebung vor. Die Dateien mit den Oberflächennetzen der einzelnen Sektoren der Körperteile werden beim Erstellen der Klasse eingelesen und in der Klassenstruktur abgelegt.
Zur Kopplung an die bestehende CSE werden die Geometrieinformationen des gesamten Manikins inklusive der nötigen Transformationsmatrizen an die entsprechenden Importroutinen übergeben. Die facettierte Oberfläche wird dabei in das STL-Format umgewandelt und die Transformationsmatrizen kompatibel zu den Konventionen der CSE übergeben.

4.3 Implementation der animierten Bewegung

Vorerst nur für Visualisierungszwecke wurde aufbauend auf den parametrisierten Bewegungsmöglichkeiten des Dummy-Modells, wie in [78] beschrieben, eine animierte Bewegung integriert. Die Grundidee zur Implementation von komplexer Bewegung für das Dummy-Modell ist das Konzept der Interpolation zwischen sog. „key frames“ (siehe hierzu [78]). Üblicherweise genügt es bei der Anwendung dieser Technik, dass nur eine gewisse Anzahl dieser „key frames“ für einen Animationszyklus definiert werden muss. Im zeitlichen Ablauf kann dann zwischen zwei „key frames“ interpoliert werden, um eine gleichmäßige und flüssige Bewegungsanimation zu erreichen.
Es wurden zwei Arten von Bewegung für den parametrischen Dummy definiert: Gehen und Laufen. Für einen Animationszyklus für Gehen wurden sechs „key frames“ und für Laufen vier „key frames“ definiert. Jeweils zwischen zwei „key frames“ wird interpoliert, wobei ein kompletter Bewegungszyklus sowohl für die Lauf- als auch die Gehbewegung 30 Einzelschritte umfasst. Die Geschwindigkeit der Bewegung wird über einen Verzögerungsfaktor im Interpolationsalgorithmus kontrolliert. Bei der Aktivierung der Animation wird der Dummy zuerst in eine Ausgangsposition transformiert, von welcher aus die Animationssequenz startet.
Die Daten der „key frames“ und sämtliche Routinen der Animationen wurden ebenfalls in einer Klassenstruktur umgesetzt und mit der Root-Klasse des parametrischen Dummys verknüpft.

4.4 Integration des parametrischen Modells

Das in [77] entwickelte parametrische Manikinmodell wurde in die Visualisierungsapplikation integriert und mit den vorhandenen Geometrieschnittstellen verbunden. Hierbei werden die facettierten Oberflächen der Körperteile des Manikins an die CSE übergeben. Dadurch lässt sich das Manikin in verschiedenen Detaillierungsstufen und Oberflächennetzauflösungen interaktiv in eine laufende Simulation laden. Hierzu wurde eine Eingabemaske integriert (siehe Abbildung 4.3). Über diese Maske lässt sich auch die Körperhaltung des Manikins kontrollieren und verändern, beispielsweise eine Transformation von stehender zu sitzender Position. Neben der manuellen Bewegung der Gliedmaßen des Manikins durch den Anwender wurden auch Algorithmen implementiert, die ein Gehen bzw. Laufen des Manikins simulieren (nicht genutzt für die Simulation). Hierbei werden die nötigen Transformationen der Körperteile des

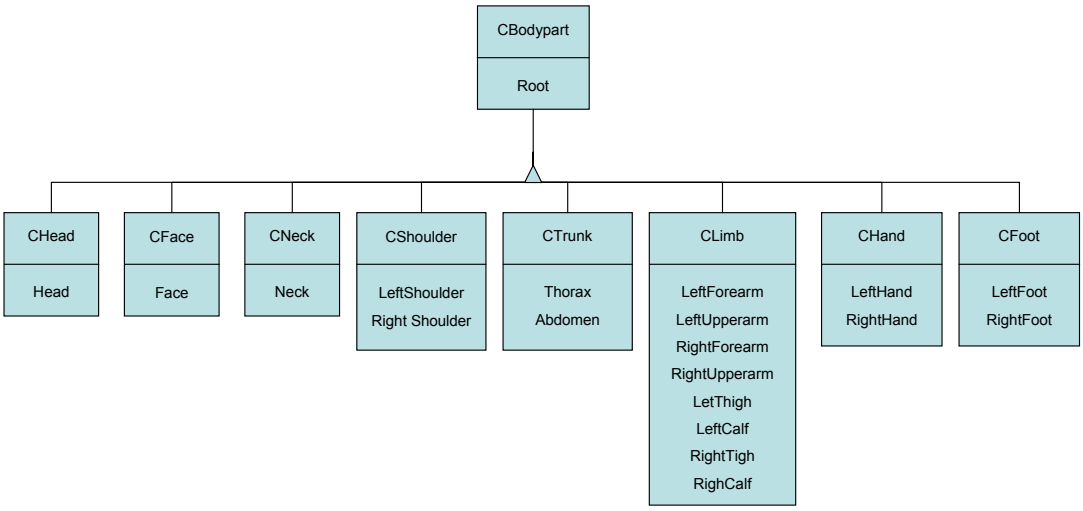


Abbildung 4.2: Klassenstruktur des parametrischen Modells.

Manikins automatisch generiert und die Transformationsmatrizen für die facettierten Geometrieobjekte der Körperteile automatisch in das System eingespeist. Die Funktionen hierfür wurden in eigene Threads ausgegliedert, um die gleichzeitige Darstellung der Bewegung und der Simulationsergebnisse im Ausgabefenster der Applikation nicht zu beeinträchtigen.

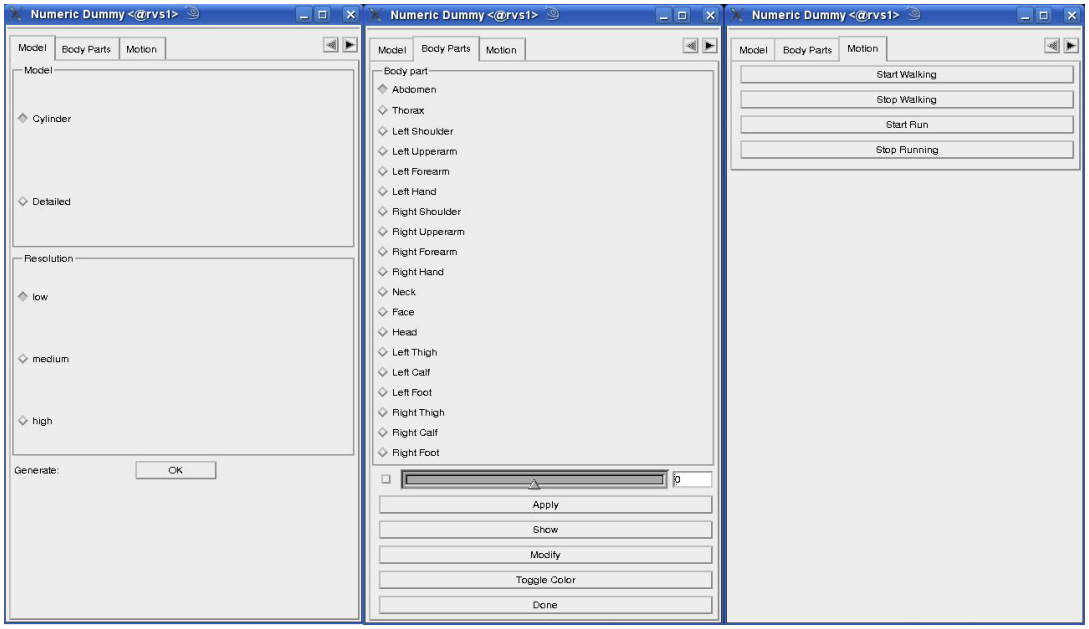


Abbildung 4.3: Eingabemaske zur Kontrolle des parametrischen Manikinmodells.

Abb. 4.4 zeigt ein im Laufmodus animiertes parametrisches Manikinmodell in einer `iFluids` CFD Simulation. Das Manikin wurde interaktiv auf einem Laufband platziert und anschließend die Laufbewegung initiiert. Die farbigen Stromlinien visualisieren die Umströmung der Objekte innerhalb der Simulation. Es sind drei *Frames* des Bewegungsalaufes des Animation Laufen

dargestellt. Zusätzlich wurden die Körperteile des Manikins farbig visualisiert. Die Möglichkeit der Einfärbung der Körperteile wurde zur Darstellung der Temperaturen eines anzukoppelnden Thermoregulationsmodells für den Dummy implementiert.

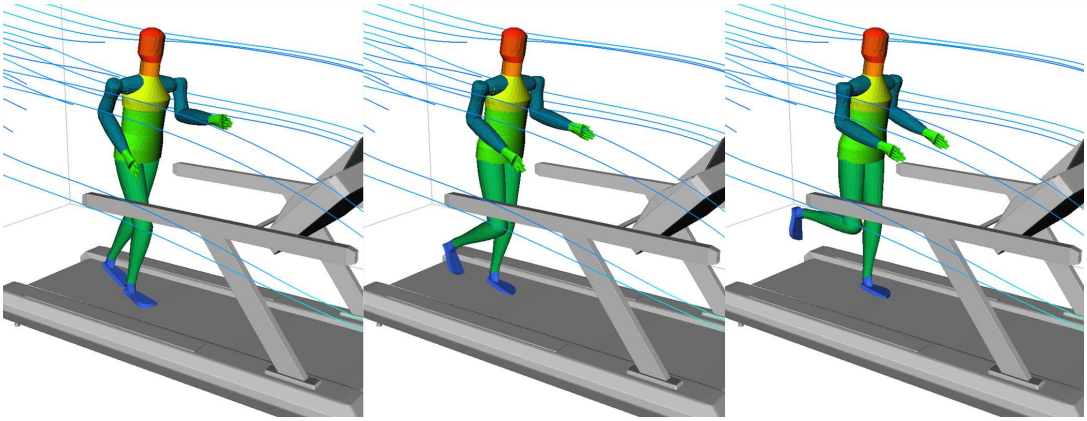


Abbildung 4.4: Animiertes parametrisches Manikinmodell innerhalb einer `iFluids` CFD Simulation. Es sind beispielhaft drei *Frames* des Laufzykluses dargestellt. Die Körperteile des Manikins wurden zur Veranschaulichung eingefärbt dargestellt.

Kapitel 5

Visualisierung in der interaktiven Strömungssimulation

Ein Kernbaustein dieser Arbeit, die Visualisierung in der interaktiven Strömungssimulation, wird nachfolgend untersucht. Ausgehend von einem Überblick über das Thema, wird auch auf die speziellen Anforderungen und Virtual-Reality (VR) Umgebungen eingegangen. Anschließend wird die Umsetzung in einer Forschungsumgebung, d.h. in `iFluids`, vorgestellt. Darauf folgt die Darstellung einer möglichen Umsetzung in industriellem Umfeld und die Evaluierung zusätzlicher Nutzungsmöglichkeiten des implementierten VR-Systems für Fremddaten.

5.1 Einführung in die wissenschaftliche Datenvisualisierung

5.1.1 Überblick

Die Geschichte der wissenschaftlichen Datenvisualisierung geht zurück bis in die Zeit der Röhrencomputer. Allerdings entstand erst mit zunehmender Verfügbarkeit von Rechenleistung und Hochleistungsrechnern in den 1980er Jahren ein steigender Bedarf nach Werkzeugen zur Analyse und Auswertung von durch Berechnungen und Simulationen entstandenen Daten, da durch die gestiegene Rechenleistung auch die produzierte Datenmenge enorm anstieg (siehe [2]). Vor allem in Anwendungen wie zum Beispiel der numerischen Strömungssimulation, Simulationen von komplexen Vorgängen in der Physik, Chemie oder Astonomie oder bildgebenden medizinischen Verfahren entstehen große Datenmengen, bei deren visueller Auswertung man auf leistungsfähige Algorithmen und Hardwareressourcen angewiesen ist.
Die Aufgabe der wissenschaftlichen Datenvisualisierung ist, dem Anwender Hilfsmittel bereit zu stellen, um die gewonnenen Daten analysieren und bewerten zu können, da das Ziel des wissenschaftlichen Rechnens „insight, not numbers“ ist ([20]). Hierbei ist eine grafische Darstellung von besonderem Nutzen, da das menschliche Gehirn komplexe Daten und Zusammenhänge so schneller begreifen kann (siehe [2]). Neben der visuellen Darstellung von Daten ist auch eine akustische oder haptische Aufbereitung von Daten möglich. Da diese jedoch für die Anwendungen, wie sie in dieser Arbeit beschrieben sind, nicht von Belang sind, wird auf diese Repräsentationen nicht weiter eingegangen.

5.1.2 Anforderungen an die wissenschaftliche Datenvisualisierung

Die wissenschaftliche Datenvisualisierung im Allgemeinen hat zum Ziel, dem Anwender einen vereinfachten Zugang zu großen und komplexen Datenmengen zu bieten, da die Auswertung der Zahlenfluten, wie sie zum Beispiel bei großen numerischen Simulationen entstehen, durch bloßes Betrachten ohne geeignete Hilfsmittel nicht mehr möglich ist.
Um diesem Ziel gerecht zu werden, müssen gewisse Grundvoraussetzungen erfüllt sein. Die Visualisierungsumgebung muss als Erstes für die darzustellenden Daten geeignet sein. Hier spielt beispielsweise die Leistungsfähigkeit der zur Verfügung stehenden Hardware eine Rolle. Des Weiteren muss die Darstellung der Daten klar, informativ und korrekt sein. Hierbei muss besondere Sorgfalt darauf verwendet werden, dass bei der Aufbereitung und Konvertierung der Rohdaten in eine andere Darstellungsform keine Fehler passieren und beispielsweise keine falschen oder irreführenden Muster entstehen. Außerdem sollten zur Visualisierung von bestimmten Ergebnissen mehrere verschiedene Darstellungstypen verfügbar sein, um die gewonnenen Daten hinsichtlich unterschiedlicher Aspekte analysieren zu können.

5.1.3 Interaktive wissenschaftliche Datenvisualisierung

Besonderheiten einer interaktiven Datenvisualisierung

Bei der Datenvisualisierung soll an dieser Stelle besonderes Augenmerk auf deren interaktive Formen gelegt werden. Zu den oben genannten Anforderungen kommen hier noch solche bezüglich des Antwortverhaltens des Systems hinzu. Zum Einen sollte das Einlesen und Konvertieren der Daten in die Simulationsumgebung möglichst schnell funktionieren, zum Anderen sollen sich Änderungen an den Parametern der Visualisierung möglichst schnell, im Idealfall in Echtzeit, auf die Datenrepräsentierung auswirken.

Statische Umgebungen

Von einer interaktiven statischen Visualisierungsumgebung spricht man, wenn die darzustellenden Daten zeitinvariant sind ([2]). Die Daten wurden durch einen externen Prozess generiert oder gemessen und werden anschließend in die Visualisierungsumgebung geladen. Die Daten können durch den Anwender in unterschiedlichen Weisen dargestellt und analysiert werden, wobei sich eine interaktive Änderung an der Art der Visualisierung oder an deren Parametern unmittelbar auf das dargestellte Bild auswirkt. Sämtliche Benutzeraktionen wirken sich jedoch nur auf die grafische Darstellung aus und haben keinen Einfluss auf die zu Grunde liegenden Daten.

Dynamische Umgebungen

Ausgehend von einer interaktiven statischen Visualisierungsumgebung wird diese bei einer interaktiven dynamischen Visualisierungsumgebung um die Veränderbarkeit der zu visualisierenden Daten ergänzt. Auch hier stammen die Daten in den meisten Fällen von externen Datenquellen, allerdings ist in diesem Fall eine direkte Anbindung der Quelle zur Visualisierung erforderlich. Die sich dynamisch verändernden Daten werden entweder kontinuierlich oder diskontinuierlich an die Visualisierungsumgebung übergeben. Der Benutzer kann sowohl

Parameter der Visualisierung als auch der zu Grunde liegenden Datengewinnung, z.B. der Simulation, direkt und möglichst intuitiv aus der Visualisierungsumgebung heraus ändern. Für eine solche interaktive dynamische Visualisierungsumgebung sind in der Literatur verschiedene Namen gebräuchlich; in dieser Arbeit wird nur der Begriff Computational Steering Umgebung verwendet.

5.1.4 Virtual-Reality Umgebungen

Verschiedene Verfahren zur Stereoprojektion

Unter Stereoprojektion versteht man Verfahren zur Darstellung von dreidimensionalen Informationen durch Projektion auf zweidimensionale Flächen. Hierbei werden für das rechte und linke Auge unterschiedliche Bilder projiziert ([10]).
Man unterscheidet primär nach aktiven und passiven Verfahren. Bei aktiven Verfahren werden Shutterbrillen eingesetzt, welche mit dem Display oder dem bildgebenden Rechner synchronisiert werden. Dieses Verfahren kann mit Projektoren und Monitoren eingesetzt werden. Hierbei werden durch den Projektor bzw. Monitor abwechselnd Bilder für das rechte und linke Auge gezeigt. Die Shutterbrille verdunkelt entsprechend das linke bzw. rechte Auge, so dass die dargestellten Bilder nur vom „richtigen“ Auge wahrgenommen werden. Hieraus ergibt sich, dass die eingesetzten Projektionsgeräte oder Monitore relativ hohe Bildwiederholraten beherrschen müssen, um ein flimmerfreies Bild zu gewährleisten, da die Projektionsgeräte eine doppelt so schnelle Bildwiederholrate wie effektiv gewünscht unterstützen müssen. Um beispielsweise in 50 Hz effektiv zu projizieren, muss der Projektor eine Darstellung in 2 * 50 = 100 Hz unterstützen. Flachbildschirme waren lange Zeit zu träge, um das schnelle Umschalten von rechtem und linkem Kanal zu ermöglichen. Erst seit kurzem sind Modelle verfügbar, die dies erlauben. Die Synchronisation zwischen Brille und Rechner geschieht meist über ein Infrarotsignal, wodurch ein direkter Blickkontakt zum entsprechenden Emitter nötig ist. Ein Vorteil dieser Technik bei Verwendung von Projektoren ist, dass nur ein Projektor für das Stereobild benötigt wird und auch jede beliebige Leinwand verwendet werden kann.
Bei passiven Stereoprojektionen kommen statt einer aktiv gesteuerten Brille solche mit Filterfolien oder -gläsern zum Einsatz. Am häufigsten werden Polarisationsfilter verwendet. Hierbei kommen zwei Projektoren zum Einsatz, welche jeweils mit Polarisationsfiltern, die um 90° versetzt angeordnet sind, versehen werden. Somit wird die Kanaltrennung durch Verwendung von polarisiertem Licht erreicht. In den Filterbrillen für den Anwender sind ebenfalls entsprechende Polarisationsfilterfolien vorhanden, wodurch sichergestellt wird, dass das polarisierte Licht für den rechten Stereokanal nur die Polarisationsfolie am rechten Auge passieren kann, wohingegen es durch den Filter am linken Auge geblockt wird und umgekehrt. Bei dieser Projektionsmethode muss der Betrachter seinen Kopf jedoch stets gerade halten und darf ihn nicht nach rechts oder links neigen, da das polarisierte Licht die Filterfolien aufgrund des gedrehten Winkels dann nicht mehr passieren kann. Ein weiterer Nachteil besteht darin, dass silberbeschichtete Leinwände zum Einsatz kommen müssen, da das Licht bei der Reflexion an weißen Leinwänden die Polarisation wieder verliert; dieses Problem tritt bei Auslegung als Rückprojektionsanlage systembedingt nicht auf. Ebenso sollte bedacht werden, dass die Projektoren durch den Einsatz der Filter einen Teil ihrer Helligkeit verlieren und somit entsprechend lichtstärker ausgelegt werden müssen. Als Vorteil gegenüber aktiven Stereoprojektionstechniken ist allerdings zu nennen, dass die Systeme aufgrund der einfacheren Komponenten in der Anschaffung deutlich

günstiger sind.
Neben der Verwendung von Polarisationsfiltern existieren auch noch andere Ansätze in der passiven Stereoprojektion, wie die Farbanaglyphenprojektion oder die KMQ-Projektion [1]. Bei der Farbanaglyphenprojektion werden statt der Polarisationsfilter Farbfilter, z.B. Rot und Grün oder Blau und Gelb, eingesetzt. Hierbei treten allerdings Farbverschiebungen auf. Bei der KMQ-Projektion werden die Bilder versetzt übereinander projiziert und der Anwender trägt eine Prismenbrille, welche die versetzt projizierten Bilder dem jeweils richtigen Auge zuordnet und den Eindruck schafft, die Bilder wären am gleichen Ort. Allerdings ist der Hauptnachteil dieser Verfahren, dass sich der Benutzer in einem genau definierten Punkt vor der Projektionsfläche aufhalten muss und diese Verfahren somit nur wenige Nutzer, typischerweise zwei bis drei, zulässt.

Autostereoskopische Displays

Als Sonderform der passiven Stereoprojektionstechniken ist der Bereich der sogenannten autostereoskopischen Displays zu nennen. Hierbei handelt es sich um TFT-Displays, die es ermöglichen, ohne Verwendung von speziellen Brillen ein 3D-Bild darzustellen. Diese sind systembedingt allerdings hauptsächlich für die Verwendung an Einzelarbeitsplätzen geeignet, da der Betrachtungswinkel für einen dreidimensionalen Eindruck sehr schmal ist und die Geräte in ihrer Bildschirmdiagonale handelsüblichen TFT-Monitoren entsprechen.

Komponenten einer VR-Projektion

Als Grundlage für die in dieser Arbeit vorgestellten Forschungen kam hauptsächlich eine passive Stereoprojektion zum Einsatz, weshalb ein solches Setup an dieser Stelle näher analysiert werden soll.
Als erste Komponente ist der Rechner zu nennen, der für die Visualisierung zuständig ist. Da nur ein passives Setup betrachtet wird, entfällt hier die Anforderung zur Synchronisation von Shutterbrillen. Soll die Visualisierung von beiden Stereokanälen auf dem gleichen Rechner erfolgen, so sind entweder eine Grafikkarte mit zwei Videoausgängen oder zwei Grafikkarten erforderlich. Um ein flüssiges Arbeiten zu gewährleisten, kann eine Workstation-Grafikkarte mit guter OpenGL-Beschleunigung zum Einsatz kommen (zum Beispiel Nvidia Quadro Serie).
Als Nächstes sind die eingesetzten Projektoren zu nennen. Da hier für jeden Kanal ein eigener Projektor verwendet wird, sind die Anforderungen verglichen mit aktiven Verfahren gering, so dass ein handelsüblicher Datenprojektor Anwendung finden kann. Die beiden Geräte werden übereinander auf einem Gestell fixiert und so justiert, dass sie ihr Bild deckungsgleich und unter Ausgleich von Bildverzerrungen auf die Projektionsfläche werfen (siehe Abbildung 5.1). Zur Trennung der beiden Stereokanäle wird direkt vor die Objektive der beiden Beamer je ein Polarisationsfilter, 90° verdreht zu einander, montiert. Als Projektionsfläche dient bei normaler Projektion eine silberbeschichtete Leinwand, da nur diese das polarisierte Licht unverändert zurück wirft. Alternativ kann auch eine Rückprojektionswand verwendet werden, allerdings erfordert diese einen entsprechend großen Raum hinter der Projektionsfläche.
Zur Benutzerinteraktion können die Eingabegeräte des Visualisierungsrechners verwendet werden. Allerdings gestaltet sich die Interaktion mit einer dreidimensionalen Szene mit Hilfe einer

[1] benannt nach den Erfindern Koschnitzke, Mehnert und Quick

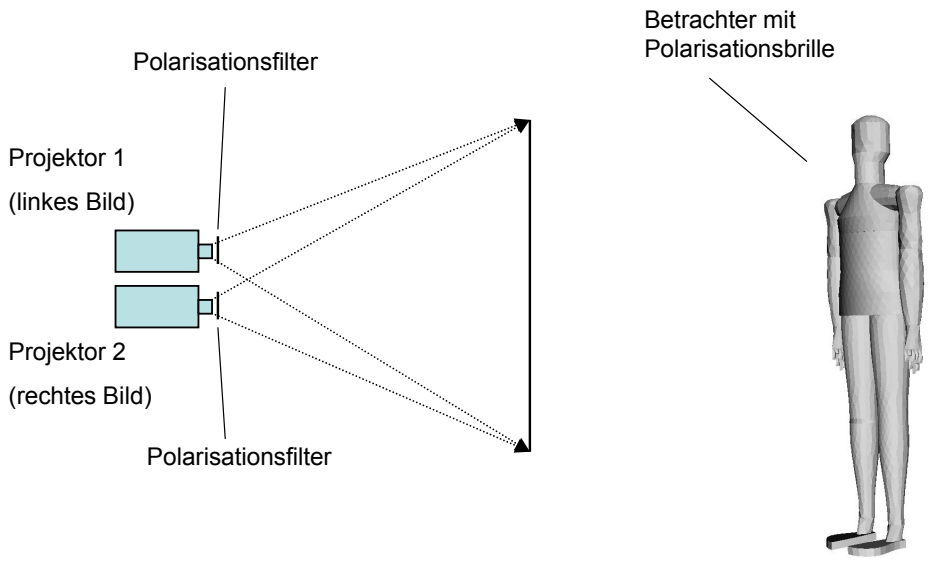


Abbildung 5.1: Schematische Darstellung der Komponenten einer passiven Stereoprojektion.

herkömmlichen Maus oft schwierig. Zusammen mit der Polarisationsbrille für den Anwender ist das Setup für eine einfache Anlage komplett.
Dieses Setup lässt sich zur besseren Benutzbarkeit und für einen besseren Stereoeindruck noch modifizieren und ergänzen. Um eine bessere Interaktion mit der Szenerie zu ermöglichen, sollte ein Eingabegerät, welches für die Verwendung in 3D-Umgebungen konzipiert wurde, eingesetzt werden. Hier bietet sich entweder eine 3D-Maus („Space Mouse“), ein Datenhandschuh oder ein „Stylus“ an. Datenhandschuh oder „Stylus“ erfordern ein Tracking-System, welches die Positionierung der Eingabegeräte im Raum zulässt. Ein solches Tracking-System wird auch benutzt, um die Kopfposition des Nutzers zu bestimmen und die virtuelle Kameraposition innerhalb der 3D-Szene entsprechend anzupassen. Head-Tracking liefert einen besonders guten Stereoeindruck für den Benutzer, dessen Kopf oder Stereobrille verfolgt wird, da sich durch Drehen des Kopfes oder Bewegung der Person durch den Raum die Perspektive ändert. Allerdings leidet darunter der Stereoeindruck für andere Betrachter der gleichen Projektion, da ein Head-Tracking nur für einen Benutzer möglich ist. Zudem ist Head-Tracking meist nur bei aktiven Stereoprojektionen sinnvoll, da sich die Limitationen der passiven Stereoprojektionen, v.a. hinsichtlich der Bewegungsfreiheit des Nutzers, hier kontraproduktiv auswirken.

5.2 Anwendung in der Forschung: Datenvisualisierung im interaktiven Strömungssimulator `iFluids`

5.2.1 Verwendete Software-Bibliotheken

OpenInventor

Im interaktiven Strömungssimulationsprogramm `iFluids` werden verschiedene Softwarebibliotheken eingesetzt. Bei der visuellen Darstellung der Geometrie und der Berechnungsergebnisse

kommt die Software Open Inventor von der Firma vsg[2] zum Einsatz. Diese kommerzielle, objektorientierte Softwarebibliothek ermöglicht eine leistungsstarke Visualisierung der Geometrie und bietet ausgefeilte Optionen zur Darstellung von Strömungsdaten. Die Ergebnisvisualisierung wird über die MeshViz Erweiterung des Open Inventor realisiert. Ein weiterer Vorteil der eingesetzten Software besteht in der umfangreichen Unterstützung von Virtual-Reality-Umgebungen und von verschiedenen Hardwareplattformen.

Orbacus Corba

Ein weiterer Softwarebaustein des `iFluids` Pakets ist die objektorientierte Middleware CORBA[3]. Diese kommt in der kooperativen Version von `iFluids` ([4]) und in der speziellen Implementierung im Siemens Medialab (siehe Abschnitt 5.3) zum Einsatz. Hierbei handelt es sich um eine durch die Object Managment Group (OMG) standardisierte Middleware, die eine plattformübergreifende Kommunikation zwischen verschiedenen Prozessen ermöglicht. Ein Kernmerkmal von CORBA ist die Trennung von Schnittstelle und Implementierung. Die Schnittstelle wird in einer standardisierten Beschreibungssprache (OMG IDL[4]) plattformunabhängig definiert, welche dann automatisch in plattformspezifischen Code umgewandelt wird. Es stehen Mappings in verschiedenste Sprachen, u.a. C, C++, Java oder Python, zur Verfügung. Dieser plattformspezifische Code kann dann direkt auf der Clientseite verwendet werden und wird auf Serverseite um die entsprechenden Funktionen und Objekte ergänzt. Der Programmierer sieht nur die definierten Schnittstellen; die Kommunikationsaufgaben werden durch CORBA Routinen abgewickelt. Da die Schnittstellendefinition sehr strikt ist und die einzelnen Softwareobjekte nur die Schnittstelle und nicht den dahinter stehenden Code sehen, können nicht nur verschiedene Plattformen, sondern auch verschiedene Implementationen des CORBA Standards miteinander kommunizieren.

5.2.2 Unterstützte Hardware und Plattformen

Desktop-Systeme

Die implementierte Visualisierungskomponente kann auf verschiedenen Hardware- und Softwareplattformen eingesetzt werden. Als einfachste Plattform sind zuerst handelsübliche Desktopsysteme zu nennen. Hierbei werden für die Visualisierung sowohl Microsoft Windows als auch Linux Systeme unterstützt. Wird ein Windows System zur Visualisierung verwendet, so muss die CORBA basierte kooperative Version der Visualisierungsapplikation eingesetzt werden, da es hier zu einer plattformübergreifenden Kommunikation zwischen der Visualisierungsplattform und der Linux basierten Simulationsplattform kommt. Diese plattformübergreifende Kommunikation wird von Standard MPI Implementationen nicht unterstützt. Im einfachen Desktopfall kommt nur eine 2D-Visualisierung zum Einsatz. Sie dient vornehmlich zur Unterstützung bei der Modellierung des Simulationsgebietes.
Ist am entsprechenden Desktop-System 3D-taugliche Hardware vorhanden (siehe Abbildung 5.2), d.h. beispielsweise eine entsprechende Grafikkarte, eine Shutterbrille mit Zubehör und ein Monitor mit ausreichender Bildwiederholungsfrequenz, wird auch hier die Darstellung des

[2] früher tgs/Mercury
[3] Common Object Request Broker Architecture
[4] Interface Definition Language

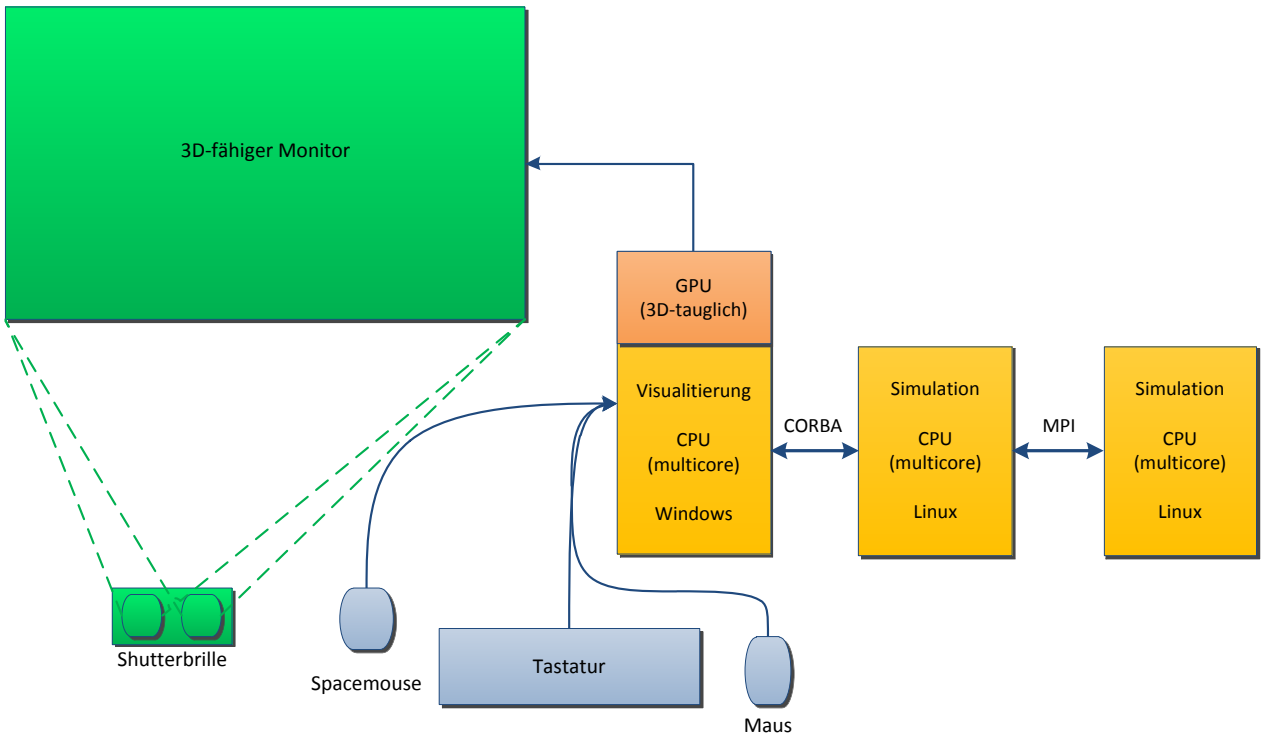


Abbildung 5.2: Schematische Darstellung der Hardwarekomponenten eines Desktop-Systems von `iFluids`.

Simulationsgebietes und der Ergebnisse in 3D unterstützt. Die Unterstützung für Standard-Hardware wird direkt über entsprechende Funktionen der OpenInventor-Bibliothek realisiert.

VR-Systeme

Für dedizierte Virtual-Reality Umgebungen wurden zusätzlich zu den bereits in der OpenInventor-Bibliothek vorhanden 3D-Optionen weitere, für spezielle Konfigurationen optimierte Darstellungen hinzugefügt. Als Beispielkonfiguration für eine 3D-Virtual-Reality-Umgebung dient die lehrstuhleigene Installation einer 3D-Rückprojektionsanlage. Hierbei handelt es sich um die Installation eines Passivstereo-Systems mit einer Rückprojektionsleinwand. Es kommen zwei DLP-Projektoren mit linearen Polarisationsfiltern zum Einsatz, die ein sich auf der Leinwand überlappendes Bild liefern. An weiterer Hardware kommt eine Dual-Opteron Workstation mit 4 GB Arbeitsspeicher und einer Grafikkarte der Workstationklasse zum Einsatz. Die Grafikkarte verfügt über zwei Videoausgänge, welche mit den beiden Projektoren verbunden sind. Als Betriebssystem kommt eine openSuse Linuxdistribution zur Anwendung, und die grafische Oberfläche ist so konfiguriert, dass die Arbeitsfläche die doppelte Breite der Auflösung der Projektoren in horizontaler Richtung hat und hälftig von je einem Projektor dargestellt wird.
Neben dem Einsatz auf VR-Systemen mit einer Projektionswand ist auch die Verwendung spezieller Konfigurationen, beispielsweise einer Holobench, wie sie im LRZ installiert ist, denkbar. Hierbei handelt es sich um eine im Schnitt L-förmige Virtual-Reality-Umgebung mit zwei Projektionsflächen für Tisch und Wand (siehe Abbildung 5.3), welche zusätzlich noch mit speziellen 3D-Eingabegeräten und Head-Tracking ausgestattet ist.

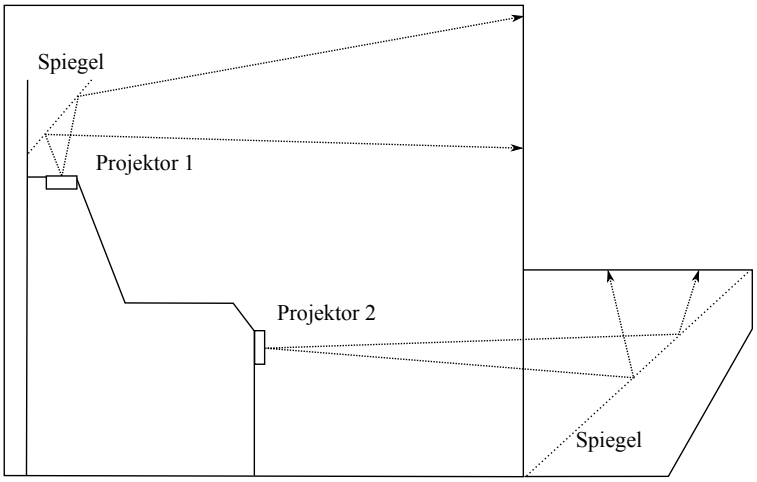


Abbildung 5.3: Schematische Darstellung des Aufbaus der Holobench am LRZ.

Remote Visualization Systeme

Eine weitere Anwendungsplattform sind Remote-Visualization-Systeme, wie sie etwa am LRZ (mit den sogenannten Maschinen rvs1 oder gvs1 und gvs2[5]) zur Verfügung stehen. Hierbei wird die rechenintensive Visualisierung der Daten nicht auf dem lokalen Rechner des Anwenders vorgenommen, sondern auf einen für diese Aufgaben spezialisierten Rechner mit besonders leistungsfähiger Grafikhardware ausgelagert. Dort erfolgt das Rendering der auf dem Monitor des Anwenders darzustellenden Szene, welche dann als einfaches 2D-Bitmap komprimiert an den lokalen Rechner über ein Netzwerk übertragen wird und dort mit wenig Rechenaufwand dargestellt werden kann. Ein solches System gestattet dem Anwender die Visualisierung auch von komplexen, hochauflösenden Simulationen auf Rechnern mit einfacher Grafikhardware. Ein weiterer Vorteil einer solchen Konfiguration ist, dass der Benutzer für seinen lokalen Arbeitsplatz keine besonders schnelle Anbindung an den Simulationsrechner benötigt. Diese ist nur zwischen Simulationsrechner und Remote-Visualization-Rechner (RV) erforderlich, da nur hier die Berechnungsergebnisse mit hohem Durchsatz übertragen werden. Zwischen dem RV-System und dem lokalen System genügt auch eine langsamere Anbindung, im Extremfall auch nur ADSL.
Gerade für die in dieser Arbeit beschriebene CSE bietet der Remote-Visualization-Ansatz Vorteile. Da es sich bei der Strömungssimulation um eine rechenintensive Aufgabe handelt, bei welcher auch große Datenmengen anfallen, sind sowohl für Simulation als auch Visualisierung leistungsstarke Rechner notwendig. Durch den RV-Ansatz können diese Rechner zentral vorgehalten und von mehreren Anwendern lokal am Arbeitsplatz genutzt werden.
Auf der im Kontext dieser Arbeit genutzten RV-Maschine rvs1 kommt die *Shared Visualisation Software* der Firma Sun zum Einsatz. Diese basiert hauptsächlich auf den beiden Open Source Projekten VirtualGL und TurboVNC (siehe [39]).
Die Remote Visualisierung kann auf zwei unterschiedliche Arten genutzt werden, nämlich in einem sogenannten *Direct* Modus und in einem sogenannten *Raw* Modus.
Beim *Direct* Modus kommt nur VirtualGL zum Einsatz, jedoch wird auch auf der Client Seite

[5]Siehe auch `http://www.lrz.de/services/compute/visualisation/`

ein X-Server[6] benötigt, da es sich beim Rechner rvs1 um eine Linux Maschine handelt.
In diesem Fall werden die 3D Grafikbefehle, und nur diese, auf dem RV-System und somit auf der dort installierten leistungsfähigen Grafikhardware ausgeführt. Alle 2D Grafikbefehle, wie sie zum Beispiel für die Darstellung der Steuerelemente des Fensters der Benutzeroberfläche der CSE benötigt werden, werden so wie bei Standard X11-Tunnel-Verbindungen über *Secure Shell* (ssh) übertragen. Die 3D Befehle werden auf der Grafikhardware des RV-Rechners gerendert. Die dabei entstehenden 2D Bilder werden anschließend mittels TurboJPEG komprimiert und über VirtualGL an den Client übertragen. Der dort laufende VirtualGL Client-Prozess empfängt die Daten, entpackt sie und kopiert sie in das vom lokal laufenden X-Server generierte Fenster. Vorteile dieser Betriebsart ist die höhere Geschwindigkeit bei großen Bandbreiten zwischen Client und RV-Server, sowie die Eigenschaft, dass nur der 3D-Inhalt komprimiert und übertragen wird. Somit werden 2D Menüs und Text der Benutzeroberfläche über den lokalen X-Server dargestellt, was zu einer optimalen, scharfen Darstellung führt. Nachteil, v.a. für Windows Client-Systeme, ist jedoch der zwingend benötigte X-Server.
Beim *Raw* Modus, welcher auch als *X11 image transport* bezeichnet wird, wird auf der Clientseite nur der TurboVNC Client benötigt.
Diese Betriebsart eignet sich vor allem für Client Systeme, auf denen kein X-Server verfügbar ist, d.h. hauptsächlich Microsoft Windows-Systeme. Hierbei läuft auf dem Serversystem ein zweiter X-Server, genauer gesagt der VNC-Server. Dieser überträgt den Inhalt des virtuellen Bildschirms TurboJPEG komprimiert an die Clientapplikation, welche lokal auf dem Rechner des Anwenders läuft. Der Unterschied zwischen einem normalen VNC-Server und dem TurboVNC-Server besteht darin, dass letzterer die hardwarebeschleunigte Ausführung von 3D Grafikbefehlen erlaubt. Zusätzlich verfügt TurboVNC über einen andere Komprimierungsalgorithmus, welcher besser für 3D Grafik geeignet ist. Ein Nachteil dieser Methode ist jedoch, dass der gesamte virtuelle Desktop komprimiert und übertragen wird. Dies bedeutet, dass auch der Text und die Fenster der Benutzeroberfläche einer Applikation auf dem Remote Visualization System erzeugt werden. Somit kann es auch bei diesen zu sichtbaren Artefakten aufgrund der JPEG-Komprimierung kommen.
Die Remote-Visualisierung, vor allem im *Direct* Modus, eignet sich auch zur Kombination mit stereoskopischen Projektionsverfahren auf Clientseite. Für die Nutzung in einer VR-Umgebung muss jedoch eine Passivstereokonfiguration gewählt werden, bei welcher der rechte und linke Kanal separat vom RV-System zur lokalen Projektion übertragen und dort dargestellt werden.
In Abbildung 5.4 ist schematisch die Visualisierungspipeline für die beiden möglichen Betriebsarten dargestellt.
Die Hardwarebasis der rvs1 stellt ein Sun x4600 Server dar. Er verfügt über 16 Prozessorkerne (acht Dualkern Opteron CPUs) und insgesamt 128 GB Arbeitsspeicher. Als Betriebssystem kommt der SuSE Linux Enterprise Server (SLES) 10.2 zum Einsatz. Die Grafikleistung wird von vier Nvidia Quadro FX5500 Grafikkarten (in zwei Nvidia Quadroplex Einheiten) mit je 1 GB Grafikspeicher zur Verfügung gestellt, so dass maximal vier Anwender parallel auf der Maschine arbeiten können. Sollte ein Anwender mehr als eine Grafikkarte benötigen, reduziert sich die Anzahl der maximalen Nutzer entsprechend. Die Speicherbandbreite der Grafikkarten beträgt 33,6 GB/s und es können maximal 225 Millionen Dreiecke pro Sekunde verarbeitet werden. Die Grafikkarten sind mit den CPUs über 16 PCIexpress Kanäle verbunden.

[6] Bei Nutzung eines Windows Rechners als Client System ist zwingend der X-Server Exceed der Firma *Open Text*, vormals *Humingbird*, einzusetzen.

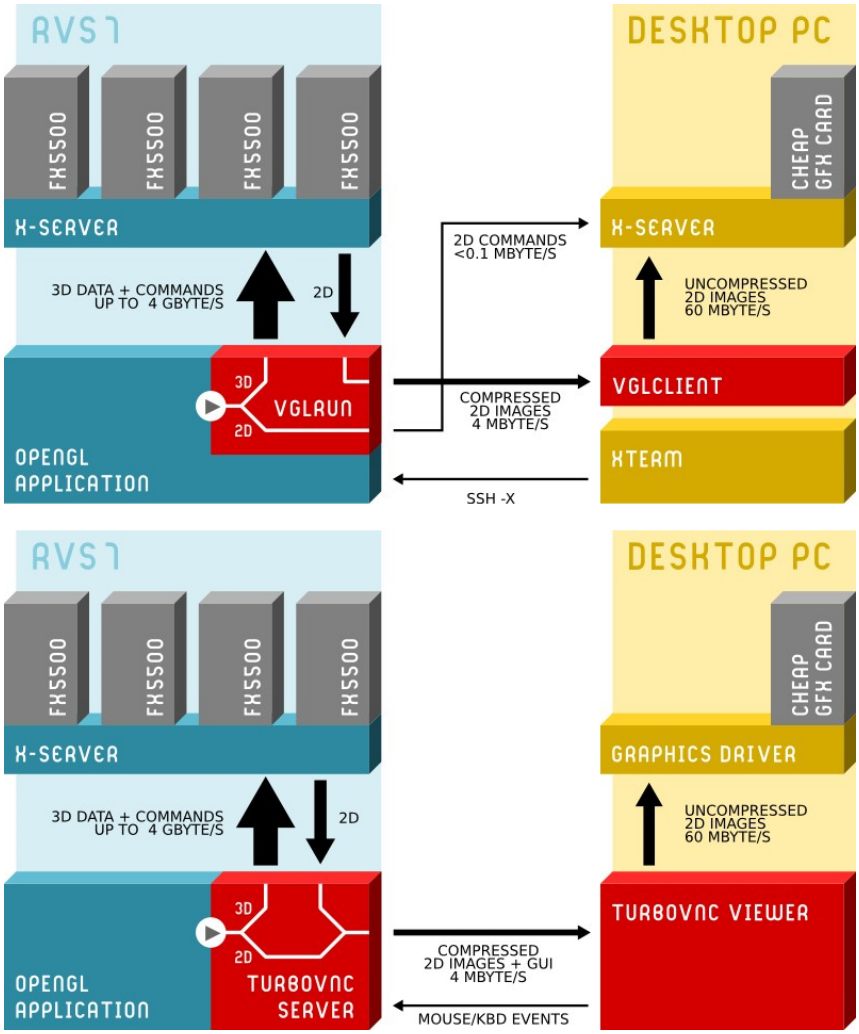


Abbildung 5.4: Schematische Darstellung der Visualisierungspipeline für den *Direct* Modus (links) und den *Raw* Modus (rechts) (Quelle: LRZ).

Die rvs1 befindet sich im gleichen Netzwerksegment wie der HLRB II, das bedeutet, dass ein direkter gegenseitiger Zugriff möglich ist. Die Netzwerkanbindung erfolgt über eine 10 GBit/s Ethernet Verbindung.
Zur Koordination der Nutzung des Systems durch unterschiedliche Anwender kommt ein Reservierungssystem zum Einsatz, in welchem der Anwender unter Angabe der Anzahl der benötigten Grafikkarten, der Nutzungsdauer und des Startzeitpunkts Rechenzeit auf der Maschine reservieren muss.
Anders als die rvs1, die für Nutzer des HLRB II reserviert ist und auch nur von in der HLRB II Firewall freigeschalteten Rechnern nutzbar ist, stehen die Systems gvs1 und gvs2 einer breiteren Nutzergruppe zur Verfügung und sind auch weltweit erreichbar.
Die RV-Server gvs1 und gvs2 basieren ebenfalls auf einem Sun x4600 Server, allerdings mit 32 statt 16 Prozessorkernen (acht Vierkern Opteron CPUs) und der doppelten Menge an Arbeitsspeicher, d.h. 256 GB pro System. Als Betriebssystem wird ebenfalls SLES 10.2 verwendet. Jedes der beiden Systeme besitzt ebenso vier Grafikkarten, jedoch kommen hier Nvidia Quadro FX5800 Karten mit je 4 GB Grafikspeicher zum Einsatz. Die verbauten Grafikkarten

sind CUDA[7]-fähig. Auch diese beiden Server sind über eine 10 GB/s Ethernet Verbindung an das LRZ Netzwerk angeschlossen.

5.2.3 Software-Layout der CSE

Die `iFluids` CSE liegt in zwei Versionen vor, einer Einzelplatz- und einer Mehrplatzapplikation. Beide verfügen hinsichtlich der Visualisierungsfunktionen über die gleichen Eigenschaften, unterscheiden sich aber in der Form der Kommunikation und teilweise in der Bedienung.

Einzelplatz-Applikation

Bei der Einzelplatzapplikation besteht das CSE aus zwei miteinander gekoppelten Applikationen, der Visualisierungsapplikation und dem Simulationskern mit Gittergenerator. Die Kommunikation erfolgt bei dieser Version über MPI (Message Passing Interface).
Die Applikation wurde in objektorientierter Programmierweise erstellt und basiert auf einer Klassenstruktur. Der Code gliedert sich grundlegend in drei Teile: Einen Setupteil, der sowohl die Kerndaten der Simulation als auch die Geometrie und die Randbedingungen enthält, einen Displayteil, der die für den Benutzer sichtbare grafische Oberfläche, inklusive der Visualisierung der Geometrien und der Simulationsergebnisse enthält und einen Kommunikationsteil, in dem der Datentransfer zwischen Visualisierung und Simulation abgewickelt wird (siehe Abbildung 5.5).

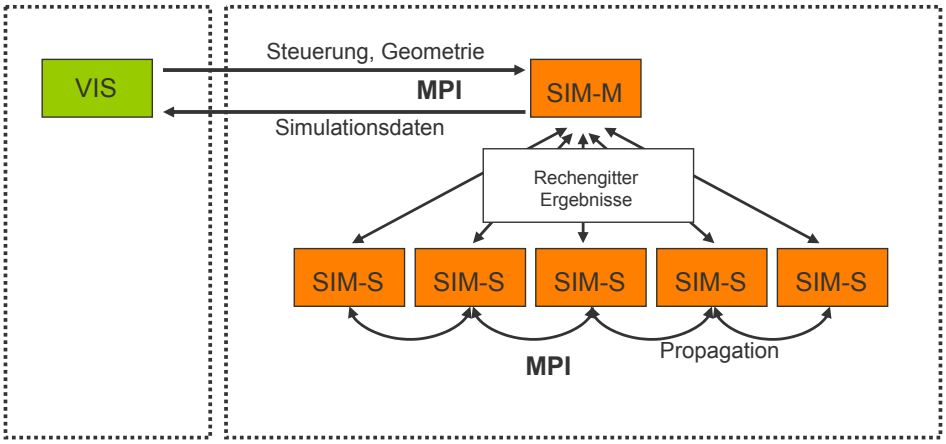


Abbildung 5.5: Schematische Darstellung der Komponenten der Einzelplatz-Version der CSE (nach Wenisch [72].

Durch die Kapselung der verschiedenen Programmkomponenten ist es möglich, sie durch andere Komponenten zu ersetzen. Dies wurde im vorliegenden Code für die Kommunikation umgesetzt. Die Kommunikationsfunktionen werden über eine Interfaceklasse aufgerufen, welche den Einsatz von MPI für die Einzelplatz-Applikation und von CORBA für die Mehrplatz-Applikation erlaubt, ohne dass andere Teile der Visualisierung von der jeweils verwendeten Kommunikations-Bibliothek beeinflusst werden. Prinzipiell ist auch ein Tausch der Display-Komponente möglich, doch wurde dies mangels Bedarf nicht umgesetzt.

[7] Compute Unified Device Architecture.

Mehrplatz-Applikation

Neben der Einzelplatzversion wurde zusätzlich von Borrmann [6] auch eine zweite kollaborative Version, die auf den selben Grundfunktionen aufbaut, implementiert. In dieser Version des CSE wurde die Kommunikation zwischen Visualisierungs-Frontend und Simulation mittels CORBA realisiert. Um den Simulationskern unangetastet zu lassen und da der Simulationskern zum Zeitpunkt der Entwicklung als reiner C-Code implementiert war, wurde hier kein CORBA-Interface implementiert. Dies machte bei der Übertragung der Daten von und zum Simulationskern einen Zwischenschritt notwendig. Es wurde eine Serverapplikation eingefügt, welche als so genannter Simulationsserver fungiert. Dieser stellt das MPI-Interface für die Simulations-Applikation bereit.
Da der Hauptzweck dieses Zweigs der CSE eine gemeinsame Arbeit am selben Simulation-Setup war, wurde neben dem Simulationsserver auch noch ein so genannter Geometrieserver eingeführt, welcher die Zusammenarbeit zwischen den einzelnen Benutzern und den optionalen mehreren Simulationsservern koordinierte. Am Geometrieserver werden alle Visualisierungsclients registriert und er hält sowohl das Simulationssetup als auch die Geometrieobjekte zusammen mit deren Randbedingungen zentral für alle verbundenen Visualisierungsclients und Simulationsapplikationen bereit. Werden von einem Benutzer Änderungen vorgenommen, werden diese automatisch an alle Clients, Visualisierung und Simulation verteilt. Während alle Änderungen am Setup zentral über den Geometrieserver verteilt werden, erfolgt die Übermittlung der Simulationsergebnisse direkt von einem Simulationsserver an den konnektierten Visualisierungsclient; auch hierfür kommt CORBA zum Einsatz. Dies ist von Vorteil, wenn Anwender beispielsweise an verschiedenen Standorten am selben Modell arbeiten wollen, aber die Netzwerkverbindung nicht schnell genug ist, um die Simulationsergebnisse zwischen den Standorten zu übertragen. In diesem Fall werden nur die Geometrie- und Setupinformationen vom und zum Geometrieserver zwischen den Standorten übertragen, die Simulation hingegen läuft an jedem Standort separat lokal mit einem eigenen Simulationsserver. Eine schematische Skizze ist in Abbildung 5.6 gegeben.

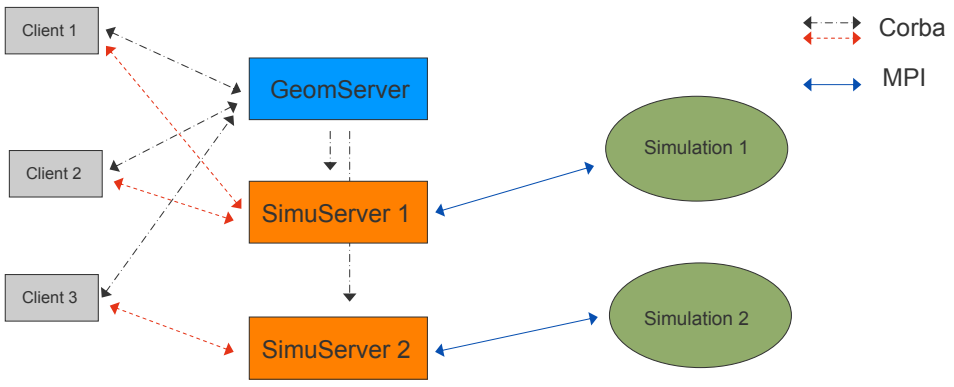


Abbildung 5.6: Schematische Darstellung der Komponenten der kooperativen CSE (nach Borrmann [6]).

Im Visualisierungs-Client kommt prinzipiell derselbe Display-Teil wie in der Einzelplatz-Applikation zum Einsatz. Einzig die Bedienerführung wurde um Funktionen zur Steuerung der kooperativen Umgebung erweitert. Der Setup-Teil ist ebenso identisch mit der Einzelplatz-Applikation. Unterschiedlich ist jedoch der Kommunikationsteil, da hier MPI durch CORBA

ersetzt wurde.

Benutzerinterface

Beim Benutzerinterface kommen Standardelemente der verwendeten OpenInventor-Bibliothek zum Einsatz. Hier wird auch zusätzlich zwischen der Einzelplatz- und der Mehrplatz-Applikation unterschieden. Während bei Ersterer ein 3D-Menü, welches in die laufende Simulation eingeblendet wird, Anwendung findet, kommt bei Letzterer eine normale Menüleiste zum Einsatz. Zusätzlich stehen auch Tastaturbefehle für häufig genutzte Funktionen, wie zum Beispiel das Einblenden von Stromlinien, zur Verfügung.
In Abbildung 5.7 ist das Standardfenster der Einzelplatz-Applikation zu sehen. Zu erkennen sind hier die durch die OpenInventor-Bibliothek vorgegeben Bedienelemente zur Navigation innerhalb der Szene, beispielsweise Zoom und Rotationen. Die Funktionen der CSE sind über das Menü, wie in Abbildung 5.8 dargestellt, zu bedienen.

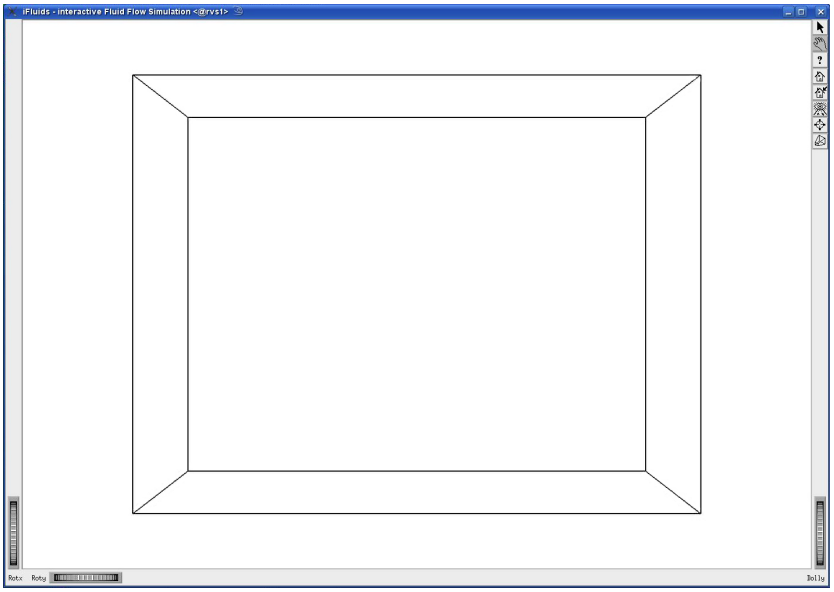


Abbildung 5.7: GUI der Einzelplatz-Applikation mit den Standard Bedienelementen der OpenInvertor-Bibliothek. Die dargestellte Box stellt die äußere Berandung des Strömungsgebiets dar.

Mit der Maus oder einem speziellen 3D-Eingabegerät, wie zum Beispiel eine Space-Mouse, lässt sich einerseits die Szene verschieben, rotieren oder skalieren, andererseits lassen sich so auch Geometrieobjekte und Elemente zur Strömungsvisualisierung platzieren und bearbeiten.

5.2.4 Implementierte Visualisierungstechniken

Strömungsdaten

Zur Visualisierung von Strömungsdaten wird auf die MeshViz-Erweiterung der OpenInventor-Bibliotheken zurückgegriffen. Zur Darstellung der x-, y- und z-Geschwindigkeitskomponenten und des Drucks stehen sowohl Iso-Flächen als auch Schnittebenen zur Verfügung. Letztere

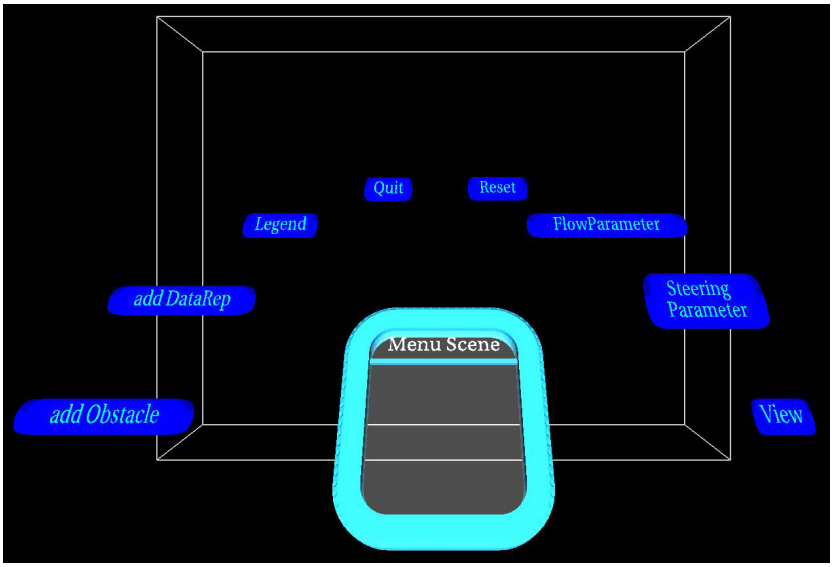


Abbildung 5.8: Menü der Einzelplatz-Applikation.

können beliebig im Raum angeordnet und bewegt sowie rotiert werden. Die Größe des darzustellenden Werts wird über die Farbgebung wiedergegeben, welche über eine einblendbare Farbskala quantifiziert wird. Als weitere Visualisierungsoption für Geschwindigkeiten stehen Vektorebenen zur Verfügung, welche ebenfalls frei im Raum angeordnet werden können und Richtung und Betrag der Geschwindigkeiten über Pfeile in variabler Länge repräsentieren. Ebenso zur Verfolgung der Strömung dienen Stromlinien und Strompunkte, für die als Quellen Stäbe, Rechtecke und Kreise in unterschiedlichen Größen frei im Strömungsgebiet platziert werden können. Die Geschwindigkeit der Strömung wird auch hier über die Farbgebung der Linien oder Punkte kodiert.

Thermische Daten

Thermische Daten, welche in der Simulation gewonnen wurden, können ebenfalls visuell dargestellt werden. Hierbei ist zu unterscheiden, ob es sich um Daten des Fluids oder um Oberflächentemperaturen handelt. Temperaturen und die Dichte des Fluids können, wie die anderen Fluidgrößen auch, mittels Isoflächen und Schnittebenen dargestellt werden. Temperaturen an den Oberflächen der Geometrieobjekte werden durch Kolorierung der entsprechenden Facetten-Subsets dargestellt. Auch hierbei reicht die Farbskala von Blau bis Rot, wobei Blau die tiefste und Rot die höchste vorkommende Oberflächentemperatur repräsentiert. Die Skala wird dynamisch der jeweiligen Simulation angepasst.

Komfortkriterien

Des Weiteren besteht die Möglichkeit, Ergebnisse der Berechnung von thermischem Komfort, welche durch die gekoppelte Software iZone (siehe [60]) berechnet wurden, auf dem implementierten, parametrischen Dummy darzustellen. Hierzu werden die entsprechenden Facetten-Subsets des Manikins, wie bei der Darstellung der Oberflächentemperatur, farbig eingefärbt. Die gewählte Farbe repräsentiert den Behaglichkeitslevel.

5.3 Anwendung in der Industrie: Interaktive Strömungssimulation im Siemens Medialab

In der Abteilung CT PP 2 der Siemens AG ist ein VR Raum („Medialab“) installiert. Dieser wurde hauptsächlich zur Präsentation von Simulationsergebnissen jeglicher Art eingerichtet. Er besteht aus insgesamt drei über Eck angeordneter Leinwänden plus zugehöriger Technik (siehe Abbildung 5.9).

5.3.1 Projektionstechnik

Die 3D Darstellung erfolgt mittels einer passiv Stereoprojektion. Hierzu werden pro Leinwand zwei Beamer eingesetzt, die je über einen um 90° gedrehten Polarisationsfilter verfügen. Der Nutzer trägt Polarisationsbrillen, welche ebenfalls mit entsprechenden Polarisationsfiltern ausgestattet sind.
Insgesamt sind also sechs Beamer installiert, die je an einen eigenen bildgebenden Rechner per DVI-Kabel angeschlossen sind.

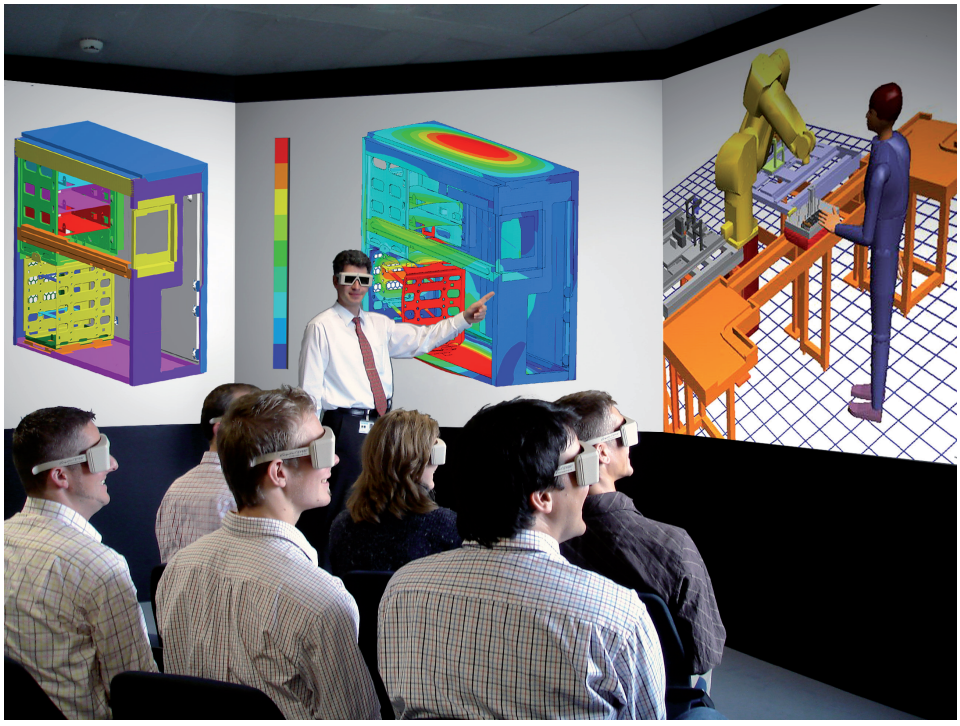


Abbildung 5.9: Medialab der Abteilung CT PP 2 der Siemens AG (Quelle: Siemens AG)

Zusätzlich ist noch ein separat stehender TFT-Monitor vorhanden, der zusammen mit der angeschlossenen Peripherie (Tastatur, Maus und Space-Mouse) zur Steuerung der Anlage dient.

5.3.2 Rechnerausstattung

Es stehen insgesamt acht „Athlon 64“ basierte Rechner zur Verfügung, welche über ein Gigabitnetzwerk vernetzt sind. Wie bereits erwähnt sind sechs dieser Rechner direkt mit je

einem Beamer verbunden und die beiden anderen Rechner dienen als Steuerungs- und für die hier beschriebene Anwendung als Simulationsrechner. Es handelt sich um handelsübliche Standardhardware.
Die Rechner, die an die Beamer angeschlossen sind (Visualisierungsrechner), sowie der Steuerrechner werden mit Microsoft Windows XP Professional betrieben, wohingegen der für die Simulation genutzte Rechner unter Suse Linux 9.3 64bit betrieben wird. Generell besteht aber bei allen Rechner die Möglichkeit, sie sowohl unter Windows als auch unter Linux zu betreiben. Die Auswahl des zu bootenden Betriebssystems erfolgt komfortabel und zentral für alle Rechner über ein webbasiertes Interface.

5.3.3 Portierung und Installation

Wie bereits beschrieben wird die Visualisierungsbibliothek OpenInventer verwendet. Jedoch unterstützt diese weder in der Standardvariante noch in der Multipipe Version eine Visualisierung, bei der der rechte und linke Kanal des Stereobildes von verschiedenen Rechnern generiert wird. Genau dies ist aber für den speziellen Aufbau des Medialabs erforderlich.
Um dieses Problem zu umgehen, wurde zuerst getestet, die Visualisierung zwar nur auf einem Rechner durchzuführen, die Ausgabe jedoch auf einen Xserver auf einem anderen Rechner umzuleiten und dort darzustellen. Dies erfordert jedoch den Einsatz einer Multipipe Lizenz für OpenInventor und das Betriebssystem Linux bzw Unix. Ein Nachteil dieser Lösung wäre auch, dass sich so nur eine der drei Leinwände im Siemens Medialab ansteuern lassen würde. Zudem erzeugt der Export der grafischen Ausgabe einen enormen Netzwerkverkehr, weil das Protokoll hierfür unkomprimiert arbeitet. Es ist aber, im Gegensatz zur Verwendung von z.B. VNC, weiterhin eine hochqualitative Ausgabe möglich, die auch auf die Hardwarbeschleunigungsfunktionen (z.B. OpenGL) der Grafikhardware zugreifen kann. Hauptgrund, warum diese Lösung jedoch fallen gelassen wurde, ist, dass in der Abteilung die Entscheidung fiel, die Visualisierung komplett unter Windows zu betreiben und somit eine Lösung wie diese, die zwingend den Einsatz von Linux bzw. Unix voraussetzt, nicht mehr möglich war.
Nach erneuter intensiver Analyse der zur Verfügung stehenden Mittel wurde die Programmierung eines eigenen Stereomodus für das Medialab der Abteilung innerhalb der Software `iFluids` beschlossen. Dies wird umgesetzt, indem die kooperative Version der Applikation um zusätzliche Fähigkeiten erweitert wurde. Konkret wird nun auf jedem zur Visualisierung benutzen Rechner eine Instanz des Clients gestartet, welche alle mit dem selben „GeomServer“ verbunden sind. Über die Konfigurationsdatei lässt sich für jeden Client festlegen, ob entweder das linke, das rechte oder überhaupt kein Stereobild gezeigt werden soll. Der Client auf dem Steuerrechner, dessen Bildschirmausgabe auf dem TFT-Monitor dargestellt wird, wird nicht im Stereomodus betrieben und stellt als einziger auch die Menüzeile und sonstige Einblendungen etc. in der GUI dar.
Weiterhin wurde eine Leader-Follower-Technik zur Kopplung der Clients implementiert. Dies bedeutet, dass die verschiedenen Clients in einer Weise gekoppelt sind, dass die Darstellung auf den „Stereo-Clients“ der Darstellung auf dem „Steuer-Client“ direkt folgt. Bewegt der Benutzer auf dem Steuerrechner also die Szene oder blendet Stromlinien ein, so erfolgen diese Aktionen bzw. Bewegungen zeitgleich auch auf der Stereoleinwand.
Da für jeden Rechner, auf dem ein Client ausgeführt wird, auch eine Lizenz des OpenInventor zur Verfügung stehen muss, wurde in der augenblicklichen Ausbaustufe die Darstellung auf das

mittlere Segment der Leinwand beschränkt. Ein weiterer Ausbau ist jedoch technisch möglich.

5.3.4 Steuerung der Anlage

Die Steuerung der Anlage sowie die Navigation in der virtuellen Umgebung wird am Kontrollrechner vorgenommen. Hier steht ein komfortables Interface zur Steuerung der Beamer zur Verfügung, mit welchem diese sich beispielsweise per Mausklick ein- und ausschalten lassen. Zusätzlich wurden kleine Tools installiert, um die Visualisierungsclients auf den mit den Beamern verbundenen Rechnern und die Simulation auf dem Rechencluster bequem starten und stoppen zu können.
Über die auf diesem Kontrollrechner laufende Instanz des Visualisierungsclients wird Geometrie in die Simulation geladen sowie durch die Visualisierung der Simulation navigiert. Dies geschieht in 2D mittels einer handelsüblichen Maus, da eine stereoskopische Darstellung auf dem verwendeten TFT-Monitor nicht möglich ist. Alternativ zur Verwendung einer normalen Computermaus läßt sich an diesem Kontrollrechner auch eine Space-Mouse betreiben.

5.4 Visualisierungstechniken zur Darstellung von Simulationsdaten aus kommerziellen CFD Programmen in einer VR-Umgebung

5.4.1 Problembeschreibung

Als eine weitergehende Fragestellung dieser Arbeit wurde untersucht, wie die generische Visualisierungsinfrastruktur, welche durch die beschriebene CSE vorhanden ist, auch zur Darstellung von Strömungsdaten aus kommerziellen Softwaresystemen genutzt werden kann. Als konkretes Beispiel soll hier die Kopplung des kommerziellen Strömungslösers ANSYS CFX mit der Visualisierungskomponente von `iFluids` gezeigt werden, welche durch moderate Anpassungen erreicht werden konnte. Hierbei wurde systembedingt nur eine offline-Visualisierung der gewonnenen Resultate betrachtet.
Da das Rechengitter in ANSYS CFX auf einem unstrukturierten Netz basiert und in der CSE ein strukturiertes Gitter verwendet wird, besteht das Hauptproblem in der Konvertierung der Daten zwischen den beiden nicht übereinstimmenden Gittern.

5.4.2 Verwendete Software

Ansys CFX

Als Strömungslöser kam in dieser Untersuchung die Software ANSYS CFX zum Einsatz. Hierbei handelt es sich um ein kommerzielles, leistungsfähiges general purpose CFD-Programm. Es untergliedert sich in einen Preprocessor, einen Löser und ein Postprocessing-Werkzeug.

Visualisierungs-Applikation aus `iFluids`

Die bereits vorgestellte Visualisierungsapplikation unseres CSE soll zur Darstellung der gewonnenen Simulationsdaten genutzt werden. Dies wird angestrebt, da die Visualisierungsapplika-

tion die Nutzung in einer VR-Umgebung erlaubt. Hierbei sollten die aus der CSE bekannten Visualisierungsmöglichkeiten, wie z.B. Stromlinien oder Schnittebenen, genutzt werden können.
In diesem Umfeld kam die Corba-basierte kollaborative Version der Visualisierungsumgebung (siehe [6]) zum Einsatz.

5.4.3 Extraktion der Ergebnisdaten aus CFX

Es wird davon ausgegangen, dass die Simulationsergebnisse aus CFX bereits vorliegen, das Preprocessing und die Simulation selbst werden in dieser Untersuchung deshalb außer Acht gelassen. Für die Extraktion der Ergebnisse wurden verschiedene Optionen untersucht. So wurde beispielweise evaluiert, ob sich die im Postprocessor von CFX vorhandenen Interpolationsfunktionen zur Interpolation der Daten vom unstrukturierten auf das nötige strukturierte Gitter nutzen lassen. Aufgrund der Funktionsweise dieser Funktionen lieferte dieser Weg nur wenig brauchbare Ergebnisse. Konkret führen die vorhandenen Interpolationsalgorithmen nur für ähnliche Netze und Gebiete zu verwertbaren Resultaten. Da das rechteckige, strukturierte Gitter jedoch für gewöhnlich stark vom Rechengebiet in CFX abweicht, führte dieser Ansatz unter anderem auch zu einer Extrapolation von Ergebnissen in Gebiete, die außerhalb des Strömungsgebietes lagen.
Aufgrund dieser Ergebnisse wurde eine eigenständige Interpolationsroutine entwickelt [52], welche das Mapping vom unstrukturierten CFX-Netz auf das strukturierte Gitter der CSE über geeignete Interpolationsalgorithmen durchführt.

5.4.4 Interpolation der Ergebnisse von einem unstrukturierten Netz auf ein strukturiertes Netz

Das strukturierte Gitter wird anhand der minimalen und maximalen Raumkoordinaten des unstrukturierten Gitters und eines vorgegebene Gitterabstands generiert. Die Definition des Gitters erfolgt von der Mitte des ursprünglichen unstrukturierten Rechengebiets aus.
Für die Interpolation wird die Kenntnis über die Beschaffenheit des strukturierten Gitters ausgenutzt und eine teure Nachbarschaftssuche vermieden. Als Datenstruktur für die Zwischenspeicherung der Daten während der Interpolation wird eine Liste gewählt. Je Datenpunkt des strukturierten Gitters werden 14 Werte gespeichert: Die Koordinaten in den drei Raumrichtungen, die drei Koordinaten im Gitter, sieben Fluid-Variablen (Geschwindigkeiten in u-, v- und w-Richtung, Dichte, Druck, Temperatur und Spannung) und ein Zähler. Man geht in umgekehrter Richtung vor. Man beginnt mit einer Schleife über die Knoten des ursprünglichen, unstrukturierten Rechengebiets. Hierbei werden die Koordinaten erfasst. Über die Koordinaten eines Punktes P kann ein Einheitswürfel, d.h. ein Würfel mit Kantenlänge dx, im strukturierten Gitter definiert werden, welcher den Punkt P beinhaltet. Es werden die Knoten-Indizes diese Einheitswürfels im strukturierten Gitter berechnet. Wenn ein Knoten außerhalb des Gebiets liegt, lässt sich dies über einen Knotenindex mit einem negativen Wert oder einem Wert größer der maximalen Knotenanzahl im strukturierten Gitter erkennen.
Im nächsten Schritt wird für die gültigen Knoten der Abstand zum Punkt P errechnet. Ist der bestimmte Abstand eines Gitterknotens kleiner als ein definierter Grenzwert c, werden die sieben Fluidvariablen von P auf diesen Knoten kopiert und der Zähler für diesen Knoten um eins erhöht. Sollte ein Gitterknoten im Einflussbereich von mehreren Knoten des

ursprünglichen unstrukturierten Netzes liegen, so werden von jedem dieser Punkte die Werte der Fluidvariablen auf den Gitterknoten geschrieben und der Zähler dementsprechend erhöht. Zur Mittelwertbildung werden in einem weiteren Durchgang nach Durchlaufen aller Punkte des unstrukturierten Netzes die Fluidwerte an jedem Knoten mit einem Zähler größer eins durch den Wert des Zählers dividiert (siehe Algorithmus 5.1).

Algorithm 5.1 Interpolations-Algorithmus

for all $Knoten_{unstrukt.}\,(x, y, z)$ **do**
  Berechne Position des $Knoten_{unstrukt.}$ im strukt. Gitter
  Bestimme die 8 umliegenden $Knoten_{strukt.}$ im strukt. Gitter
  Berechne die Distanzen d zw. dem $Knoten_{unstrukt.}$ und den 8 umliegenden $Knoten_{strukt.}$
  for all $Knoten_{strukt.}$ **do**
    if $d < \sqrt{3} \cdot Gitterweite/2$ **then**
      Addition der Fluidvariablen des $Knoten_{unstrukt.}$ auf den jeweiligen $Knoten_{strukt.}$
      Erhöhung des Zählervariable des jeweiligen $Knoten_{strukt.}$ um 1
    end if
  end for
  for all $Knoten_{strukt.}$ mit Zählervariable > 1 **do**
    Division der Werte der Fluidvariablen durch den Wert der Zählervariable
  end for
end for

Da bei diesem Ansatz von den Punkten des ursprünglichen unstrukturierten Netzes ausgegangen wird, ergibt sich das Problem, dass manchen Fluidknoten im strukturierten Netz keine Werte zugewiesen werden, da der Abstand zu Knoten des ursprünglichen Netzes zu groß ist. Um diesem unerwünschten Effekt zu begegnen, wurde eine „smoothing“ Funktion eingeführt, die solche Knoten speziell behandelt.
Während dieses Prozesses werden solchen Knoten die gemittelten Werte von sechs umliegenden Knoten zugewiesen. Hierbei muss jedoch vorab getestet werden, ob der zu betrachtende Punkt gültig, d.h. innerhalb der Fluiddomäne liegt. Ebenso soll vermieden werden, dass Knoten auf den Rändern beeinflusst werden.
Die durch den „smoothing“ Algorithmus gewonnenen Daten werden in einem temporären Feld zwischengespeichert und erst nach Durchlaufen des ganzen Feldes in das eigentliche Datenfeld geschrieben, um eine Extrapolation in ungültige, d.h. nicht zur Fluid-Domäne gehörende Gebiete zu verhindern.

5.4.5 Übergabe der Daten an `iFluids`

Nach Beendigung der Interpolationsschritte müssen die Daten noch in die von der CSE verwendete Datenstruktur überführt und an die Visualisierung übertragen werden.
Die Ankopplung an die CSE erfolgt über einen sog. „DataServer“, welcher den Platz des Simulations-Servers der kooperativen Version der CSE (siehe [4]) einnimmt. Hierbei lösen die Befehle zum Starten der Simulation das Einlesen der Daten und die Interpolationsfunktionen aus. Die Aktivierung des Daten-Updates führt zur Übertragung der Daten. Somit sind keine Änderungen an den übrigen Komponenten der kooperativen CSE notwendig.

Die Fremddaten werden durch den Daten-Server von einem Datenträger eingelesen, wie oben beschrieben behandelt und an die Visualisierung weitergeleitet (siehe Abbildung 5.10). Im Vergleich mit der ursprünglichen Version der CSE entfallen sämtliche MPI-Schnittstellen; die anfallende Kommunikation wird über Corba-Schnittstellen abgewickelt. Die Setup-Informationen, wie z.B. der Gitterabstand, werden vom Geometrie-Server an den „DataServer“ übertragen.

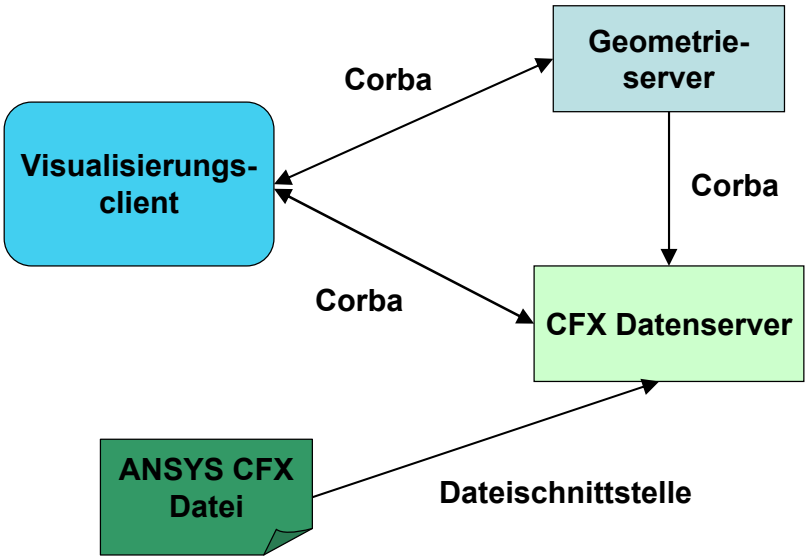


Abbildung 5.10: Kommunikationsstruktur zur Visualisierung von Fremddaten aus Ansys CFX in `iFluids`.

5.4.6 Darstellung durch die Visualisierungsapplikation in `iFluids`

Innerhalb der Visualisierungsapplikation, konkret dem Client der kooperativen Version der CSE, stehen alle Visualisierungsoptionen der CSE zur Verfügung. Dies sind beispielsweise Stromlinien oder Schnittebenen für die verschiedenen Fluidvariablen.
Weiter muss sich der Nutzer durch die Adaption der neuen Komponenten, d.h. des sog. „DataServers“, an die bestehende CS-Umgebung nicht in der Bedienung umstellen.
Es besteht weiterhin die Möglichkeit, über die Visualisierungsapplikation Geometrieobjekte in das System zu laden. Da es sich jedoch um eine offline-Visualisierung handelt, hat dies keine Auswirkung auf die dargestellten Fluiddaten. Es erfolgt lediglich eine überlagerte Darstellung von Geometriedaten und visualisierten Fluidparametern.
Die stereoskopische Visualisierung in einer 3D-Umgebung, wie beispielsweise in 5.3 beschrieben, ist weiterhin möglich. Abbildung 5.11 zeigt einen Vergleich der Postprocessing Visualisierung in `iFluids` und Ansys CFX.

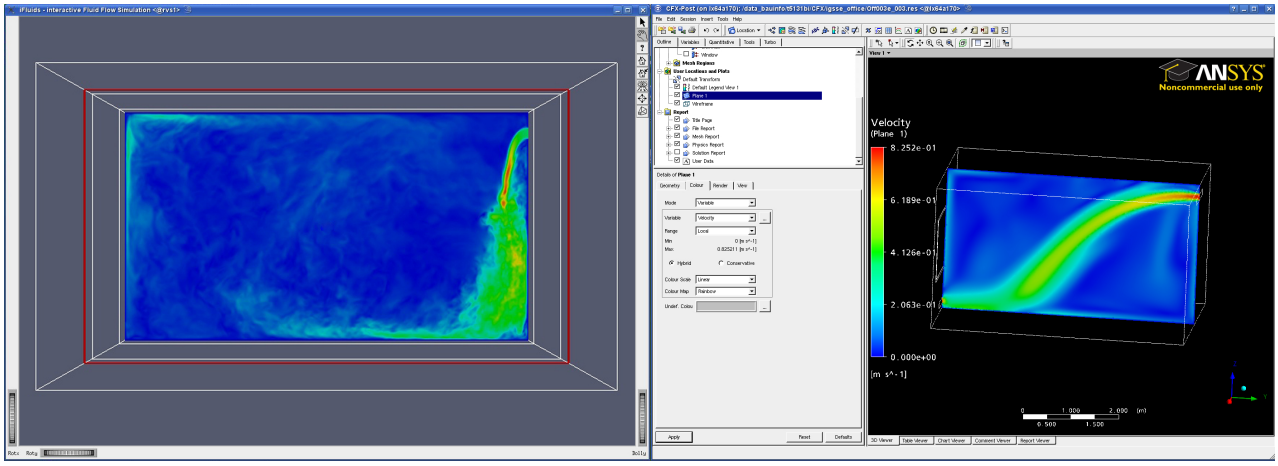


Abbildung 5.11: Visualisierung mit iFluids (links) und Ansys CFX (rechts).

Kapitel 6

Kommunikation und Datenstrukturen für eine parallelisierte interaktive Strömungssimulation

In einer interaktiven Strömungssimulation werden große Datenmengen erzeugt bzw. müssen zwischen Simulation und Visualisierung übertragen werden. Es soll dem Nutzer jedoch trotzdem ermöglicht werden mit einer möglichst geringen Verzögerung mit der Simulation zu interagieren, was eine besondere Herausforderung darstellt. Daher werden zwangsläufig Methoden des parallelen Rechnens notwendig, um große Datenmengen bewältigen zu können bzw. umfangreiche Aufgaben schneller bearbeiten zu können.

6.1 Paralleles Rechnen

Paralleles Rechnen kann verschiedene Ziele verfolgen ([57], [42]):

- Das Rechnen von großen Problemen.
- Die schnellere Berechnung eines Problems.
- Die Nutzung spezieller, angepasster Hardware.
- Erzielung eines höheren Durchsatzes, beispielsweise bei Parameterstudien.

Generell gehören in den Ingenieurwissenschaften Aufgaben aus der Strömungsmechanik zu den Problemen mit den meisten Freiheitsgraden (DOF), welche somit auch die meiste Rechenzeit beanspruchen. Dies können zum einen Anwendungen mit einem besonders großen Rechengebiet, aber auch besonders fein aufgelöste Probleme sein, um beispielsweise Turbulenzen über eine DNS (Direct Numerical Simulation) aufzulösen.
Bei der Hardware für große Hochleistungsrechner unterscheidet man zwischen verschiedenen Klassen. Zu nennen sind hier Vektorrechner, Rechencluster (homogen oder inhomogen) und große Multiprozessor Systeme, wie beispielsweise die SGI Altix 4700. Es sind auch noch weitere zum Teil hoch spezialisierte Systeme am Markt verfügbar, jedoch wurden diese für die hier beschriebene Anwendung nicht betrachtet, da deren Einsatz größere, sehr spezifische Änderungen

am Code erfordern würden. Da Vektorrechner zwar theoretisch für die in dieser Arbeit gezeigten Simulationen eingesetzt werden können, aber nicht zum Einsatz kamen, wird auf diese im Weiteren auch nicht näher eingegangen.
Bei Rechenclustern handelt es sich um eigenständige, vernetzte, handelsübliche PC- oder Workstationsysteme. Üblicherweise kommen hierbei eine hohe Anzahl an Knoten sowie schnelle, für diesen Anwendungszweck optimierte Netzwerkverbindungen (z.b. Myrinet oder Infiniband) zur Anwendung.
Bei großen Multiprozessorsystemen handelt es sich um große, meist homogene Systeme mit einer sehr hohen Anzahl an Prozessoren bzw. Prozessorkernen. Das Jaguar Cray XT5-HE System, welches aktuell (Stand November 2010) auf Platz zwei der Top500 Liste der schnellsten Rechner geführt wird, kommt beispielsweise auf 224162 Rechenkerne und erreicht damit eine maximale LINPACK[1] Leistung von 1759 TFlops. TFlops steht für 10^{12} *Floating Point Operations Per Second* (deutsch: Gleitkommaoperationen pro Sekunde) und stellt eine Maßeinheit für die Geschwindigkeit von Computersystemen oder Prozessoren dar [73].
Ein Merkmal zur Klassifikation von parallelen Rechnern ist das Speicherlayout des Systems. Hier wird in folgende zwei Konzepte unterschieden:

- Shared memory (Gemeinsamer Speicher).
- Distributed memory (Verteilter Speicher).

Bei einem Shared Memory System sind die Prozessoren über entsprechende Verbindungen mit den Speichersegmenten verbunden. Jeder Prozessor hat Zugriff auf den globalen gemeinsamen Speicheradressraum. Ein Vorteil dieser Systeme ist die einfache Implementation von parallelen Programmen, da jeder Prozess auf alle Daten direkt zugreifen kann. Die Prozesse kommunizieren durch Nutzung des gemeinsame Speichers. Dies kann zu Geschwindigkeitsvorteilen führen. Ein Nachteil des Konzepts ist jedoch, dass sogenannte *Locking* Mechanismen benötigt werden, um Speicherzugriffskonflikte verschiedener Prozesse zu verhindern. Oft variiert die mögliche Speicherzugriffsgeschwindigkeit entsprechend der physikalischen Lage bzw. Entfernung zwischen Prozessor und Speichersegment. Aus diesem Grund wird teilweise zusätzlicher lokaler Speicher/Cache bereit gehalten ([57]).
Bei Distributed Memory Systemen hat jeder Prozessor seinen eigenen lokalen Speicher und Speicheradressraum. Damit ein Prozess auf Daten zugreifen kann, müssen diese somit lokal im Arbeitsspeicher vorhanden sein. Eine Kommunikation zwischen verschiedenen Prozessoren bzw. ein Datenaustausch erfolgt über explizites Senden von Nachrichtenpaketen über ein Netzwerk (*message passing*) ([57]). Diese Systeme zeigen bei zunehmender Anzahl an Prozessoren eine deutlich bessere Skalierbarkeit. Die Leistungsfähigkeit eines solchen Systems ist wesentlich von der Art und der Geschwindigkeit des Kommunikationsnetzwerkes abhängig.
Eine weiteres Unterscheidungsmerkmal der verschiedenen parallelen Rechnersysteme ist die Art des physikalischen Speicherzugriffs. Hier unterscheidet man in:

- UMA (Uniform memory access)
- NUMA (Non uniform memory access)

[1] Bei LINPACK handelt es sich um eine numerische Programmbibliothek zur Lösung von linearen Gleichungssystemen, welche auch zur Geschwindigkeitsmessung von Hochleistungsrechnern eingesetzt wird.

- NORMA (No remote memory access)

In folgender Tabelle 6.1 sind die möglichen Kombinationen hinsichtlich Adressraum und Speicheranordnung dargestellt ([57]):

	zentaler Speicher	verteilter Speicher
globaler AR	UMA, SMP	NUMA
verteilter AR	$\emptyset$	NORMA

Tabelle 6.1: Mögliche Kombinationen hinsichtlich Adressraum und Speicheranordnung.

Bei einem UMA System greifen alle Prozessoren mit der gleichen Zugriffsgeschwindigkeit auf den gemeinsamen Speicher zu, wobei jeder Prozessor noch zusätzlichen lokalen Cache Speicher haben kann.
Die Speicherzugriffszeit bei einem NUMA System variiert hingegen je nach physikalischer Position der Speicherzelle. Trotzdem bilden alle Speichersegmente einen gemeinsamen Adressraum. Jedoch kann auf Speicher, der lokal an einem Prozessor angesiedelt ist von diesem aus schneller zugegriffen werden als auf Speicher, welcher an einem anderen Prozessor installiert ist.
UMA und NUMA Systeme kommen bei Shared Memory Systemen vor. NORMA hingegen trifft für Distributed Memory Systeme zu. Auch hier lässt sich nochmals in *Uniform Communication Access* (UCA) und *Non Uniform Communication Access* (NUCA) unterscheiden. Bei UCA Maschinen kommunizieren alle Knoten mit der gleichen Geschwindigkeit, wohingegen bei NUCA Systemen die Netzwerkgeschwindigkeit zwischen den einzelnen Knoten je nach Lage variiert.
Die unterschiedlichen Übertragungsraten bei NUMA bzw. NUCA Systemen können durchaus zu Leistungseinschränkungen eines ungünstig aufgeteilten Problems führen. Ziel sollte es bei NUMA sein, benötigte Daten möglichst nahe an den betreffenden Prozessoren zu speichern, um somit eine bestmögliche Performance zu erreichen. Das gleiche gilt entsprechend für NUCA Systeme, bei denen Prozesse, die untereinander einen höheren Kommunikationsbedarf haben, so auf dem System platziert werden sollten, dass ein möglichst schneller Interconnect zur Verfügung steht.
Dies führt zu den möglichen Arten der Kopplung der einzelnen Komponenten. Unabhängig davon, ob es sich um ein *Shared Memory* oder *Distributed Memory* System handelt, existieren verschiedene Kopplungsschemata.

- dynamische Topologien
  - Switches
  - Kreuzschienenverteiler (Crossbar)
- Statische Topologien

Bei der statischen All-to-All Kopplung ist jeder Prozessor direkt mit allen anderen Prozessoren bzw. jedem Speicher verbunden. Dies ist theoretisch eine optimale Kopplung, da die Kommunikationslänge, d.h. die Anzahl der Kommunikationssegmente, die für eine Nachricht bzw. einen Datenzugriff genutzt werden muss, immer eins beträgt. Praktisch ist diese Art der Kopplung aus Kostengründen kaum umsetzbar, da pro Prozessor $p - 1$ Verbindungen nötig

wären (p bezeichnet die Gesamtanzahl der Prozessoren des Systems) und somit für den Vernetzungsaufwand $\mathcal{O}(p^2)$ gilt. Bei einer dynamischen Kopplung wird der Kommunikationskanal zum Datenaustausch zur Laufzeit geschaltet und anschließend wieder abgebaut.

Dynamische Topologien

Wenn dynamische Topologien zur Anwendung kommen, gibt es wiederum verschiedene Möglichkeiten dies umzusetzen. Alternativ hierzu können Zwei-Wege Switches eingesetzt werden, was die Anzahl der Switches reduziert, aber die Kommunikationslänge ggf. verlängert. Neben diesen speziellen Arten können auch gewöhnliche Netzwerk Switches zur Anwendung kommen; dies trifft v.a. für den Einsatz in Rechenclustern zu. In diese Kategorie fallen Fast Ethernet oder Gigabit Ethernet Switches, genauso wie schnelle, optimierte Netzwerke wie Myrinet oder Infiniband.

Statische Topologien

Bei statischen Netzwerken können verschiedene Topologien zur Vernetzung von Prozessoren und Speicher zum Einsatz kommen. Beispiele hierfür sind lineare Strukturen, Ringtopologien, sternförmige Netze, Gitterstrukturen, Binärbäume oder Hyperwürfel.
Bei linearen oder Ringtopologien ist jeder Prozessor nur mit seinen beiden direkten Nachbarn verbunden. Somit ergibt sich für weit auseinander liegende Prozessoren eine hohe Kommunikationslänge. Dies ist auch der Grund, warum diese Netze für eine große Prozessoranzahl p ungeeignet, da zu langsam, sind. Vorteile bieten „höher dimensionale“ Topologien, wie Netze, Binärbäume oder Hyperwürfel.
Eine häufig genutzte Netzwerktopologie für große parallele Maschinen sind Binärbäume. Hierbei handelt es sich um ein zweidimensionales Netzwerk mit N Knoten und $N-1$ Kanten. Das Netz hat einen Ursprungsknoten, von welchem zwei Kindknoten abgehen. Von jedem dieser Knoten zweigen anschließend wiederum zwei Knoten ab. Dadurch ergibt sich bei einheitlicher Netzwerkbandbreite in allen Schichten ein Flaschenhals in den höheren Schichten. Dies kann umgangen werden, in dem in jeder höheren Schichte die Netzwerkkapazität verdoppelt wird. Man erhält dann einen so genannten *fat tree*. Diese Architektur findet beispielsweise im HLRB II (siehe Kapitel 7) Anwendung.

6.2 Parallelisierungsstrategie für `iFluids`

Die Parallelisierung der Computational Steering Umgebung wurde mittels Message Passing Interface (MPI) umgesetzt. Hierbei kommen, je nach Anforderung und Plattform, verschiedene Implementationen dieser Bibliothek zur Anwendung. Zusätzlich werden bei bestimmten Versionen, beispielsweise der kollaborativen Version, Teile der Kommunikation statt mittels MPI über CORBA abgewickelt.
Unabhängig von der Version der CSE wurde auf der Simulationsseite ein Master-Slave-Konzept umgesetzt. Hierbei übernimmt der Master-Knoten sämtliche Kommunikationsaufgaben mit der Visualisierungsapplikation, die Datenhaltung von Geometrieobjekten und Simulationsparametern auf Simulationsseite sowie die vollautomatische Gittergenerierung. Das Rechengitter wird entsprechend der auf der aktuellen Hardware benötigten Gebietszerlegung zerteilt und

zusammen mit den notwendigen Parametern für die Simulation an die sog. Slave-Knoten verteilt.
Des Weiteren übernimmt der Master-Knoten auch das Einsammeln der Teilergebnisse der einzelnen Rechenknoten. Die Übertragung der gesammelten Daten der Simulation werden in einem durch den Benutzer vorgegebenen Update Intervall von x Simulationszeitschritten an die Visualisierungsapplikation zur Auswertung übertragen.
Aufgrund dieses Konzepts sind $n + 1$ Simulationsknoten für eine Gebietszerlegung (siehe Abschnitt 6.3) in n Teile erforderlich.
Durch dieses Vorgehen können Performanceeinbußen der Simulation bedingt, durch die Kommunikation mit der Visualisierung vermieden werden. Diese Einbußen kommen besonders bei einem Setup mit eingeschränkter Bandbreite zwischen Simulationsrechnern und Visualisierungshardware zum Tragen.
Auch auf der Visualisierungsseite der CSE wurde die Kommunikation von der Datenvisualisierung entkoppelt, um dem Nutzer ein unterbrechungsfreies Arbeiten zu ermöglichen. Dies erfolgte jedoch durch Einführung eines zusätzlichen Kommunikations-Threads und nicht über einen zusätzlichen MPI-Prozess.

Eine schematische Darstellung der Kommunikationsstruktur ist in Abbildung 5.5 gegeben.
Bei der Implementierung der Kommunikation wurde ein Ereignis basiertes Konzept umgesetzt, bei dem auf Simulations- und Visualisierungsseite jeweils ein sogenannter Mailbox Thread läuft, welcher auf Nachrichten der jeweils anderen Seite wartet. Basierend auf der Art der Nachricht wird anschließend eine entsprechende Funktion aufgerufen; beispielsweise eine MPI-Empfangsfunktion zum Empfang eines neuen Geometrieobjekts. Somit werden starre Sende-Empfangs-Muster aufgebrochen.
Über diesen Mechanismus wird sichergestellt, dass es zu keinem Deadlock zwischen Simulation und Visualisierung kommt. Unter einer Deadlock-Situation versteht man in diesem Zusammenhang einen Zustand, bei dem das Programm bei einer Anweisung stehen bleibt, welche nicht abgeschlossen werden kann. Dies kann in diesem Fall entstehen, wenn beide MPI-Prozesse (Visualisierung und Simulation) gleichzeitig versuchen Daten an den jeweils anderen Prozess zu senden. Wenn jedoch der andere Prozess nicht empfängt weil er selbst versucht Daten zu senden, kann die Sendeoperation nicht abgeschlossen werden und das Programm „verklemmt“ sich.

6.3 Gebietszerlegung

Die Parallelisierungsbibliothek MPI ist im Gegensatz zu andern Ansätzen, wie beispielsweise OpenMP, auf den Einsatz auf verteilten Systemen mit eigenständigem Hauptspeicher ausgelegt. Dies bedeutet, dass das Rechengebiet entsprechend der Anzahl der verwendeten Simulationsprozesse unterteilt werden muss und ein einzelner Prozess im Umkehrschluss keinen Zugriff mehr auf das gesamte Gebiet hat, da die einzelnen Teile im lokalen Speicher der Prozesse abgelegt sind. Dadurch ergibt sich auch für gewisse Probleme, z.B. in der Strömungssimulation, die Notwendigkeit des Datenaustausches zwischen aneinandergrenzenden Gebieten.
Zur Lösung des Problems für innere Gebietsränder werden beispielsweise zusätzlich so genannte *ghost* Knoten oder auch *ghost* Zonen eingeführt (siehe Abbildung 6.1). Diese müssen vor jedem Iterationsschritt des Algorithmus mit dem benachbarten Gebiet ausgetauscht werden.

Wenn für einen Algorithmus mehr als nur ein angrenzender Knoten benötigt wird, kommen *ghost* Zonen mit einer entsprechend den Anforderungen definierten Größe zum Einsatz.

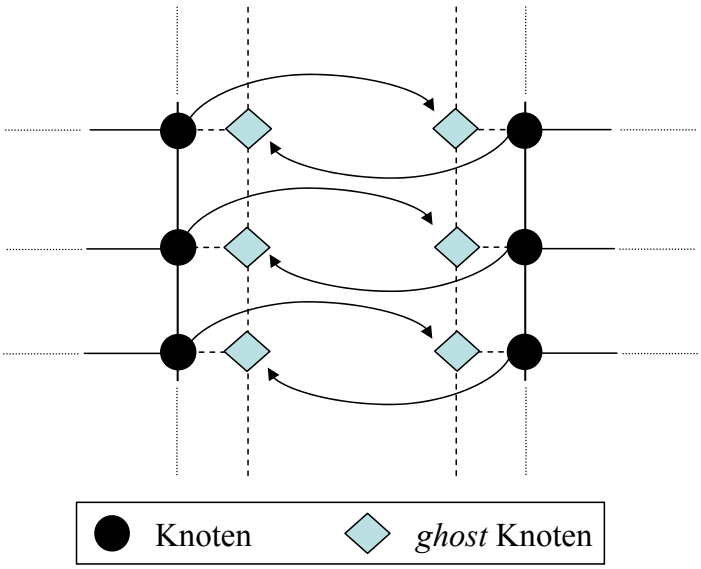


Abbildung 6.1: Datenaustausch zwischen zwei benachbarten Gebieten mittels zusätzlicher *ghost* Knoten in 2D (nach [57]).

Für die Gebietszerlegung gibt es in der Literatur zahlreiche verschiedene Ansätze. Eine optimale Zerlegung ist im Allgemeinen dann erreicht, wenn die Rechenlast gleichmäßig auf die zur Verfügung stehenden Rechenknoten verteilt ist und der Kommunikationsaufwand zwischen den Knoten minimal ist.
Die ersten Ideen zur Gebietszerlegung gehen auf Schwarz [51] zurück. Prinzipiell unterscheidet man in überlappende und nicht überlappende Ansätze, wobei sich dies auf sich überlappende bzw. nicht überlappende Simulationsgebiete bezieht.
In der Fachliteratur wurden verschiedene Methoden zur Gebietszerlegung speziell für den Einsatz mit der Lattice-Boltzmann Methode vorgestellt. Ein Ansatz von Freudinger et al. [18] basiert beispielsweise auf hierarchischen Gittern und nutzt die frei erhältliche METIS Bibliothek [44]. Bei METIS handelt es sich um eine Sammlung von Programmen u.a. zur Partitionierung von unstrukturierten Graphen und Finite-Elemente Netzen.
Wenn, so wie in der CSE `iFluids`, strukturierte Gitter zur Anwendung kommen, ist oftmals eine Unterteilung in Scheiben oder Blöcke vorteilhaft. Untersuchungen hinsichtlich der Qualität von Scheiben- und Blockzerlegung für einen LBM Code wurden zum Beispiel von Satofuka et al. [50] und Wenisch [71] speziell für die Architektur der Hitachi HPC-Systeme durchgeführt. In Abbildung 6.2 sind verschiedene Formen der rechteckigen Gebietszerlegung für 3D Gebiete dargestellt. Wie jedoch auch die Untersuchungen von Satofuka et al. zeigen, ist eine optimale Gebietszerlegung auch von der verwendeten Hardware abhängig. Hier spielt zum einen die Speicherarchitektur (distributed oder shared Memory) oder auch die Leistungsfähigkeit des Netzwerkinterconnects eine Rolle. Des Weiteren beeinflusst auch die zur Anwendung kommende MPI Distribution und der Charakter der Simulation die Effizienz einer gewählten Gebietszerlegung. So lieferte in der von Satofuka et al. vorgestellten Untersuchung eine Zerlegung in Scheiben wider erwarten ein besseres Ergebnis als eine Unterteilung in Blöcke.

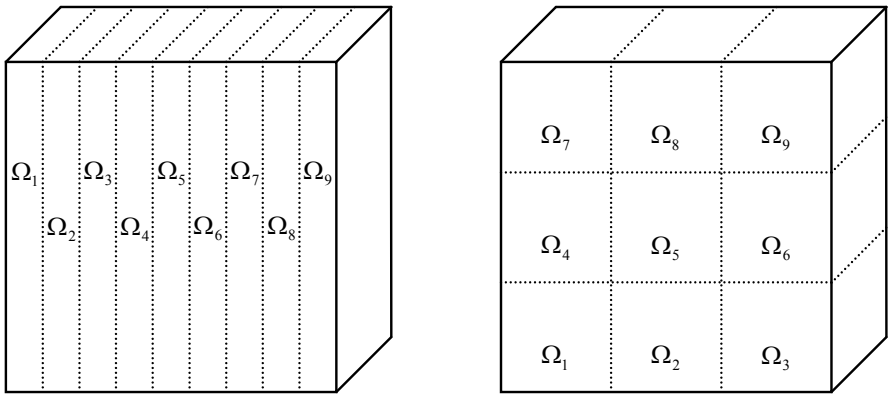


Abbildung 6.2: Verschiedene Möglichkeiten zur Gebietszerlegung eines drei dimenionalen Gebiets (nach [57]).

Da der Code der CSE `iFluids` in den Anfängen auf den Einsatz auf dem ehemaligen Höchstleistungsrechner Hitachi SR8000 (im Gegensatz zur Altix 4700 handelte es sich hierbei um einen Pseudo-Vektorrechner) optimiert wurde, fiel die Wahl der Gebietszerlegung zum damaligen Zeitpunkt auf ein Scheiben basiertes Schema. Prinzipiell lassen sich in dem Code jedoch auch andere Schemata, wie beispielsweise ein blockorientierter Ansatz, nutzen (siehe van Treeck [58]).
Weitere moderne Ansätze zur Gebietszerlegung nutzen raumfüllende Kurven (auch FASS[2]-Kurven genannt) wie Hilbert- oder Peano-Kurven [1].
Mit der Portierung der CSE auf die massiv parallele Umgebung des HLRB II zeigen sich jedoch sehr deutlich die Probleme des ursprünglich gewählten Gebietszerlegungsverfahrens für die hauptsächlich betrachteten Simulationsgebiete. Das Einsatzgebiet der CSE ist die Simulation von Luftströmungen in Innenräumen und ggf. eine daran anschließende Komfortbewertung. Typischerweise werden hierbei Räume betrachtet, bei denen die Abmessungen in den drei Raumrichtungen in einer ähnlichen Größenordnung vorliegen. Es erfolgt jedoch nur eine Gebietszerlegung in Scheiben in x-Richtung. Dadurch ist die maximal mögliche Anzahl an Prozessen stark eingeschränkt, da die theoretisch kleinstmögliche Scheibe der Gebietszerlegung eine Ausdehnung von einem Voxel in x-Richtung, aber von $n_{y,max}$ Voxel in y-Richtung und $n_{z,max}$ Voxel in z-Richtung hat. Die absolut höchstmögliche Anzahl an Prozessen zur Parallelisierung eines Problems ist somit $n_{x,max}$, wobei dies der Anzahl der Voxel in x-Richtung entspricht und definiert ist als die Länge des Gebiets dividiert durch den gewählten Gitterabstand. Da das Gebiet in y- und z-Richtung jedoch nicht unterteilt wird, ist es durchaus wahrscheinlich, dass ein Problem trotz Nutzung der maximal möglichen Anzahl an Prozessen nicht so zu parallelisieren ist, dass das Teilgebiet eines Prozesses klein genug ist, um in den Cache eines Knotens zu passen und somit die maximal mögliche Leistung zu erzielen.
Kanal- oder röhrenartige Strukturen, beispielsweise auch Blutgefäße, mit einer im Vergleich zu den anderen Raumrichtungen großen Ausdehnung in nur eine Raumrichtung ließen sich mit dieser Art der Gebietszerlegung jedoch gut parallelisieren. Wie in Abbildung 6.3 gut zu erken-

[2]Space-**f**illing, self-**a**voiding, **s**imple und self-**s**imilar.

nen, ist für diese Art von Geometrien zum einen der Kommunikationsaufwand[3], d.h. die Anzahl der zu übertragenden Variablen, kleiner und die Geometrie lässt sich in mehrere einzelne Abschnitte zerlegen. Verglichen hiermit entsteht bei einer blockartigen Struktur ein wesentlich größerer Kommunikationsaufwand und die Anzahl der möglichen Unterteilungen bei diesen Geometrien ist durch eine minimale Scheibendicke stärker limitiert als bei länglichen Strukturen, bei denen eine wesentlich größere Ausdehnung in der Richtung der Gebietszerlegung, verglichen mit den beiden anderen Raumrichtungen, existiert. Für die angestrebte Anwendung wäre jedoch eine blockweise Gebietszerlegung zu bevorzugen.

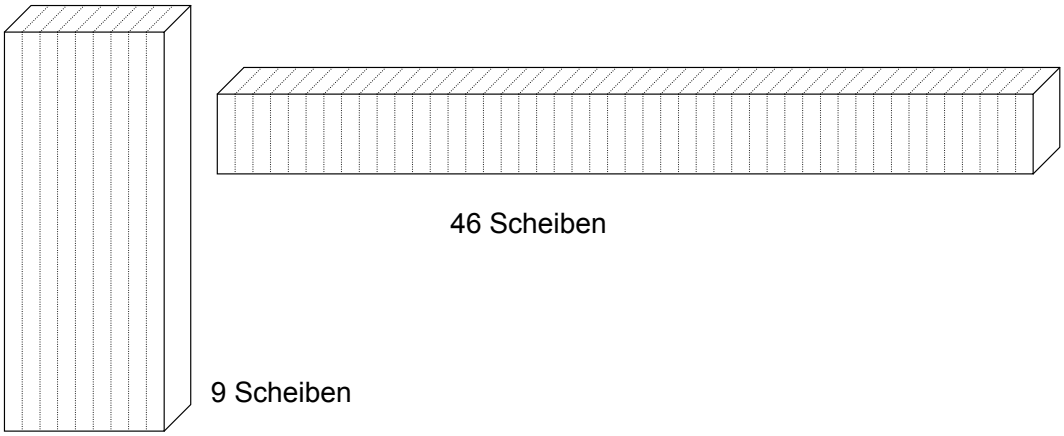


Abbildung 6.3: Scheibenweise Gebietszerlegung für verschiedene Geometrien.

6.4 Softwarebibliotheken zum parallelen Rechnen

Es existieren mehrere Möglichkeiten, um einen Rechencode zur Steigerung der Rechenleistung auf mehreren Prozessoren parallel auszuführen. Gängige Softwarebibliotheken zu diesem Zweck sind beispielsweise MPI (Message Passing Interface) und OpenMP, welche jeweils unterschiedliche Konzepte verfolgen.

6.4.1 Message Passing Interface

Bei MPI handelt es sich um einen Standard zum Austausch von Nachrichten zwischen verteilten Rechnern. Es ist ein weitgehend plattform- und programmiersprachenunabhängiges Kommunikationsinterface, welches sowohl eine Punkt zu Punkt als auch kollektive Kommunikation, d.h. im Rahmen von Gruppierungen (z.B. durch Broadcast, Scatter, Gather) unterstützt. Ausgehend von der Definition des Protokolls wurden diverse Implementierungen umgesetzt. Zu nennen sind hier beispielsweise MPICH, LAM/MPI, OpenMPI oder herstellerspezifische Implementierungen, wie sie zum Beispiel von SGI für die Altix-Architektur entwickelt wurden. Aufgrund der definierten Funktionsaufrufe ist es jedoch im Allgemeinen problemlos möglich einen Programmcode mit verschiedenen MPI-Implementierungen zu nutzen, ohne dass hierfür

[3]Dieser ist proportional zur Oberfläche der bei der Gebietszerlegung entstandenen Gebiete.

im Quellcode Änderungen erforderlich sind. Der MPI-Standard unterscheidet MPI-1 und MPI-2, wobei MPI-2 größtenteils abwärts kompatibel zu MPI-1 ist. Die CSE `iFluids` lässt sich mit beiden Versionen betreiben.

Die meisten Implementationen erlauben jedoch keine Kommunikation zwischen unterschiedlichen Plattformen, wie es zum Beispiel bei der Kommunikation zwischen HLRB2 und rvs1 am Leibniz Rechenzentrum der Fall wäre. Bei der Standard MPI-Version des HLRB2 handelt es sich um eine auf die spezielle ccNUMA[4]-Architektur des Systems angepasste Implementation des Herstellers SGI. Da es sich bei dem Visualisierungssystem rvs1 um ein Opteron-basiertes System eines anderen Herstellers handelt, ist eine direkte Kommunikation unter Anwendung dieser MPI-Version nicht möglich.
Um dieses Problem zu umgehen, kann auf alternative Implementierungen wie beispielsweise OpenMPI ausgewichen werden. Diese erlauben, entsprechende Versionen vorausgesetzt, eine hybride Kommunikation zwischen unterschiedlichen Plattformen. Hierbei kann jedoch die Leistungsfähigkeit der speziellen Kommunikationsinfrastruktur zwischen den Rechenknoten nicht mehr voll ausgenutzt werden.
Eine weitere, ebenfalls mit `iFluids` genutzte Möglichkeit besteht in der Verwendung spezieller Bibliotheken wie PACX-MPI[5] (siehe [38]). Hierbei handelt es sich um eine Bibliothek zur Kopplung von verschiedenen Rechnern bzw. Rechenclustern über ein lokales Netzwerk oder sogar das Internet.
Die Nutzung einer Bibliothek wie PACX-MPI hat den Vorteil, dass für die Kommunikation zwischen den Simulationsprozessen auf dem Rechencluster bzw. Hochleistungsrechner eine optimierte, herstellerspezifische MPI Implementation zum Einsatz kommen kann und nur für die Kommunikation zwischen Simulation und Visualisierung die spezielle Kommunikationsbibliothek genutzt werden muss. Dies erlaubt die vollständige Ausnutzung von speziellen Hochleistungsverbindungen zwischen den Simulationsprozessen. Das Kommunikationskonzept von PACX-MPI sieht jedoch die Bündelung sämtlicher Kommunikation zwischen den unterschiedlichen Plattformen durch sogenannter *Daemons* vor. Dies hat einerseits gewisse Vorteile, v.a. wenn beispielsweise Firewalls etc. zu überwinden sind, andererseits werden hierdurch zwei zusätzliche MPI-Prozesse nur für die Kommunikation benötigt.

6.4.2 OpenMP

OpenMP basiert anders als MPI nicht auf dem Konzept, das gleiche Programm auf mehreren Rechenknoten parallel auszuführen, sondern stellt eine Implementierung von „Multithreading“ dar.
Hierbei handelt es sich um eine Strategie, bei welcher von einem Master-Thread ausgehend eine gewisse Anzahl an Slave-Threads gestartet werden, zwischen denen genau definierte Rechenoperationen des Master-Threads aufgeteilt werden. Alle Threads bearbeiten einen Teil der Aufgabe zeitgleich, wobei die Kontrolle der Threads und deren Verteilung auf verschiedene Prozessoren durch die Laufzeitumgebung erfolgt [45].
OpenMP ist Teil des verwendeten Compilers und ebenfalls für mehrere Hardware- und Softwareplattformen sowie verschiedene Programmiersprachen verfügbar. Die Nutzung erfolgt durch

[4] Cache coherent NUMA.
[5] **PA**rallel **C**omputer e**X**tension

die Verwendung von entsprechenden Compiler Pragmas im Quellcode und durch das Setzen von entsprechenden Umgebungsvariablen zur Laufzeit des Programms. Durch seine Portabilität und Skalierbarkeit kann es sowohl auf Arbeitsplatzrechnern als auch auf Höchstleistungsrechnern eingesetzt werden.
In erster Linie ist OpenMP auf „shared-memory“ Computer ausgerichtet, kann jedoch durch Erweiterungen auch auf anderen Systemen eingesetzt werden. Des Weiteren ist ein hybrider Einsatz zusammen mit MPI möglich. Für die Anwendung in der CSE `iFluids` wurde ein solcher hybrider Ansatz untersucht.
Da bei OpenMP nicht der gesamte Code parallel ausgeführt wird, müssen die zu parallelisierenden Abschnitte durch entsprechende Anweisungen gekennzeichnet werden. Bevor eine so markierte Stelle erreicht wird, werden die zusätzlichen Threads gestartet; nachdem der Codeabschnitt durchlaufen wurde, werden die Threads wieder zusammengefügt.
Bei OpenMP findet keine direkte Kommunikation zwischen den Threads statt. Der Datenaustausch erfolgt über den Arbeitsspeicher des Systems. Bei NUMA Systemen ist des Weiteren darauf zu achten, dass zusätzliche OpenMP Threads auf Prozessorknoten, die direkten Zugang zum selben Arbeitsspeicher haben, gestartet werden. Sollte dies nicht geschehen, erfolgt jeder Speicherzugriff von nicht lokalen Threads über die Netzwerkverbindung, im Falle des HLRB II über NUMALink, was die Latenzzeiten massiv ansteigen lässt und einen Leistungseinbruch des Codes zur Folge hat. Der gleiche Effekt tritt auch auf, wenn auf Rechnern wie dem HLRB II von einem Prozess mehr Speicher benötigt wird, als lokal vorhanden ist. Auch dann muss Speicher in anderen Teilen des Systems allokiert werden; bei jedem Zugriff auf Daten, die in nicht lokalem Speicher abgelegt werden müssen, kommt es somit auch zu einem massiven Leistungseinbruch. Dies ist vor allem bei einer OpenMP Parallelisierung von großen Problemen zu beachten, da hierbei das Rechengebiet, anders als bei einer MPI-basierten Parallelisierungsstrategie, nicht unterteilt und auf die einzelnen Prozesse und deren lokalen Speicher verteilt wird. Somit bedarf bei solchen Problemen die Allokierung von Speicher und das Ablegen der Daten einer besonderen Aufmerksamkeit durch den Programmierer. Daten werden im Normalfall möglichst nahe bei dem Thread abgelegt, welcher sie zuerst „anfasst“. Es handelt sich also um einen Hardware-aware-Ansatz, bei dem im Idealfall möglichst hardwarenahe programmiert wird, um eine optimale Leistung zu erzielen.

Kapitel 7

Optimierung und Performance Benchmark auf dem HLRB II

Diese Arbeit entstand im Kontext von zwei Forschungsprojekten, wobei bei dem Projekt ComfSim [64] eine enge Partnerschaft mit dem Leibniz Rechenzentrum (LRZ) in Garching bei München bestand und somit eine Fokussierung auf den dort installierten, leistungsstarken Bundeshöchstleistungsrechner HLRB II gegeben war. In diesem Kapitel wird der Höchstleistungsrechner vorgestellt und die Arbeiten zur Leistungsoptimierung der Simulationsumgebung im Allgemeinen und insbesondere für diese Hardware beschrieben. Hierbei wurden entsprechende Benchmark Rechnungen durchgeführt und ausgewertet.

7.1 Beschreibung des Benchmark Problems: Separatorenraum

Zur Evalierung der Performance der `iFluids`-Simulationsumgebung auf dem Bundeshöchstleistungsrechner HLRB II am Leibniz Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ) in Garching bei München wurde ein Beispiel mit einer realen Geometrie definiert. Konkret kam das Modell eines Separatorenraums aus dem Schiffbau zum Einsatz. Separatoren werden zur Ölaufbereitung benötigt und erzeugen eine hohe Abwärme, weshalb sich die Fragestellung der effizienten Kühlung des Raumes ergibt. Bei dem hier betrachteten Problem wird nur die Konvektion simuliert. Dieses Beispiel wird auch detailliert von van Treeck in [59], [66] beschrieben.

7.2 Technische Spezifikation des HLRB II

Bei dem Höchstleistungsrechner Bayern II (HLRB II) handelt es sich um einen stark parallelen „Supercomputer“, welcher im Neubau des Leibniz Rechenzentrums der Bayerischen Akademie der Wissenschaften in Garching bei München untergebracht ist. Der HLRB II ist auch derzeit (Stand November 2010) noch der größte shared-memory Rechner weltweit mit einem Gesamtarbeitsspeicher von 39 TByte.
Es handelt sich dabei um ein System aus der Altix 4700 Baureihe des Herstellers SGI[1] welches

[1] bis 1999: Silicon Graphics Incorporated

2006 in Betrieb genommen und im April 2007 auf den jetzigen Stand ausgebaut worden ist. Der Rechner verfügt über 9728 Prozessorkerne des Typs Intel Itanium2 Montecito Dual Core mit einer Taktrate von 1,6 GHz. Er erreicht damit eine theoretische Spitzenleistung von 62,3 TFlop/s und eine Linpack Leistung von 56,5 TFlop/s.
Das System ist in 19 Rechenpartitionen mit je 512 Prozessorkernen unterteilt. Die Prozessoren sind auf so genannten *blades* untergebracht, wobei sich auf jedem *blade* zwei Prozessorkerne befinden. Bei sechs der 19 Partitionen des Systems wurde die Anzahl der Prozessorkerne pro *blade* auf vier verdoppelt. Diese Partitionen werden als *high density* Partitionen bezeichnet, wohingegen es sich bei den restlichen 13 Partitionen um *bandwidth* Partitionen handelt. Die *high density* Partitionen wurden geschaffen, um die Gesamtanzahl der Prozessorkerne zu erhöhen ohne zusätzlich Grundfläche zu benötigen. Diese Partitionen haben jedoch den Nachteil, dass sich vier statt zwei Prozessorkerne die verfügbare Speicherbandbreite teilen. Dies macht sich v.a. bei Anwendungen mit häufiger und intensiver Speichernutzung negativ hinsichtlich der Gesamtperformance bemerkbar. Für solche Anwendungen sollte somit die Nutzung von *high density* Partitionen vermieden werden. Jeder Rechenkern besitzt lokal 4 GByte Arbeitsspeicher und auf jeder Partition läuft eine separate Betriebssystem Instanz; es kommt SLES[2] 10 zur Anwendung.
Die Kommunikation zwischen den *blades* erfolgt mittels NUMALink 4. Die maximale lokale Speicherbandbreite beträgt 8,5 GByte/s, welche von zwei bzw. vier Rechenkernen geteilt wird. Aufgrund der Konfiguration der NUMALink 4 Verbindungen (siehe Abbildung 7.1) reduziert sich die verfügbare Bandbreite jeweils beim Übergang von einer auf vier Partitionen und beim Übergang auf mehr als vier genutzte Partitionen. Anwendungen, die maximal 510 (bzw. 508 bei *high density* Partitionen) Kerne verwenden, können auf einer Partition laufen. Wird diese Grenze überschritten, wird ein Job auf mehr als eine Partiton verteilt, was aufgrund der vorgenannten Charakteristik der NUMALink Konfiguration zu einem Geschwindigkeitseinbruch, v.a. bei der Verwendung von teuren Kommunikationsaufrufen (beispielsweise kollektive MPI Aufrufe) führt. Im ungünstigsten Fall kann somit eine Simulation mit 511 Prozessen signifikant langsamer sein als die selbe Simulation ausgeführt auf 510 Prozessorkernen.

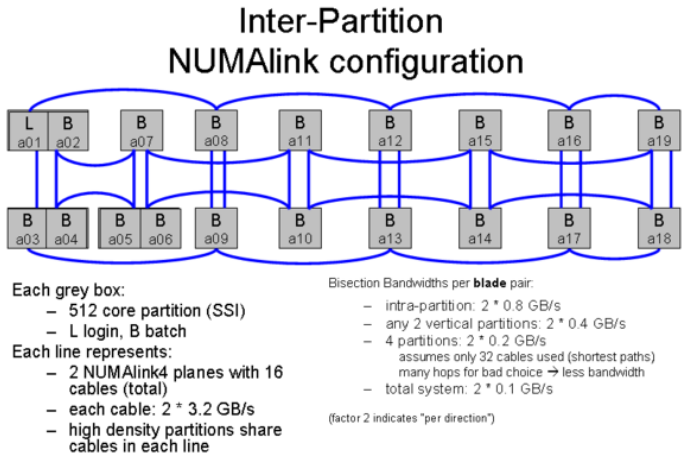


Abbildung 7.1: Konfiguration des NUMALink des HLRB II (Quelle: LRZ).

[2]SUSE Linux Enterprise Server 10

Der Großteil des Systems ist nicht direkt für den Benutzer zugänglich und nur über *batch processing* nutzbar. Für interaktive Anwendungen besteht jedoch die Möglichkeit, auf einem kleinen Teil des Systems Rechenzeit mit direktem Zugang zu den reservierten Prozessorkernen über eine Kommandozeile zu erhalten.
Für die Maschine stehen insgesamt 600 TByte direkt angeschlossenen Festplattenspeichers und zusätzlich 60 TByte NAS[3] Speicher zur Verfügung.

7.2.1 Beschreibung des Geometriemodells

Das Modell enthält neben den eigentlichen Separatoren noch alle zusätzlichen Einbauten in diesem Maschinenraum, v.a. Rohrleitungen und ist somit sehr komplex (siehe Abbildung 7.2). Die Abmessungen betragen:

Abmessungen	
x-Richtung	12,66 m
y-Richtung	6,11 m
z-Richtung	4,737 m

Das Geometriemodell des Separatorenraums wurde in einem CAD-System erstellt und liegt im STL-Dateiformat vor und umfasst 295.899 Facetten. Die Geometriedatei wurde in einzelne Unterobjekte (solids) untergliedert, um die Definition der unterschiedlichen Randbedingungen zu ermöglichen.

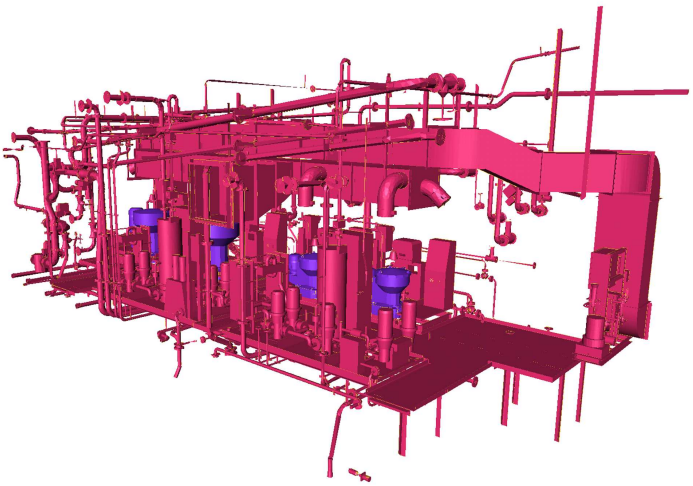


Abbildung 7.2: Geometrie des für den Performance Benchmark verwendeten simulierten Separatorenraums.

Die Wände sind nicht Bestandteil der Geometriedatei dieses Beispiels und werden durch die Standarddefinition der Simulationsumgebung gesetzt. Es wird ein rechteckiger Raum angenommen.

[3] Network Attached Storage

7.2.2 Beschreibung der Randbedingungen

Der Separatorenraum wurde mit Temperaturrandbedingungen versehen. Hierbei wurden zwei Temperaturen unterschieden. Den Separatoren wurde eine Temperatur von 60°C und den restlichen Einbauten eine Temperatur von 30°C zugewiesen.
Die Wände wurden mit einer Bounceback Randbedingung und als adiabat definiert. Ein- und Ausströmrandbedingungen sind in diesem Beispiel gleich Null definiert, d.h. der Raum wird als geschlossen angenommen.

7.2.3 Beschreibung der Simulationsumgebung (HLRB2)

7.2.3.1 Setup der Simulation

Für die Performanceanalyse wird in diesem Kontext hauptsächlich die Leistung der Simulationskomponente untersucht.
Hierzu wurde die Visualisierungsapplikation durch eine Dummy-Applikation ersetzt, die nur die Aufgaben des Einlesens der Geometriedatei übernimmt. Der hybride thermische Simulationskern mit Master- und Slave-Simulationsknoten entspricht der in der CSE verwendeten Version, d.h. es kommt eine scheibenartige Gebietszerlegung zur Anwendung.
Um den Einfluss der Kommunikation von Ergebnisdaten von der Simulation an die Visualisierung zu minimieren, wurde die Updaterate, welche bestimmt, in welchem Zyklus Daten an die Visualisierung übertragen werden, auf einen genügend hohen Wert gesetzt.
Die einzelnen Simulationsläufe werden nicht interaktiv, sondern über das batch-queuing-System des HLRB II abgewickelt.

7.2.3.2 Speed-up, strong scaling/weak scaling

Bei der Evaluierung der Skalierbarkeit eines Problems auf Hochleistungsrechnern werden zwei verschiedene Arten unterschieden. Zum einen in das sog. *strong scaling*, bei welchem untersucht wird, wie sich die Rechenzeit unter Erhöhung der Anzahl der Prozessoren p bei gleich bleibender Problemgröße N_{DOF}, d.h. Anzahl der Freiheitsgrade (DOF), verhält und in das sog. *weak scaling*. Hierbei bleibt die Problemgröße je Prozessor konstant, d.h. die Gesamtproblemgröße ist p mal der Problemgröße auf einem Prozessor.
Die Beschleunigung der Berechnung bei einer *strong scaling* Betrachtung wird als *speed-up* bezeichnet, wohingegen man bei einer *weak scaling* Betrachtung der Skalierbarkeit von *scale-up* spricht [57]
Der *speed-up* in Abhängigkeit von der Anzahl der Prozessoren ist definiert als

$$S(p) = \frac{T(1)}{T(p)} \tag{7.1}$$

wobei $T(k)$ die Rechenzeit des Problems jeweils auf k Prozessoren ist. Es gilt: $0 \leq S(p) \leq p$.
Die parallele Effizienz $E(p)$ ist definiert als der normalisierte *speed-up*:

$$E(p) = \frac{S(p)}{p} \tag{7.2}$$

Entsprechend gilt $E(p) \leq 1$, wobei man bei $S(p) = p$ bzw. $E(p) = 1$ von einem idealen *speed-up* spricht.
Der *scale-up* in Abhängigkeit von der Anzahl der Prozessoren und der Problemgröße ist entsprechend definiert als ([57]):

$$S^*(p, N_{DOF}) = \frac{T(1 \cdot N_{DOF})}{T(p \cdot N_{DOF})/p} \tag{7.3}$$

Der *scale-up* wird nach [19] bei $S^* = 1$ als linear bezeichnet. $S^* < 1$ bedeutet eine Verminderung des *scale-up* (sublinear) und bei $S^* > 1$ ist der Geschwindigkeitszuwachs superlinear.
Eine Beschränkung des *speed-up* ergibt sich aus dem seriellen Anteil, da nicht alle Operationen parallelisiert werden können (z.B. I/O).
In der Realität limitiert auch der mit wachsender Anzahl der verwendeten Prozessoren p zunehmende Kommunikations-Overhead die Skalierbarkeit der Applikation. Der Kommunikations-Overhead kann durch eine geschickte Wahl der Gebietszerlegung für das Rechengebiet optimiert werden. Die parallele Leistung eines Simulationscodes wird auch von einer gleichmäßigen Verteilung der Rechenarbeit auf alle verfügbaren Prozessoren bestimmt. Im konkreten Fall der CSE `iFluids` betrifft dies zum Einen die Verteilung der Voxel auf die einzelnen Prozesse, zum Anderen aber auch das Verhältnis von Fluid- zu „Objekt“-Voxel auf einem Prozess, da die teuren Rechenoperationen nur für Fluid-Voxel ausgeführt werden. Bei einer ungünstigen Verteilung kann somit ein einzelner Prozess zur Ausbremsung der anderen Prozesse führen, da nach jedem Zeitschritt die Ergebnisse der Teilgebiete wieder auf dem Masterknoten gesammelt werden und im Propagationsschritt[4] die Randknoten mit den benachbarten Prozessen ausgetauscht werden müssen, bevor mit der Berechnung des nächsten Zeitschritts fortgefahren werden kann.

7.3 Evaluierung der Optimierungsmöglichkeiten

Performance Optimierung in `iFluids`

Während der Entwicklungsphase wurden am Simulationscode und an der Kommunikation zwischen Simulation und Visualisierung zahlreiche Optimierungen vorgenommen und Messungen hierzu durchgeführt. Als eine erste Optimierung wurde bereits in der früher Entwicklung durch Wenisch und van Treeck [70] ein verteiltes Softwarekonzept eingeführt, welche in einem signifikanten Leistungszuwachs resultierte. Hierbei wurde auf der Simulationsseite ein sogenannter Kollektor-Knoten (SIM-M) eingeführt, über welchen die Kommunikation mit der Visualisierung abgewickelt wird. Dieser Knoten ist außerdem für die Netzgenerierung zuständig, verteilt die Daten auf die (Slave-) Berechnungsknoten (SIM-S) und sammelt die Ergebnisse, welche an die Visualisierung übertragen werden sollen wieder von den SIM-S Knoten ein. Dieses ermöglicht eine Überlappung zwischen Kommunikation und Berechnung. Abbildung 7.3 zeigt den Unterschied zwischen zwei Herangehensweisen, mit und ohne Kollektor-Knoten.
Als „Flaschenhals“ bei der Kommunikation kann die Übertragung von Ergebnissen zwischen Hochleistungsrechner und Visualisierungskomponente identifiziert werden. Um dieses Verhalten quantifizieren zu können, wurden von *Wenisch* zahlreiche Messungen durchgeführt. Abb.

[4] In diesem Schritt der Simulation werden Teilchenverteilungen im Rechengitter zum nächsten Knoten propagiert.

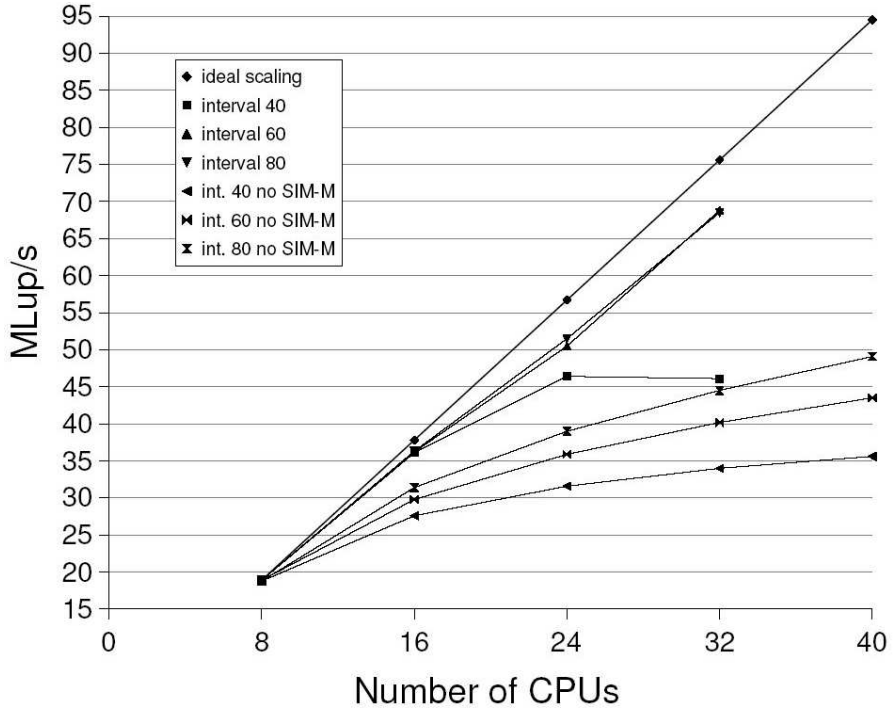


Abbildung 7.3: Skalierbarkeit für verschiedene Updateraten (HLRB I). Leistungsmessung in MLup/s für den Visualisierungsprozess. Schlechte Effizienz, wenn kein Masterknoten (SIM-M) verwendet wird. (Quelle: *Wenisch*)

7.4 zeigt die Ergebnisse eines von *Wenisch* durchgeführten Benchmarks für das HLRB I System Hitachi SR-8000 (Leibniz Rechenzentrum, Garching) und für ein SGI Altix 3700 System (Sara Computing Centre, Amsterdam), dem Vorläufer des Bundeshöchstleistungsrechners Altix 4700 am LRZ. Hierbei wurde gemessen, wie sich die Leistung verhält, wenn die Visualisierung ebenfalls auf dem Hochleistungsrechner durchgeführt wird, d.h. Verbindungen über das Netzwerk umgangen werden (offline Performance). Die Abb. 7.4 zeigt in diesem Fall einen möglichen Leistungszuwachs um 70% zwischen den beiden Systemen. Diese Ergebnisse lassen sich auch auf das HLRB II System übertragen, wobei sich der „Flaschenhals“ hier durch die nochmals höhere Netzwerkbandbreite, v.a. bei Nutzung des Visualisierungssystems rvs1 am LRZ, zusätzlich entschärft hat.
Ein weiterer wichtiger Optimierungsschritt war die Optimierung der algorithmischen Performance des seriellen Codes auf dem HLRB II System. Hierbei wurden zum Einen Code Strukturen optimiert als auch spezielle Optimierungen, beispielsweise über Compiler-Direktiven, für die im HLRB II verbaute Hardware, Intel Itanium 2, und den verwendeten Intel Compiler vorgenommen. Durch diese Maßnahmen konnte die Performance von ca. 450 MFlop/s (unoptimierter Code) auf ca. 1400 MFlop/s gesteigert werden. Dies entspricht ungefähr 23% der Peak Performance einer CPU und ca. 4 MLup/s[5], was angesichts der aufwändigeren Berechnungen durch den verwendeten MRT-Ansatz und das implementierte thermische Modell einen sehr guten Wert darstellt.
Ergänzend wurde auch noch die parallele Performance (sowohl *scale-up* als auch *speed-up*)

[5]Mega lattice updates per second. Dieses Geschwindigkeitsmass gibt an, wie viele (Millionen) Knoten pro Sekunde während der Berechnung durchlaufen werden können.

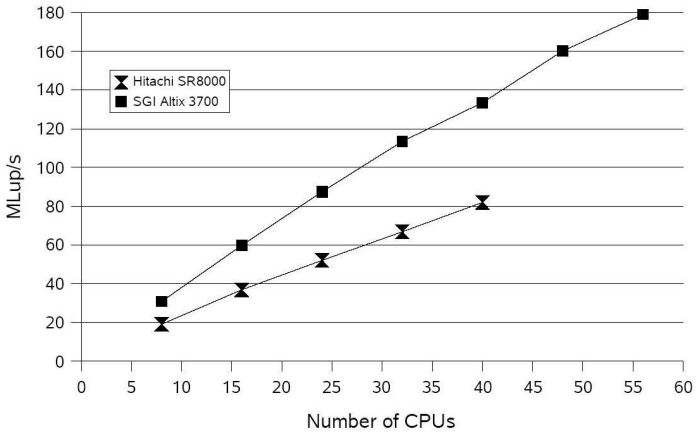


Abbildung 7.4: Offline performance auf Hitachi SR8000 und SGI Altix 3700, gemessen für den Visualisierungsprozess [69].

auf dem System untersucht. Hierbei stellte sich unter anderem heraus, dass aufgrund der aufwändigen Kommunikation des Codes auf dem HLRB II System unbedingt eine *bandwidth* Partition und keine *density* Partition zu verwenden ist, da bei Verwendung der Letzteren die Performance auf ca. 65% einbricht (siehe Abb. 7.5). Dieses Verhalten lässt sich auf die Kommunikation über den Numa-Link zurückführen, da auf *density* Partitionen vier statt zwei Cores über diese Verbindung kommunizieren.

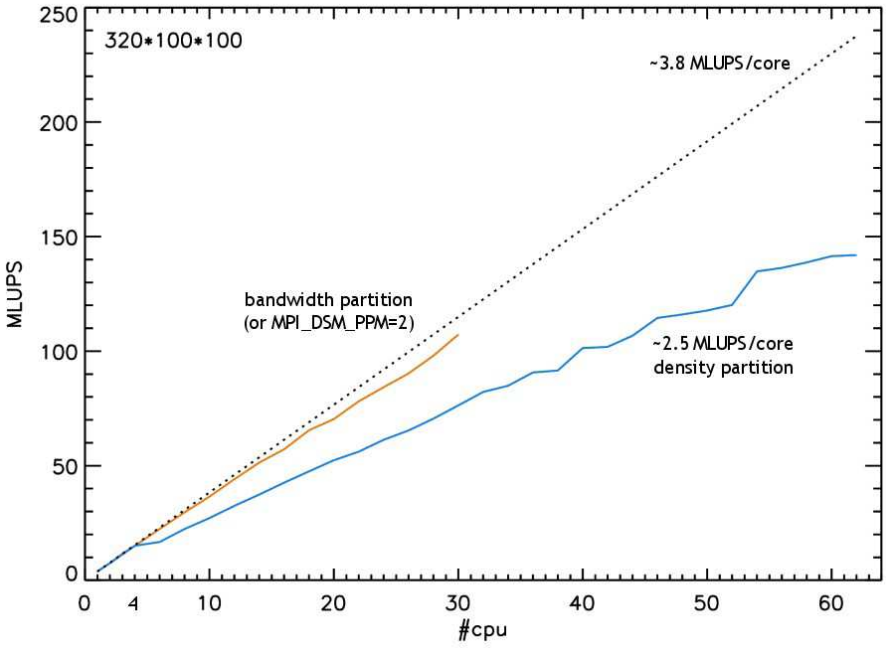


Abbildung 7.5: Skalierungsverhalten des Codes auf *bandwidth* und *density* Partitionen des HLRB II.

7.3.1 Optimierung des Simulationskernels

Die Analyse und Optimierung der Programmstruktur des Rechenkerns wurden an einem seriellen Code durchgeführt, da so die Ergebnisse ohne Einflüsse aus der Parallelisierung, d.h. beispielsweise Kommunikation, betrachtet werden konnten.
Die Ausgangsperformance der Simulation lag unoptimiert bei ca 450 MFlop/s. Verglichen mit einer Spitzenleistung von 6,4 GFlop/s eines Kerns des im HLRB II verbauten Itanium II Prozessors und einer mittleren Applikationsperformance auf dem System von ca. 10% der Spitzenleistung, d.h. 640 MFlop/s, stellt diese Leistung eines sehr schlechten Wert dar. Dies trifft vor allen dann zu, wenn man berücksichtigt, dass laut Erfahrungen des LRZ bei Lattice-Boltzmann Codes im Allgemeinen mehr als 10% der Spitzenleistung erreichbar sind.
Alle Werte wurden komplett nicht-interaktiv ohne Datentransport zum Visualisierungsfrontend und mit einem leeren Rechengitter ermittelt. Die Arbeiten wurden zudem in einer engen Kooperation mit dem Leibniz Rechenzentrum, den Betreibern des HLRB II, ausgeführt.

7.3.1.1 Compiler Direktiven

In einem ersten Schritt wurden noch keine großen Änderungen am Code vorgenommen. Zuerst wurde nur konsequent von den Optimierungsoptionen des Intel Compilers Gebrauch gemacht. Anschließend wurde mittels eines Profilings analysiert, wo die *hot spots*, d.h. die Stellen im Code, an denen die meiste Rechenzeit verbraucht wird, liegen. Dies sind die Funktionen für die Kollision, die Propagation und für das Lösen der Temperaturgleichung.
Es wurde überprüft, ob für alle wichtigen, teuren Schleifen das *Software Pipelining* benutzt wurde. Dort, wo dies nicht der Fall war, wurden entsprechende Compiler Direktiven eingebaut. Um die Vektorisierung durch den Compiler zu erzwingen, wurde beispielsweise die Direktive *ivdep* des Intel Compilers genutzt. Zusätzlich wurden noch, wenn möglich, sämtliche Anweisungen aus dem Code entfernt, die das *Software Pipelining* verhindern. Hierzu gehören beispielsweise *print*-Anweisungen oder *if*-Abfragen.
Schleifen, bei denen ein so genannter *register spill* auftrat, wurden aufgeteilt. Bei einem *register spill* sind mehr Variablen aktiv als im Register der CPU vorgehalten werden können. Deshalb müssen Variablen zwischen dem Register und dem Speicher transferiert (*spill*) werden. Dies beeinträchtigt die Leistung, da der Zugriff auf Variablen im Register schneller ist als der Zugriff auf den Speicher.
Ergänzend wurden die Funktionen auf überflüssige Operationen untersucht, welche ggf. entfernt wurden.
Diese Optimierungen und das Nutzen der Optimierungsmöglichkeiten des Compilers beschleunigten die Simulation bereits auf ca. 800 MFlop/s, was einer Steigerung um ca. 77% entspricht.

7.3.1.2 Verbesserung der Codestruktur

Zusätzlich zu oben genannten Optimierungen wurde auch die Codestruktur analysiert und optimiert, um eine weitere Geschwindigkeitssteigerung zu erzielen.
Der Kollisionsschritt des Lattice-Boltzmann Algorithmus ist von Rechenoperationen und der Propagationsschritt stark von Speicheroperationen dominiert. Hieraus ergibt sich ein Potential, manche Aufgaben parallel abwickeln zu können. Aus diesem Grund wurden sogenannte Ghost-Nodes, wie von Wenisch [68] für einen BGK-Code vorgeschlagen, integriert. Diese

ermöglichen eine Zusammenlegung von Kollision und Propagation und dadurch eine effizientere Programmstruktur.
Des Weiteren wurde festgestellt, dass *if*-Anweisungen innerhalb der *energy* Schleife, d.h. bei der Lösung der Temperaturgleichung, ein *Software Pipelining* verhindern. Aus diesem Grund wurde die Schleife umgeschrieben, um dies zu lösen. Diese stören aufgrund der *predicate* Register der Intel Itanium Architektur die Vektorisierung nicht. Durch diese Maßnahme ließ sich diese Funktion um den Faktor 3 beschleunigen. Für den gesamten Code bedeutet dies, dass nur noch ca. $\frac{1}{6}$ statt wie vorher die Hälfte der Gesamtrechenzeit des Simulationskerns auf diese Funktion entfällt.
Als dritte große Änderung der Codestruktur wurde das Datenlayout zur Speicherung des Rechengitters im Arbeitsspeicher modifiziert. Ursprünglich handelte es sich hierbei um ein *Array of Structures* (AoS). Zu Testzwecken wurde diese in eine *Structure of Arrays* geändert. Es zeigte sich jedoch, dass aufgrund des großen Cache-Speichers der Itanium CPUs des HLRB II dies kaum eine Rolle spielt. Aus diesem Grund wurde die ursprüngliche Datenstruktur, d.h. AoS, beibehalten, da diese für den Kollisionsschritt optimal ist.
Die serielle Leistung auf einer CPU der letzten Version im Optimierungsschritt betrug ca. 1400 MFlop/s oder 4 MLup/s, was knapp 23% der Spitzenleistung der CPU entspricht. Diese Messung wurde mit einem leeren Gitter mit der Größe 100^3 Voxel gemacht. Andere LBM Codes erreichen auf dem Itanium 2 Prozessor 6–7 MLup/s (Donath et al. 2005) oder 8,5 MLup/s (BEST Code, Wellein et al. 2005); beide nutzen ein LBGK19 Modell. Mit 4 MLup/s ist die Leistung zwar im Vergleich etwas geringer, jedoch ist zu berücksichtigen, dass im Unterschied zu den anderen Codes der hier vorgestellte Simulationskern den MRT Ansatz nutzt und einen thermischen Teil hat, was beides zu einer deutlich größeren Anzahl an Operationen pro Voxel und Zeitschritt führt.
In der Version des Rechenkerns für die CSE können jedoch nicht alle oben genannten Optimierungen eingesetzt werden, da manche Optimierungsfunktionen des Intel Compilers (im konkreten Fall die *interprocedural optimization* – ipo) zu nicht nachvollziehbaren Stabilitätsproblemen an anderen Programmstellen der CSE führt.

7.4 Nutzung eines hybriden MPI/OpenMP-Ansatzes

Die gleichen Rechnungen wurden zusätzlich zu den parallelen Rechnungen nur mit MPI auch mit einem hybriden Ansatz MPI/OpenMPI durchgeführt. Dies sollte die mögliche zusätzliche Leistungssteigerung v.a. auf dem Höchstleistungsrechner HLRB II untersuchen.
Hierzu wurden Schleifen innerhalb der Kollisions-Routinen des Simulationskerns mittels OpenMP Compilerdirektiven zusätzlich parallelisiert.
Die Kollisionsfunktion des Lattice-Boltzmann Algorithmus wurde gewählt, da der größte Rechenzeitanteil der Simulation hierauf entfällt. Des Weiteren bieten die darin vorhandenen Schleifen einen guten Ansatzpunkt für eine zusätzliche OpenMP Parallelisierung.
Die Kollision wird an jedem Fluid Knoten des Rechengebiets lokal ausgeführt und hat somit keine Abhängigkeiten zu benachbarten Knoten. Da es sich um ein dreidimensionales Feld handelt, sind in dieser Funktion drei große Schleifen für jede Raumrichtung des Feldes vorhanden. In einem ersten Schritt wurden die äußere Schleife, dies entspricht der Laufvariablen für die x-Richtung, komplett als mit OpenMP zu parallelisierende Region definiert. Die *for*-Schleife selbst wurde mittels eines entsprechenden OpenMP Befehls mit dynamischer Threadaufteilung

parallelisiert.
Bei der Definition des OpenMP-parallelen Bereichs musste darauf geachtet werden, dass nur die globalen Datenfelder als *shared* Variablen in den einzelnen Prozessen verfügbar sind. Alle weiteren Variablen des Algorithmus wurden, genauso wie die Laufvariable der Schleife, als private Variable definiert. Dies ist nötig, um Konflikte bei Speicherzugriffen und daraus resultierende Speicherzugriffsfehler bzw. von den Prozessen gegenseitig überschriebene Ergebnisse zu vermeiden. Die Parallelisierung von Schleifen mit OpenMP kann generell nur erfolgen, wenn keine schleifenabhängigen Abhängigkeiten auftreten. Dies bedeuet, dass in der Schleife berechnete Variablen nicht von anderen Variablen, welche innerhalb dieser Schleife berechnet werden, abhängen dürfen. Deshalb ist im Vorfeld eine entsprechende *Dependency Analysis* durchzuführen.
Da es sich bei den im HLRB II verbauten Itanium Prozessoren um Dual Core Modelle handelt, liegt eine hybride MPI/OpenMP Parallelisierung mit zwei OpenMP Threads pro MPI Prozess nahe. Im Idealfall wird somit auf jeder CPU nur ein MPI Prozess platziert; der zweite Kern der CPU steht dann für den zusätzlichen OpenMP Thread zur Verfügung. Sollte bei der Definition der OpenMP Parallelisierung hierauf nicht geachtet werden, besteht die Gefahr, dass die OpenMP Threads ungünstig auf der Maschine platziert werden und sich Leistungseinbußen durch Zugriffe auf nicht lokalen Speicher ergeben.
Bei der Vorbereitung eines entsprechenden Rechenjobs für den Höchstleistungsrechner HLRB II ist aus diesem Grund auf eine korrekte Platzierung der Prozesse zu achten. Dies geschieht über eine geeignete Definition der entsprechenden OpenMP Umgebungsvariablen. Durch die entsprechende Option wird beispielsweise erreicht, dass das System bei der Zuweisung der MPI Prozesse auf Prozessorkerne dazwischen Kerne für die OpenMP Threads frei lässt. Durch Verwendung des *omplace* Befehls beim Starten des hybriden Rechenjobs wird sichergestellt, dass die OpenMP Threads nicht auf einem einzelnen Prozessorkern konzentriert werden. Sollte dies nicht berücksichtigt werden, ist mit einem Einbruch der Applikationsperformance zu rechnen.
Der Master-Prozess der Simulation und der Visualisierungsprozess wurden nicht bei der OpenMP Parallelisierung berücksichtigt, da dort aufgrund der Programmstruktur keine Leistungssteigerung zu erwarten ist.
Zusätzlich zur hybriden Parallelisierung mit zwei OpenMP Threads pro MPI Prozess wurde auch der Einsatz von mehr Threads untersucht, um den Einfluss der Aufspaltung auf zwei oder mehr CPUs zu quantifizieren. Hierbei kamen vier und acht Threads pro MPI Prozess zum Einsatz.
Da die maximal mögliche Anzahl an MPI Prozessen durch die verwendete scheibenbasierte Gebietszerlegung des Rechengebiets stark eingeschränkt ist, stellt der Einsatz einer hybriden Parallelisierung mit Hilfe von OpenMP eine interessante Option dar. Hiermit lässt sich v.a. für die teure Kollisionsfunktion die Anzahl der nutzbaren Prozessorkerne nochmal deutlich erhöhen, ohne Änderungen an der Gebietszerlegung vornehmen zu müssen.

7.5 Studien zur Skalierbarkeit

Im nachfolgenden Abschnitt werden die Benchmark Rechnungen für die Untersuchung der Leistungsfähigkeit und der Skalierbarkeit auf parallelen Hochleistungsrechnern dargestellt und die gewonnenen Ergebnisse erläutert.

7.5.1 Problemgröße

Die Anzahl der Voxel, und damit der Freiheitsgrade (DOF) des Systems, wurde zur Betrachtung des *strong scalings* bei 3.035.520 konstant gehalten, wohingegen für die Analyse des *weak scalings* die Anzahl der DOFs pro Rechenprozess konstant gehalten wurde. Es wurde einmal der *scale-up* mit einem kleinen Problem, welches klein genug ist um auf jedem Simulationsknoten in den Cache der CPU zu passen, und einem größeren Problem, welches nicht mehr vollständig in den Cache der CPU passt, untersucht. Als kleines Problem wurden 10.000 Voxel pro Prozessorkern und als großes Problem 300.000 Voxel pro Prozessorkern gewählt.
Aufgrund der Parallelisierungsstrategie des Simulationskerns und den Abmessungen der verwendeten Testgeometrie, ist mit diesem Modell keine Untersuchung des *weak scalings* möglich, da mit den festgelegten Größenabmessungen eine konstante Anzahl an Voxeln pro Rechenprozess bei variierender Prozessanzahl nicht möglich ist.
Deshalb kamen für die Untersuchung des *weak scaling* totale Problemgrößen (angegeben mit der Anzahl der jeweils eingesetzten Anzahl an Rechenprozessen), wie sie in Tabelle 7.1 aufgetragen sind, zur Anwendung.

Proc	Voxel kl. Problem	Voxel gr. Problem
1	10000	300000
2	20000	600000
3	30000	900000
5	50000	1500000
...		
10	100000	3000000
20	200000	6000000
...		
60	600000	18000000

Tabelle 7.1: Anzahl der Freiheitsgrade bei der Untersuchung des *weak scaling*.

Um dennoch den Einfluss von Objekten zu berücksichtigen, wurde ein einfaches Geometrieobjekt in der Mitte des Testgebiets platziert. Die Resultate der Benchmark-Rechnungen und deren Auswertung sind in Abschnitt 7.6.3 dargestellt.

7.5.2 Benchmark Rechnungen ohne OpenMP

Zuerst wurde das Problem mit dem Standard Rechenkern der CSE ohne zusätzliche Parallelisierung mittels OpenMP gerechnet. Hierbei wurde sowohl das *strong scaling* als auch das *weak scaling* der Applikation betrachtet.
Die Rechnungen wurden einmal auf *density* Partitionen und einmal auf *bandwidth* Partitionen durchgeführt. Es kam jeweils die durch den Hersteller SGI speziell auf die Hardware der Altix 4700 optimierte MPI Implementierung und teilweise zu Vergleichszwecken auch die Intel MPI Implementierung in Version 3.2 zum Einsatz. Dies erfolgte, um die Leistungssteigerung durch die maschinenspezifische Implementierung quantifizieren zu können.
Als Compiler wurde der Intel C++ Compiler eingesetzt, da dieser speziell für die Verwendung mit Intel Itanium Prozessoren optimiert ist und somit die beste Codeoptimierung erwarten ließ.

Begonnen wurde die Performancemessung mit einem „quasi“ seriellen Problem. Hierbei wurden für ein Problem mit 3.035.520 Voxel mit dem Altix spezifischen MPI 1,78 MLup/s und dem Intel MPI 1,74 MLup/s erzielt. Quasi seriell bedeutet in diesem Fall, dass nur ein MPI Prozess für die Berechnungsschleife genutzt wurde; das Rechengebiet wurde somit nicht aufgeteilt, es mussten keine Daten zwischen benachbarten Gebieten ausgetauscht werden. Insgesamt wurden jedoch drei Prozesse benötigt: Ein Dummy-Visualisierungsprozess, der Simulations-Masterprozess sowie der eigentliche Rechenprozess.
Für die Betrachtung des *speed-up* wurde nun die Anzahl der MPI-Prozesse erhöht. Um eine bessere Vergleichbarkeit zu ermöglichen, werden die Ergebnisse ebenfalls in MLup/s angegeben. Zusätzlich wurden die Ergebnisse noch auf einen Prozess normalisiert. Eine Zusammenfassung für die Version mit Intel MPI ist in Tabelle 7.2 gegeben.

Anzahl der MPI Rechenprozesse	MLup/s total	MLup/s normalisiert
1	1,74	1,74
2	3,09	1,54
4	5,87	1,47
8	8,99	1,12
16	16,20	1,01
32	27,72	0,87
64	43,78	0,68
127	66,00	0,52

Tabelle 7.2: Speed-up bei Verwendung von Intel MPI.

Zum Vergleich sind nachfolgend in Tabelle 7.3 für ausgewählte Beispiele die normalisierten Ergebnisse mit den beiden MPI Implementierungen gegenüber gestellt.

	Performance mit	
Anzahl der MPI Rechenprozesse	Altix MPI [MLup/s]	Intel MPI [MLup/s]
1	1,78	1,74
2	1,535	1,54
4	1,455	1,47
32	0,97	0,87
59	0,75	0,66

Tabelle 7.3: Gegenüberstellung der Performance bei Verwendung von Altix und Intel MPI.

Der Vergleich für beide Versionen wurde mit bis zu 60 Simulations-Prozessen (Sim-Prozessen), d.h. 59 Rechenprozesse und ein Masterprozess durchgeführt. Die oben dargestellten Ergebnisse wurden auf einer *density* Partition des HLRB II gewonnen. Ein Vergleich zwischen *density* und *bandwidth* Partition ist in Diagramm 7.6 aufgetragen. Es wird die totale Leistung in MLup/s dargestellt.
Maximal wurde die Leistung unter Verwendung von 127 Rechenprozessen untersucht. Eine noch stärkere Parallelisierung war für dieses Problem aufgrund der verwendeten scheibenartigen Gebietszerlegungstechnik nicht möglich, da bei einer scheibenartigen Zerlegung die maximale Anzahl an Prozessoren erreicht ist, wenn eine Scheibe nur noch ein Voxel dick ist.
Für eine niedrige Anzahl an Prozessoren wurde die Leistungsfähigkeit dieser Simulation auch auf einer modernen acht-Kern-Workstation getestet. Die Workstation ist mit zwei Intel Xeon

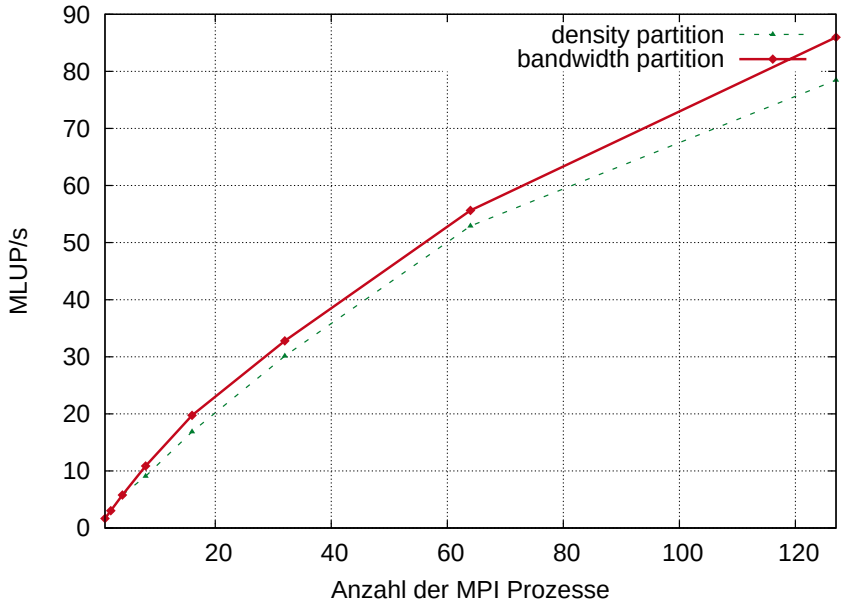


Abbildung 7.6: Gegenüberstellung Leistung auf *density* und *bandwidth* Partition.

Prozessoren vom Typ W5590 bestückt und besitzt 24GB Arbeitsspeicher. Da die Maschine nur acht physikalische Kerne besitzt, vier je CPU, konnte maximal eine Parallelisierung in sechs einzelne Gebiete erfolgen, da jeweils ein Kern für den Sim-Master und die Visualisierungssubstitution benötigt werden. Es wurde der Intel Compiler zusammen mit einer eigens für diese Maschine kompilierten OpenMPI Version verwendet. Die Ergebnisse sind in Tabelle 7.4 aufgelistet.

Anzahl der MPI Rechenprozesse	MLup/s total	normalisiert
1	4,52	4,52
2	9,06	4,53
4	12,90	3,23
6	16,73	2,79

Tabelle 7.4: Leistung der Simulation auf einer Intel Xeon Workstation.

Hierbei ist auffällig, dass der Code für wenige Prozesse auf der Xeon Workstation eine wesentlich höhere Leistung als auf dem HLRB II erzielt. Die Erklärung hierfür ist in mehreren Punkten zu suchen. Zum Einen handelt es sich hier um eine einzelne Maschine, auf welcher die MPI Prozesse zur Kommunikation nicht auf eine Netzwerkverbindung zwischen den Prozessoren angewiesen sind. Die Kommunikation erfolgt einfach über das *loop back device* des Rechners. Zum Anderen handelt es sich bei den Prozessoren um neuere Modelle mit einer höheren Spitzenleistung. Die verbaute Xeon CPU hat im Vergleich zur Itanium CPU des HLRB II eine um mehr als zweifach höhere Leistung pro Kern (13,32 GFlop/s zu 6,4 GFlop/s).

7.5.3 Benchmark Rechnungen mit OpenMP

Auch für die hybride Parallelisierung kam die Kombination aus herstellerspezifischem SGI MPI bzw. Intel MPI und Intel C++ Compiler zur Anwendung.

Bei einem ersten Test wurde festgestellt, dass die Geschwindigkeit bei Verwendung von zwei OpenMP Threads pro MPI Prozess nicht wie eigentlich zu erwarten wäre zunimmt, sondern abnimmt. Ein weiterer Geschwindigkeitsverlust tritt bei einer weiteren Erhöhung der OpenMP Threads pro MPI Prozess auf vier bzw. acht auf. Dies war jedoch in dieser Form zu erwarten, da jede CPU des HLRB II nur zwei Kerne besitzt und bei vier bzw. acht Threads diese auf zwei bzw. acht CPUs verteilt werden. Da das Rechengitter jedoch lokal im Speicher der CPU auf der auch der MPI Prozess läuft abgelegt ist, kommt es zu nicht lokalen Speicherzugriffen über den NUMALink der Maschine, was die Gesamtgeschwindigkeit des Codes negativ beeinflusst. Auch hier wurden beide MPI Versionen miteinander verglichen. Es zeigte sich, dass die Kombination aus Intel MPI und Intel Compiler meist eine deutlich geringere Leistung zeigte als die Kombination aus herstellerspezifischem Altix MPI und Intel Compiler. Für einen MPI Rechenprozess mit zwei OpenMP Threads in der hybrid zu parallelisierenden Programmregion wurden mit Altix MPI 0,43 MLup/s und mit Intel MPI 0,64 MLup/s erzielt, wobei für zwei MPI Rechenprozesse mit Altix MPI bereits 1,48 MLup/s und mit intel MPI nur noch 1,19 MLup/s erzielt wurden.
Dieser Geschwindigkeitsunterschied bestätigte sich auch bei zunehmender Anzahl der MPI-Prozesse, wie in Diagramm 7.7 dargestellt.

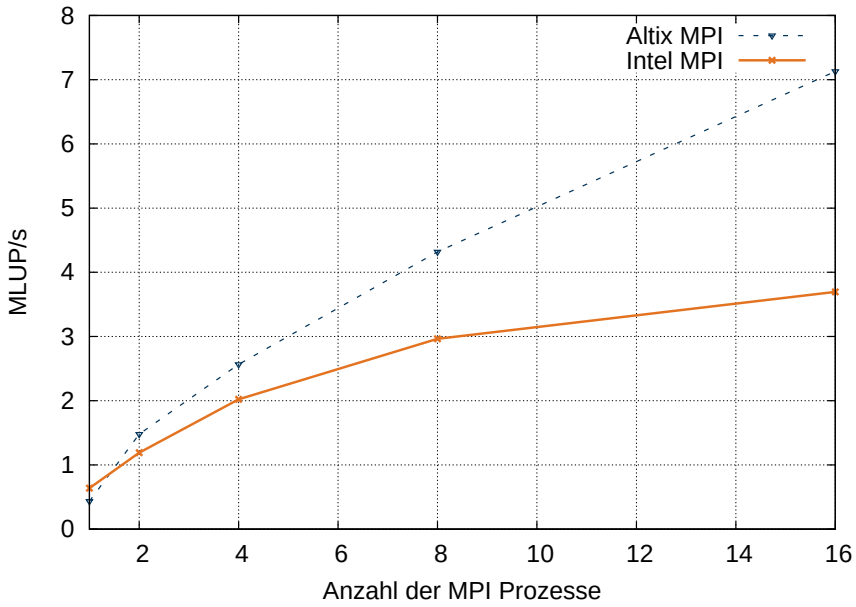


Abbildung 7.7: Vergleich Intel und SGI MPI bei hybrider MPI/OpenMP Parallelisierung.

Aufgrund des Geschwindigkeitsvorteils der Kombination Intel Compiler und Altix MPI wird nachfolgend nur noch diese im Detail betrachtet. Es wurden aufgrund der Dualkern-Prozessoren immer zwei Threads pro MPI Prozess verwendet. Die Ergebnisse für die einzelnen untersuchten Fälle sind in nachfolgender Tabelle aufgelistet:
Hierbei wurde die Gesamtleistung in MLup/s und die normalisierte Leistung, einmal bezogen auf die Anzahl der MPI Prozesse und einmal bezogen auf die Gesamtanzahl der verwendeten Rechenkerne angegeben (zwei Rechenkerne pro MPI Prozess). Die oben gelisteten Geschwindigkeitsmessungen sind auf einer *density* Partition des HLRB II gemessen worden. Beim Starten des Programms wurde der zusätzliche Befehl *omplace* zur Vermeidung der Konzentration

MPI Prozesse	MLup/s total	MLup/s pro MPI Task	MLup/s pro Rechenprozess
1	0,64	0,64	0,32
2	1,19	0,60	0,30
4	2,02	0,51	0,25
8	2,96	0,37	0,19
16	3,69	0,23	0,12

Tabelle 7.5: Leistung des hybrid parallelisierten Codes auf dem HLRB II.

von Threads auf einem CPU Kern verwendet. Da die Ergebnisse aber auf Probleme auf dem HLRB II hindeuten, wurden die Messungen auch bei Umgehung der Platzierung der Prozesse durch *omplace* vorgenommen. Hierbei stellte sich heraus, dass dies eine deutliche Leistungssteigerung brachte (siehe Tabelle 7.6).

MPI Prozesse	MLup/s total	MLup/s pro MPI Task	MLup/s pro Rechenprozess
1	1,01	1,01	0,51
2	1,84	0,92	0,46
4	3,51	0,88	0,44
8	5,74	0,72	0,36
16	9,89	0,62	0,31

Tabelle 7.6: Rechenleistung ohne Verwendung des *omplace* Befehles auf dem HLRB II.

Wie für die Betrachtung der reinen MPI-Leistung im vorangegangenen Abschnitt wurde die Geschwindigkeit auch auf einer Intel Xeon Workstation ermittelt. Die hierbei gemessenen Ergebnisse sind in Tabelle 7.7 darfestellt.

Anzahl der MPI Rechenprozesse	MLup/s total
1	6,25
2	10,1
3	13,8
4	13,4

Tabelle 7.7: Rechenleistung des hybrid parallelisierten Codes auf der Intel Xeon Workstation.

Auch hier macht sich bei dem Test auf der Xeon Workstation die wesentlich höhere Spitzenleistung der CPU zusammen mit den Geschwindigkeitsvorteilen durch die direkte Kommunikation zwischen den Prozessen ohne Nutzung eines Netzwerks sehr deutlich bemerkbar.
Es ist jedoch auffällig, dass der Code auf dem HLRB II nur etwa 10% der Leistung der Workstation zeigt. Ausgehend von den Ergebnissen ohne Verwendung von OpenMP wäre eine Leistung von ca. 40% der Workstationleistung zu erwarten. Der Grund hierfür liegt zum Einen teilweise auch an Overhead, welcher durch das Starten und Stoppen der zusätzlichen OpenMP Thread in jedem Zeitschritt entsteht. Zum Anderen macht sich hier die Architektur des Systems bemerkbar. Kommunikation erfolgt über die NUMALink-Schnittstelle und nicht mehr lokal auf einem Motherboard. Zusätzlich ist auch die Speicheranbindung und die Geschwindigkeit des Arbeitsspeichers in dem modernen Xeon System deutlich höher als auf dem HLRB II.

7.5.4 Vergleich density/bandwidth Partitionen des HLRB II

Wie in der Beschreibung der Maschine schon erwähnt, besteht diese aus so genannten *density* und *bandwidth* Partitionen. Da sich bei der *density* Partition die doppelte Anzahl an Prozessorkernen die NUMALink Anbindung teilen, ist auf diesen Partitionen mit einer geringeren Applikationsperformance des CSE Codes zu rechen, da es sich hierbei um eine kommunikationsintensive Anwendung handelt.
Bei der Definition eines batch-Jobs lässt sich spezifizieren, ob die Ausführung auf einer *bandwidth* oder *density* Partition gewünscht wird. Da diese Auswahl jedoch nicht bei allen Warteschlangen des HLRB II berücksichtigt wird, besteht zusätzlich noch die Option, dies mittels einer Umgebungsvariablen zu beeinflussen. Mit dieser Variablen lässt sich zwar nicht die Hardwareplattform wählen, jedoch kann man das Platzierungsverhalten der Anwendung auf den Blades steuern; man gibt die Maximalanzahl an Prozessen, die auf einem Blade ausgeführt werden sollen an, wobei diese kleiner oder gleich der Maximalanzahl der Prozessorkerne eines Blades sein kann.
Diese MPI-Option ist besonders wichtig, da beispielsweise die interaktive Shell des HLRB II immer auf *density* Knoten zugewiesen wird.
Zur Beurteilung des Einflusses der unterschiedlichen Hardwarestrukturen auf den beiden Partitionstypen wurden Untersuchungen mit den beiden MPI Implementierungen von Intel und von SGI gemacht. Es wurden Simulationen mit bis zu 127 MPI-Prozessen für den Rechenkern durchgeführt. Für die hybride MPI/OpenMP parallelisierte Version kamen maximal 254 CPU-Kerne zum Einsatz.
Ergänzend zu den Rechnungen mit reiner MPI Parallelisierung wurden ausgewählte Simulationen mit einer hybriden MPI/OpenMP Parallelisierung gemessen, um auch etwaige Geschwindigkeitsunterschiede für diese Option abschätzen zu können.
Im Diagramm 7.8 sind die Speed-up Kurven der Applikation für den Altix MPI Fall dargestellt. Einmal auf einer *bandwidth* Partition bzw. mit der entsprechende MPI Option zur Limitierung der Prozesse pro Blade und einmal auch auf einem *density* Blade gemessen. Angetragen sind der Speed-up der Simulation gegen die Anzahl der Prozesse.
Entsprechend wurden die Ergebnisse für die hybride Parallelisierung unter Verwendung von SGI Altix MPI ebenfalls in folgendem Diagramm 7.9 dargestellt.

7.6 Evaluierung der parallelen Performance

Im vorangegangenen Abschnitt wurden die einzelnen Leistungsmessungen ausführlich beschrieben. Anschließend werden diese nun gegenüber gestellt. Hierbei ist von besonderem Interesse, wie sich der Einsatz von OpenMP auf die Leistung auswirkt und wie sich die Wahl einer *density* oder *bandwidth* Partition auf die Leistung der Software auswirkt.

7.6.1 Auswertung der Benchmark Ergebnisse mit/ohne OpenMP

Beim Vergleich der Messergebnisse fällt sofort auf, dass die Leistung bei Anwendung eines hybriden Parallelisierungskonzepts mit OpenMP in der vorgestellten Form zu einem massiven Leistungseinbruch anstatt einer Leistungssteigerung führt. Dies ist in Abbildung 7.10 anschaulich sichtbar. Es werden jeweils die Ergebnisse mit der Altix MPI Implementierung

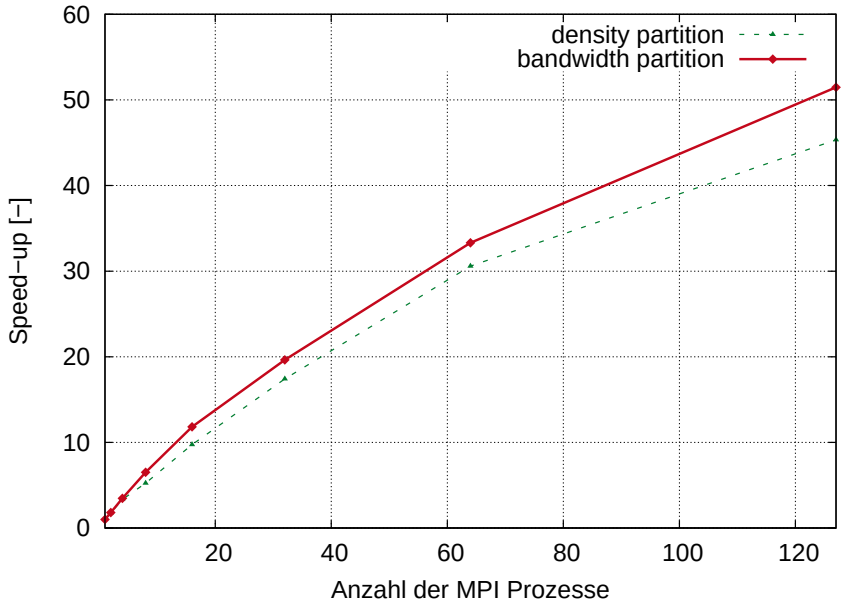


Abbildung 7.8: Gegenüberstellung der Leistung auf *density* und *bandwidth* Partitionen. Nur MPI.

dargestellt. Aufgetragen sind die Anzahl der MPI Rechenprozesse gegen die normalisierte Leistung in MLup/s, d.h. Gesamtleistung in MLup/s geteilt durch Anzahl der MPI-Prozesse für die Berechnung. Bei diesem Vergleich ist zu beachten, dass für die hybride Parallelisierung doppelt so viele Kerne aufzuwenden waren wie für die reine MPI Parallelisierung, da für jeden MPI-Prozess noch ein zusätzlicher Thread gestartet wird. Wird statt der MPI Prozesse die Gesamtanzahl der Kerne benutzt, fällt der Vergleich entsprechend noch ungünstiger aus.
Überraschend ist jedoch der Vergleich der beiden Parallelisierungsstrategien bei Nutzung einer Intel Xeon Workstation. Hier hat die hybride Parallelisierung bei Vergleich der MPI Prozesse einen Geschwindigkeitsvorteil. Bei einem Vergleich über die Gesamtanzahl der verwendeten CPUs liegt jedoch weiterhin die reine MPI Parallelisierung vorn.
Dies liegt daran, dass durch die zusätzliche OpenMP Parallelisierung nur ein Teil des Simulationsalgorithmus, genauer gesagt der Kollisionsteil, parallelisiert wird. Der restliche Teil jedoch unangetastet bleibt. Bei Erhöhung der MPI Prozesse wird jedoch das Rechengebiet jedes Prozesses kleiner.
Da für die hybride Parallelisierung mit MPI und OpenMP auch zusätzliche Prozessorkerne für die OpenMP Threads benötigt werden, scheint eine hybride Parallelisierung in diesem Fall nicht sinnvoll zu sein. Eine reine Parallelisierung mittles MPI unter Verwendung der gleichen Gesamtanzahl an Prozessorkernen, wie sie für einen hybriden Ansatz benötigt würde, führt zu einer deutlich höheren Geschwindigkeit der Simulation.
Ein hybrider Parallelisierungsansatz könnte jedoch in Betracht kommen, wenn aufgrund der Limitierung durch die Scheibendiskretisierung keine weitere Erhöhung der MPI-Prozesse mehr möglich ist. In diesem Fall lässt sich durch Nutzung der zusätzlichen OpenMP Threads die Anzahl der verwendbaren Prozesse weiter steigern. Wie jedoch der Vergleich der Ergebnisse auf der HLRB II Architektur mit den Ergebnissen der Xeon Workstation für den hybriden Parallelisierungsansatz gezeigt hat, kann es zu plattformspezifischen Problemen kommen, welche statt zu einer Geschwindigkeitssteigerung zum Gegenteil führen.
Bei dem verwendeten Benchmarkbeispiel ist die maximal mögliche Anzahl an MPI Prozessen

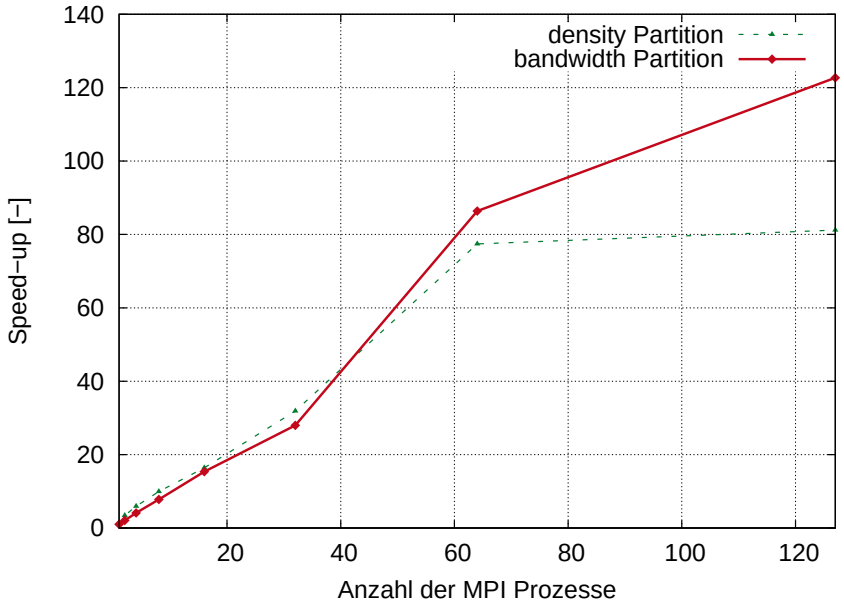


Abbildung 7.9: Gegenüberstellung der Leistung auf *density* und *bandwidth* Partitionen. Hybride Parallelisierung. Hier ist allerdings darauf hinzuweisen, dass im Diagramm die parallele Effizienz auf einer *bandwidth* Partition auf einen Wert über 1 steigt, was laut Definition gar nicht möglich ist. Dies liegt daran, dass die Problemgröße pro Rechenprozess abnimmt und ab einer gewissen Größe vollständig in den CPU Cache passt. Dies führt zu einem Leistungsschub, welcher bei der Rechnung auf nur einer CPU, welche als Referenzgröße für den *speed-up* dient, nicht vorhanden ist. Bei weiter zunehmender Anzahl der Prozesse nimmt die Effizienz wieder ab.

bei 127 erreicht, d.h. das Rechengebiet ist in x-Richtung in 127 Scheiben unterteilt. Die Scheiben haben aber in y- und z-Richtung ihre volle, ungeteilte Größe. In nachfolgender Tabelle 7.8 ist die Geschwindigkeit der Simulation für eine reine MPI Parallelisierung und eine hybride Parallelisierung mit zwei und vier Threads pro MPI-Prozess dargestellt; jeweils für eine *bandwidth* und *density* Partition (in MLup/s).

	Leistung in MLup/s für				
Num. Proc.	MPI *band.*	2 OMP *dens.*	2 OMP *band.*	4 OMP *dens.*	4 OMP *band.*
16	19,74	7,13	6,31	2,20	2,35
127	85,98	35,29	50,47	-	-

Tabelle 7.8: Leistung bei Anwendung einer reinen MPI Parallelisierung (*bandwidth* Partition) und einer hybriden Parallelisierung mit 2 und 4 OpenMP (OMP) Prozessen je MPI Prozess (jeweils auf *density* und *bandwidth* Partition).

7.6.2 Auswertung der Unterschiede density/bandwidth Partitionen

Bei der Auswertung der Unterschiede der Performance auf *density* und *bandwidth* Partitionen zeigt sich, dass die Nutzung der *bandwidth* Partitionen vorteilhaft ist (bzw. die Nutzung nur

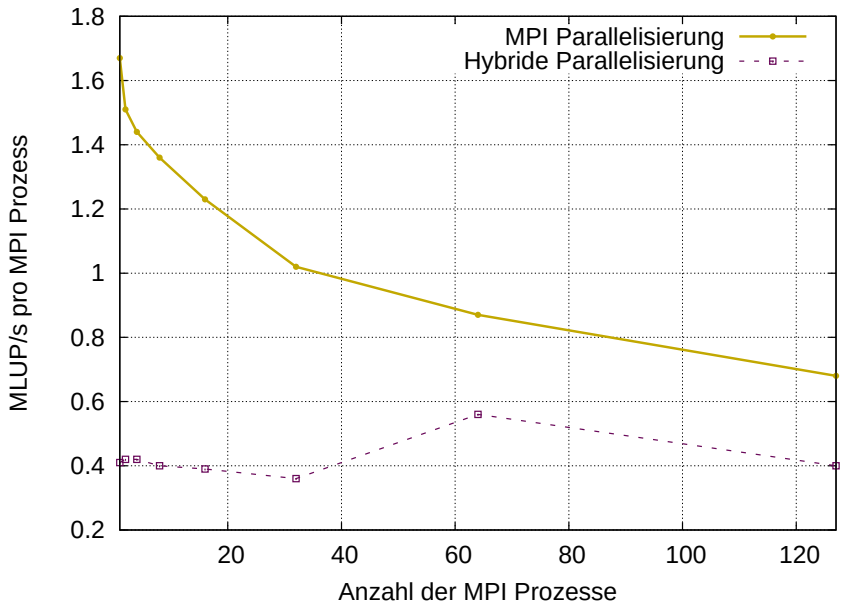


Abbildung 7.10: Vergleich von MPI und hybrider MPI/OpenMP Parallelisierung auf einer *bandwidth* Partition.

der Hälfte der Prozessorkerne auf *density* Blades). Wie in Tabelle 7.9 und in den Diagrammen 7.8 und 7.9 dargestellt, nimmt der Leistungsvorteil der *bandwidth* Blades mit zunehmender Prozessanzahl, d.h. mit einer kleiner werdenden Problemgröße je Prozessorkern, zu (SGI Altix MPI; in MLup/s).

	Leistung in MLup/s bei Nutzung von			
Num. Proc.	MPI *dens.*	MPI *band.*	hybrid *dens.*	hybrid *band.*
1	1,73	1,67	0,43	0,41
2	3,04	3,03	1,48	0,84
4	5,77	5,77	2,57	1,68
8	9,06	10,86	4,32	3,20
16	16,86	19,74	7,13	6,31
...				
127	78,47	85,98	35,29	50,47

Tabelle 7.9: Leistung des Simulationskerns bei MPI und hybrider Parallelisierung jeweils auf einer *density* und *bandwidth* Partition gegenüber gestellt.

Dies lässt sich dadurch erklären, dass bei kleiner werdender Rechendomäne pro MPI-Prozess bedingt durch die Gebietszerlegung der Anteil der Kommunikation pro Zeitschritt proportional ansteigt. Somit wirkt sich der Leistungsvorteil durch die höhere verfügbare Bandbreite stärker aus.
Bei Betrachtung der ersten Benchmark Rechnungen mit ein bis acht Prozessorkernen wird deutlich, dass der Simulationskernel bei Verwendung von bis zu einschließlich vier Prozessen auf beiden Partitionstypen noch identisch skaliert. Dies zeigt, dass die halbierte Speicherbandbreite pro Prozessorkern keine Auswirkung auf die Skalierungseigenschaften hat. Vielmehr wird deutlich, dass die halbierte Bandbreite pro Kern für die Kommunikation über NUMA-

Link verantwortlich für die unterschiedliche Leistungsfähigkeit auf den beiden Partitionstypen ist.
Wenn immer möglich sollte somit, v.a. bei Nutzung von vielen CPUs auf dem HLRB II, eine *bandwidth* Partition genutzt werden. Da für das interaktive Arbeiten normalerweise jedoch nur *density* Blades auf *density* Partitionen zur Verfügung stehen, bleibt hier nur der Umweg über die manuelle Begrenzung der Anzahl der MPI-Prozesse pro Blade über entsprechendes Setzen von Umgebungsvariablen für die MPI Umgebung. Dieses Vorgehen hat jedoch den Nachteil, dass nur noch die halbe Anzahl an CPU-Kernen einer interaktiven Shell für Simulationszwecke zur Verfügung stehen.

7.6.3 Ergebnisse der Betrachtung des *scale-ups*

Ergänzend zu den Betrachtungen des *strong scalings* des Codes wurden Untersuchungen zum *weak scaling*, d.h. bei fester Problemgröße pro Rechenprozess, unternommen.
Diese Untersuchungen lassen Rückschlüsse hinsichtlich des Einflusses der Kommunikation zwischen den Rechenprozessen auf die Gesamtleistung des Systems zu.
Die Untersuchungen wurden, wie bereits in Abschnitt 7.5.1 erläutert, mit zwei unterschiedlichen Größen der Rechengebiete durchgeführt. Zusätzlich wurden die Benchmarkrechnungen sowohl auf einer *density* als auch auf einer *bandwidth* Partition ausgeführt. Hierbei wird untersucht, wie sich die unterschiedliche verfügbare Bandbreite beim Interconnect zwischen den *blades* des HLRB II auf die Gesamtleistung auswirkt; v.a. wenn die Daten komplett im Cache des Prozessors abgelegt werden können, ohne dass langsamere Zugriffe auf den Arbeitsspeicher nötig sind und somit der Rechenkernel seine maximal mögliche Geschwindigkeit erreichen kann.
In Abbildung 7.11 ist der *scale-up* für die Benchmarkrechnungen mit dem kleinen Problem auf einer *density* Partition des HLRB II dargestellt.

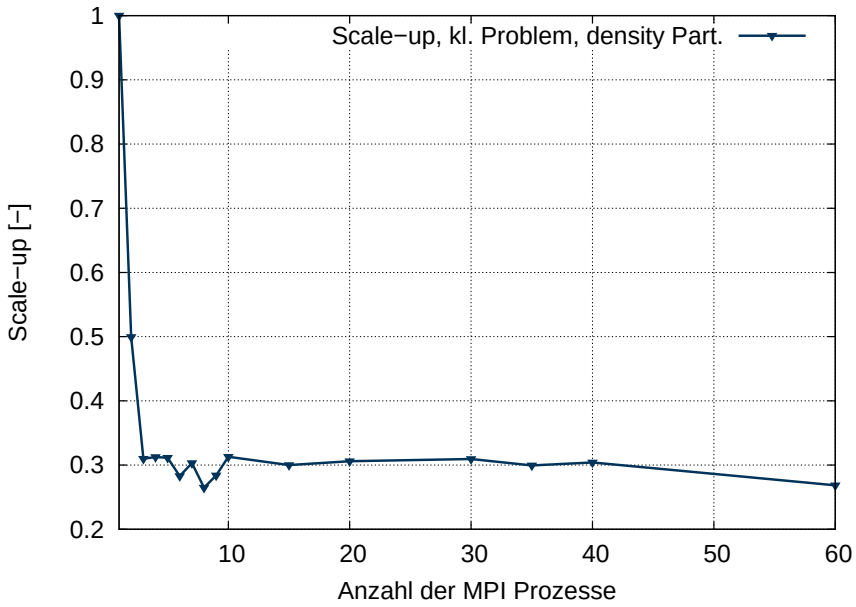


Abbildung 7.11: *scale-up* auf der *density* Partition mit einer Problemgröße pro Prozess kleiner als der CPU Cache.

Bei der Betrachtung der Geschwindigkeit fällt auf, dass die Leistung eines einzelnen Rechenprozesses bei nur einem Rechenprozess am höchsten ist, da hier keine Kommunikation zwischen einzelnen Rechenknoten stattfindet. Bei Nutzung von zwei Rechenprozessen, und somit dem Vorhandensein von Kommunikation zwischen Rechenprozessen, bricht die Geschwindigkeit des einzelnen Rechenprozesses bereits um ca. 50% ein. Bei Hinzunahme eines weiteren Prozesses, d.h. mindestens ein Prozess hat nun zwei Kommunikationspartner in der eingesetzten Scheibenparallelisierung, bricht die Geschwindigkeit der einzelnen Prozesse weiter ein; auf ca. 31% der Geschwindigkeit eines einzelnen Prozesses ohne Kommunikation. Prinzipbedingt ist die Leistungsfähigkeit des Gesamtsystems abhängig vom langsamsten Prozess, da alle anderen Prozesse einmal je Zeitschritt aufeinander warten, um Daten am Interface auszutauschen. Der weitere Abfall der Geschwindigkeit bei einer höheren Anzahl an Prozessen (ab ca. 50 MPI-Prozessen) lässt sich mit der steigenden Belastung des Master-Knotens erklären. Die Belastung resultiert zum Einen aus dem gestiegenen Kommunikationsaufwand, da mit mehr Slave-Knoten Daten ausgetauscht werden müssen, und zum Anderen aus der ansteigenden Gesamtdatenmenge, die durch den Master-Knoten bewältigt werden muss.
Man kann im weiteren Verlauf gut erkennen, dass die Leistung eines einzelnen Rechenprozesses konstant bleibt (kleinere Abweichungen sind zu vernachlässigen, da sie beispielsweise durch die Auslastung der jeweiligen Partition des HLRB II erklärt werden können).
Für ausgewählte Anzahlen an Prozessoren sind die gemessenen Geschwindigkeiten für das in den CPU-Cache passende Problem auf *density* Partitionen in Tabelle 7.10 gegeben.

Anzahl der MPI Rechenprozesse	MLup/s total	*scale-up*
1	5,38	1
2	5,37	0,50
3	5,00	0,31
20	32,95	0,31
40	65,47	0,30

Tabelle 7.10: Geschwindigkeit des in den CPU-Cache passenden Problems auf einer *density* Partition.

Der *scale-up* für die Simulationen mit kleiner Problemgröße auf einer *bandwidth* Partition ist im Diagramm 7.12 dargestellt.
Im Vergleich mit den Ergebnissen auf *density* Partitionen fällt auf, dass sich bei Problemen mit einer kleinen Anzahl an Freiheitsgraden pro Prozessorkern die erzielten Geschwindigkeiten kaum unterscheiden. Dies ist darauf zurückzuführen, dass durch die sehr kleine Problemgröße je CPU auch die zu übertragende Datenmenge zwischen zwei Rechenprozessen sehr gering ist und somit die unterschiedliche verfügbare Bandbreite eine untergeordnete Rolle spielt.
Für die Kommunikation zwischen benachbarten Rechenprozessen bieten *density* Partitionen teilweise sogar leichte Vorteile, da dort Kommunikation von vier Rechenprozessen innerhalb eines Blades abgewickelt werden kann ohne NUMALink-Verbindungen nutzen zu müssen. Die Rechenprozesse tauschen während eines Rechenzeitschritts nur Daten mit den direkt benachbarten zwei Prozessen (am Rand nur mit einem Prozess) aus. Nur wenn ein Update an die Visualisierungs- bzw. in diesem Fall an die Ersatz-Visualisierungsapplikation erfolgt, kommunizieren alle Rechenprozesse mit dem Masterprozess. Deshalb zeigt die halbierte NUMALink-Bandbreite pro Prozessorkern bei *density* Blades für dieses Benchmarkbeispiel keine negative Auswirkung.

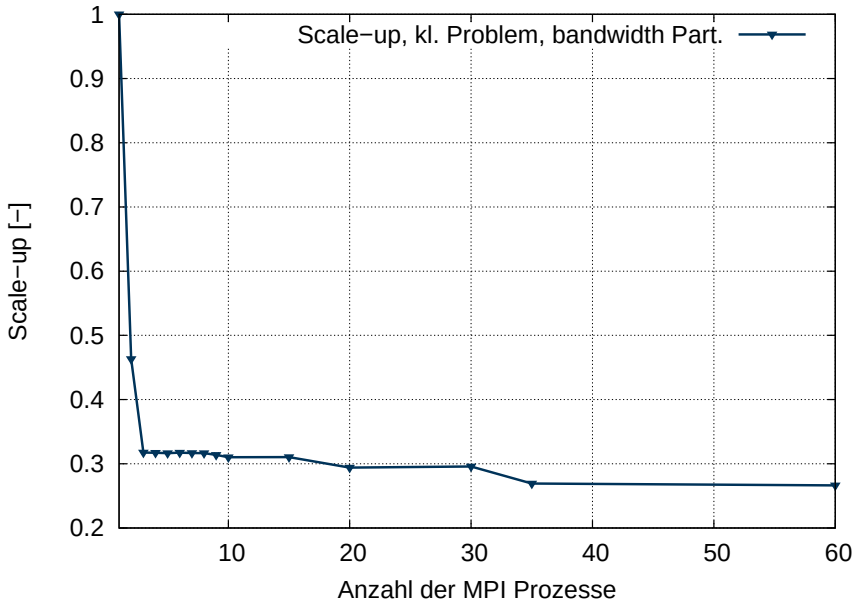


Abbildung 7.12: *scale-up* auf der *bandwidth* Partition mit einer Problemgröße pro Prozess kleiner als der CPU Cache.

Es lässt sich jedoch beobachten, dass die Proportionen der Rechendomäne einen nicht zu vernachlässigenden Einfluss auf die Geschwindigkeit der Simulation haben. Bei zunehmender Ausdehnung in y- und z-Richtung ergibt sich eine größere Fläche, über die Daten ausgetauscht werden müssen. Somit steigt der Zeitanteil, welcher auf reine Kommunikation entfällt an und die Gesamtperformance des Simulationscodes nimmt ab. Dies tritt sowohl bei einer großen als auch bei einer kleinen Zahl an Freiheitsgraden pro CPU auf.
Zur Verdeutlichung dieser Eigenart wurde für wenige MPI-Prozesse versuchsweise die Anzahl der zu übertragenden Freiheitsgrade durch Änderung der Geometrie halbiert. Die Gesamtanzahl an Freiheitsgraden pro Rechenknoten wurde aber konstant gehalten. In Abbildung 7.13 und 7.14 sind die Ergebnisse aufgetragen. Bei dem kleinen Problem mit größerem Kommunikationsaufwand fällt für zwei und drei Prozesse der *speed-up* sogar unter eins. Ebenfalls in diesem Diagramm zu erkennen ist der Einfluss der Art der Partition auf den *speed-up*. Bei dem Modell mit doppelt so viel Kommunikation tritt der Unterschied zwischen *density* und *bandwidth* Partition mit zunehmender Prozessanzahl deutlich sichtbar zu Tage, wohingegen bei dem Modell mit weniger Kommunikation dieser Unterschied nicht zu erkennen ist.
Bei einer großen Anzahl an Freiheitsgraden pro CPU und bei für die Parallelisierung ungünstigen Proportionen der Rechendomäne, d.h. einer großen Fläche über die kommuniziert werden muss, ist jedoch ein negativer Einfluss der *density* Partitionen bzw. der halbierten NUMALink-Bandbreite pro CPU-Kern deutlich festzustellen; siehe hierzu Diagramm 7.14.
Bei den Untersuchungen des *scale-up* für eine hohe Anzahl an Freiheitsgraden pro Rechenprozess wurde weiterhin der Einfluss der Speicherbandbreite auf die Geschwindigkeit deutlich. In folgender Tabelle 7.11 ist für ausgewählte Parallelisierungsstufen die Geschwindigkeit der Simulation (in MLup/s) für kleine und große Rechengebiete gegenüber gestellt.
Es ist deutlich sichtbar, dass die Leistung durch die Zugriffe auf den Hauptspeicher massiv sinkt. Der *scale-up* für große Probleme nimmt jedoch zu. Dies ist in folgender Tabelle 7.12 zusammengefasst, in welcher der *scale-up* für ausgewählte Parallelisierungsstufen, sowohl für

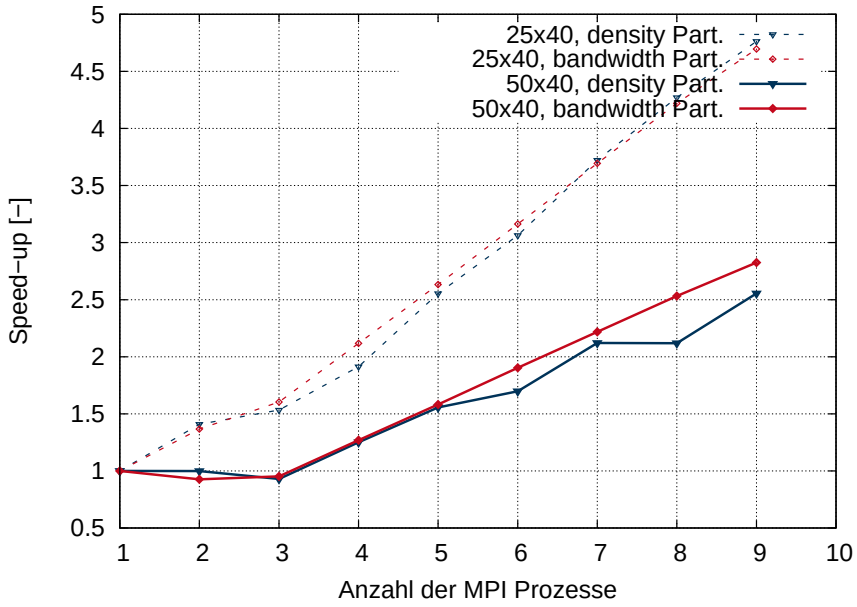


Abbildung 7.13: *Speed-up* des kleinen Modells für unterschiedliche Proportionen des Rechengebiets bei gleicher Anzahl der Freiheitsgrade.

MPI Proc.	kl. Problem, density [*MLup/s*]	kl. Problem, bandwidth [*MLup/s*]	gr. Problem, density [*MLup/s*]	gr. Problem, bandwidth [*MLup/s*]
1	5,383	5,385	1,756	1,704
2	5,378	4,987	3,152	3,403
3	5,004	5,129	4,670	4,676
20	32,946	31,664	10,547	13,204
60	86,4716	86,038	3,358	86,358

Tabelle 7.11: Geschwindigkeit der Simulation (in MLup/s) für kleine und große Rechengebiete.

ein kleines als auch für ein großes Problem, angegeben ist.
Dies erklärt sich dadurch, dass das Verhältnis aus Kommunikation und Rechenarbeit (CCR) für ein großes Problem günstiger ausfällt. In diesem Fall wurde die Anzahl der zu übertragenden Voxel konstant gehalten.

7.6.4 Zusammenfassung der Ergebnisse

Im obigen Abschnitt wurde die Leistung des Simulationskernels der CSE auf dem Höchstleistungsrechner HLRB II untersucht. Hierbei wurde auch auf die Besonderheiten der Architektur der Maschine eingegangen.
Es hat sich gezeigt, dass ein hybrider Parallelisierungsansatz für die CSE `iFluids` unpraktikabel ist. Zwar ist die Kollisionsroutine des Algorithmus theoretisch für eine Parallelisierung mittels OpenMP geeignet, da die Berechnungen nur lokal an den Knoten stattfinden und somit jeder Schleifendurchlauf unabhängig ist, aber die anderen Teile des Simulationsalgorithmus, wie beispielsweise die Lösung der Temperaturgleichung oder die Propagation, eignen sich we-

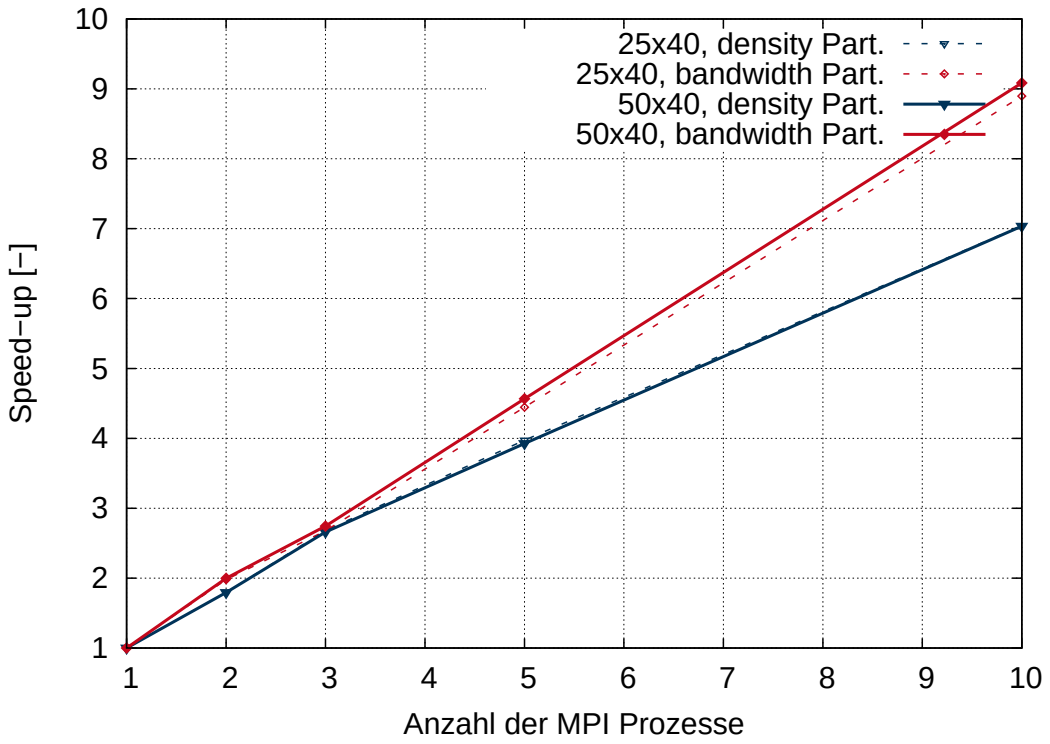


Abbildung 7.14: Speed-up des großen Modells bei Nutzung von *density* und *bandwidth* Partitionen.

MPI Proc.	kl. Problem, density	kl. Problem, bandwidth	gr. Problem, denisty	gr. Problem, bandwidth
2	0,499	0,463	0,897	0,999
3	0,310	0,318	0,886	0,915
5	0,311	0,317	0,785	0,913
10	0,313	0,310	0,704	0,909
60	0,268	0,266	0,032	0,845

Tabelle 7.12: scale-up für ausgewählte Parallelisierungsstufen, sowohl für ein kleines als auch für ein großes Problem.

niger für eine OpenMP-Parallelisierung. Somit profitiert nur ein Teil des Algorithmus von dem zusätzlichen OpenMP-Prozess pro MPI-Prozess. Eine bessere Performance ergibt sich, wenn die für zusätzliche OpenMP-Prozesse benötigten CPU-Kerne für eine stärkere MPI-Parallelisierung verwendet werden, da dadurch die Anzahl an Voxeln pro CPU-Kern sinkt.
Ebenso wurde die maximal mögliche Performanceoptimierung für den Simulationskernel untersucht. Hierbei wurde die Simulation komplett von der Visualisierung und der damit zusammen hängenden Kommunikation getrennt. Des Weiteren wurde für diese Messungen eine komplett leere Fluiddomäne betrachtet. Die Simulation innerhalb der CSE erreicht jedoch die maximal mögliche Geschwindigkeit des einzeln betrachteten Simulationskernels nicht.
Dies liegt zum Einen an der Einbindung in die Kommunikations- und Steuerstrukturen der CSE, auch wenn die Visualisierungsapplikation durch eine Dummy-Applikation ohne grafische Benutzeroberfläche ersetzt worden ist, und zum Anderen an den aus Stabilitätsgründen nicht vollständig ausgeschöpften Optimierungsmöglichkeiten. Es hat sich gezeigt, dass manche Op-

timierungsfunktionen des Intel Compilers, welche tief in die Codestruktur eingreifen und diese umstellen, zu einem instabilen Verhalten anderer Teile der Computational Steering Umgebung geführt haben. Die entkoppelte Simulation war hiervon jedoch nicht betroffen, weshalb hier eine stärkere Codeoptimierung durch den Compiler durchgeführt werden konnte, was in einer höheren Leistung resultierte.
Ein Vergleich mit der Leistungsfähigkeit der Simulation auf aktueller marktüblicher Hardware hat gezeigt, dass eine Nutzung von großen Rechnersystemen wie dem HLRB II nur bei sehr großen Problemen mit einer hohen Parallelisierung sinnvoll ist. Genau hier liegt jedoch aktuell noch ein Problem der Parallelisierungsstruktur des Rechenkerns der CSE. Aufgrund der Gebietszerlegung in Scheiben ist die maximal mögliche Anzahl an nutzbaren Prozessen sehr schnell erreicht. Gerade hinsichtlich des zunehmenden Trends zu massiv parallelen Systemen, sollte deshalb eine Umstellung der Gebietszerlegungsstrategie erfolgen, wie von van Treeck in [66] vorgestellt.
Die Nutzung einer zusätzlichen OpenMP Parallelisierung stellt zwar prinzipiell eine Möglichkeit dar, um trotz der Beschränkungen durch das Gebietszerlegungsschema noch mehr Prozessoren nutzen zu können, jedoch scheiterte dieser Ansatz zumindest auf dem HLRB II aufgrund der Performanceprobleme des hybriden Codes auf der Altix Architektur.
Darüber hinaus ist zu beachten, dass im konkreten Anwendungsfall bei kleiner werdenden Problemen pro MPI-Prozess damit zu rechnen ist, dass ein möglicher Geschwindigkeitsgewinn durch den Overhead, welcher durch das Starten und Beenden der zusätzlichen OpenMP Thread entsteht, wieder zunichte gemacht wird.
Für kleinere oder mittelgroße Probleme sind aufgrund der höheren Spitzenleistung aktueller Prozessoren wie der Intel Xeon Baureihe, verglichen mit den im HLRB II verbauten Intel Itanium II CPUs, bessere Leistungswerte auf leistungsfähigen Mehr-Prozessor-Workstations oder kleinen Workstationclustern zu erwarten.
Aufgrund der relativ geringen maximal möglichen Anzahl an MPI-Prozessen für das vorgestellte, praxisnahe Benchmark-Problem lässt sich keine eindeutige Aussage zur Skalierbarkeit bei einer sehr großen Anzahl an Prozessen machen, da das große Benchmarkproblem nicht soweit unterteilt werden kann, dass es auf einem Prozess komplett in den Cache des Prozessors passt. Bei der hier untersuchten Prozessanzahl ist allerdings gut zu erkennen, dass mit zunehmender Prozessanzahl der Kommunikationsoverhead stark ansteigt und somit die Leistung pro Prozess stetig abnimmt.
Die Untersuchung der Leistung anhand eines synthetischen Benchmarkproblemes hat den möglichen Leistungszuwachs bei Erreichen einer Parallelisierung, bei der das Problem auf den einzelnen Rechenknoten vollständig in den Cache des Prozessors passt, gezeigt.
Zusätzlich ließ sich durch diese Untersuchungen der Einfluss der Geometrie des Strömungsgebiets auf die Leistungsfähigkeit der Parallelisierung quantifizieren.
Es konnte auch gezeigt werden, dass die Wahl einer Partition des HLRB II, d.h. *density* oder *bandwidth*, zwar durchaus unterschiedliche Leistungswerte liefert, eine generelle Aussage welcher Teil der Maschine zu bevorzugen ist jedoch nicht allgemeingültig getroffen werden kann, da dies stark von der Anzahl der verwendeten Prozessoren und der Größe und Geometrie des Problems abhängt.

Kapitel 8

Referenzsimulationen

In diesem Kapitel werden zwei Referenzsimulationen vorgestellt, mit deren Hilfe die interaktive Simulationsumgebung `iFluids` evaluiert worden ist. Bei dem ersten Modell handelt es sich um einen beheizten horizontalen Zylinder und bei dem zweiten Modell um die Modellierung des Testraums aus dem IGSSE Projekt *2-08 Building, users, climate*. Bei beiden Modellen wurde das Problem zusätzlich zu Vergleichszwecken in einem validierten, kommerziellen CFD Programm modelliert und berechnet.

8.1 Benchmarkproblem: Horizontaler Zylinder

8.1.1 Problembeschreibung

Für die Evaluierung der Qualität der Simulationsergebnisse aus `iFluids` im Vergleich zu einer Referenzlösung wurde ein Benchmark Problem definiert. Hierbei handelt es sich um einen geheizten Zylinder in einem abgeschlossenen Raum (siehe Abbildung 8.1). Für die Bestimmung einer Referenzlösung wurde das definierte Problem in einem kommerziellen, validierten CFD Simulationsprogramm modelliert und simuliert. Als Programm wurde Ansys CFX gewählt.
Der modellierte Raum ist rechteckig und hat die Dimensionen:

x-Richtung	2,20 m
y-Richtung	1,70 m
z-Richtung	1,00 m

Tabelle 8.1: Dimensionen des modellierten Raums.

Der darin befindliche Zylinder hat einen Durchmesser von $0,2m$ und eine Länge von $1,0m$. Er ist in z-Richtung orientiert und reicht von $z = 0m$ bis $z = 1,0m$, d.h. von Wand zu Wand. Die Achse liegt auf $x = 1,1m$ und $y = 0,4m$. Für die Simulation in `iFluids` liegt die Geometrie im STL-Dateiformat vor und besteht aus 134 Facetten. Die Randbedingung wurde direkt in der Geometriedatei definiert. Für die Referenzsimulation wurde der Zylinder direkt im Preprocessor von Ansys CFX definiert.
Der Oberfläche des Zylinders wurde eine Temperatur von $303K$ und den Oberflächen des umgebenden Raumes eine Temperatur von $293K$ zugewiesen. Das den Zylinder umgebende Fluid wurde als Luft mit einer Initialtemperatur von $293K$ und einer Dichte entsprechend

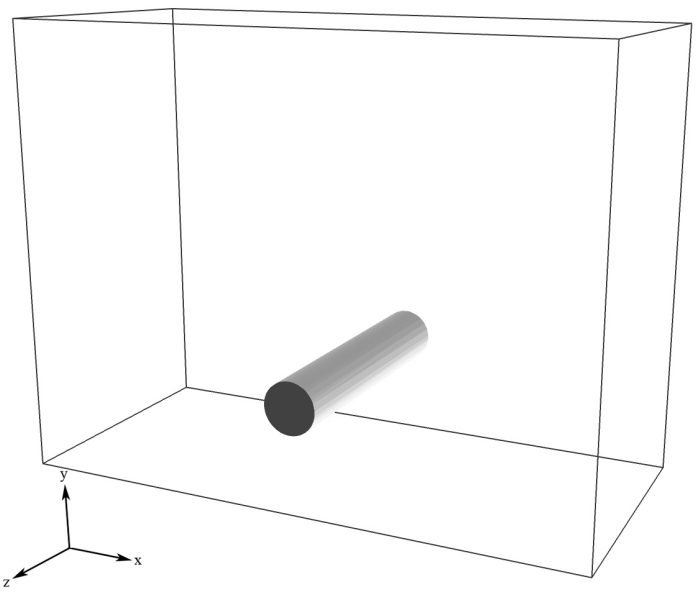


Abbildung 8.1: Geometrie des Benchmark Problems zur Bestimmung der Qualität der Ergebnisse aus `iFluids`.

einem Druck von $1bar$ definiert. Die Wände haben eine slip-Randbedingung. Aufgrund der Wahl der Strömungsparamter ergibt sich eine Rayleigh-Zahl (siehe Gleichung 3.31) von ca. $Ra = 1,7 \cdot 10^7$. Dies bedeutet, dass sich die Strömung bereits in einem Übergangsbereich von laminarem zu turbulentem Verhalten befindet. Als charakteristische Länge wurde der Durchmesser des beheizten Zylinders $d_{cyl} = 0,2m$ gewählt
Zur Gegenüberstellung der beiden Simulationen wurden zwei Messpunkte P1 und P2 definiert (siehe Abbildung 8.2). Eine Vergleich des Wärmeübergangs in beiden Simulationsarten und eine Analyse des Konvergenzverhaltenswurde im Rahmen dieser Arbeit nicht durchgeführt. Der Wärmeübergang für die hier verwendete hybride thermische LB-Methode wurde z.B. von van Treeck in [58] detailiert betrachtet.

8.1.2 Simulation in CFX

Da es sich bei Ansys CFX um ein sehr mächtiges, kommerzielles Simulationswerkzeug handelt, steht eine Vielzahl an Möglichkeiten zur Definition und Feinjustierung von zu simulierenden Problemen zur Verfügung.
Für die Berandung wurde der CFX-Randbedingungstyp *WALL* mit den Eigenschaften *No Slip* und der Wandrauhigkeit *Smooth Wall* gewählt. Für den Übergang zwischen heißem Zylinder und Fluid wurde ebenfalls *No Slip* und *Smooth Wall* definiert; für den Wärmeübergang wurde *Conservative Interface Flux* ausgewählt.
Es wurden Simulationen mit zwei verschiedenen Turbulenzmodellen für das Fluid durchgeführt. Ein k-epsilon Modell mit skalierbarer Wandfunktion und ein LES-Modell mit einer Smagorinsky Konstante von 0,2.
Für die Simulation mit dem LES Turbulenzmodell wurde eine CFL-Zahl[1] von $\sigma = 0,75$ gewählt.

[1] Auch Courant-Zahl genannt. Nach den Mathematikern R. Courant, K. Friedrichs und H. Lewy benannt.

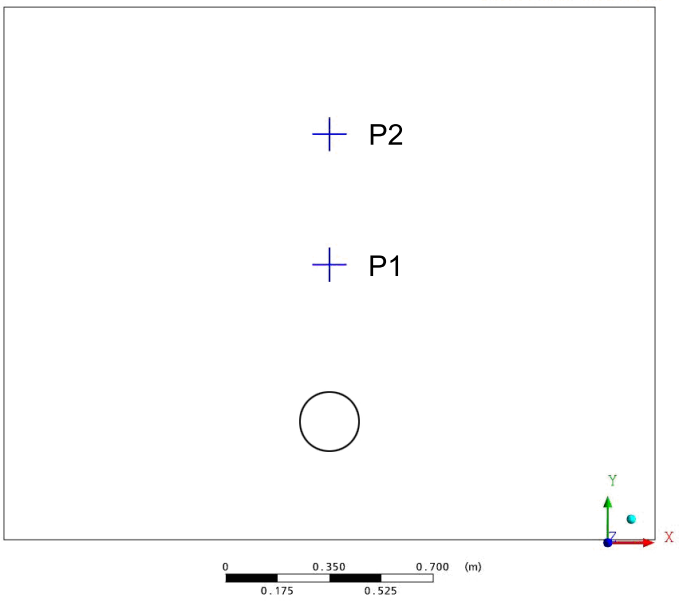


Abbildung 8.2: Messpunkte für die Ausgabe der Simulationsdaten.

Zum Vergleich der beiden Modelle wurde die zeitliche Entwicklung der Geschwindigkeit und der Temperatur an den in Abbildung 8.2 definierten Punkten verglichen. Der Vergleich der Geschwindigkeiten am Punkt 2 ist in Diagramm 8.3 dargestellt.
Da es sich bei den Ergebnissen, die mit dem k-epsilon Modell berechnet wurden, modellbedingt um zeitlich gemittelte Werte handelt, wurde für die Ergebnisdaten des LES Modells eine polynomiale Regressionsanalyse vorgenommen. Der Verlauf der durch diese Analyse gewonnenen Trendlinie des Geschwindigkeitsverlaufs ist zusätzlich in Diagramm 8.3 aufgetragen. Hierbei zeigt sich im späteren Verlauf eine gute Übereinstimmung der beiden Modelle. Generell ist jedoch zu erkennen, dass beim LES Modell ein ausgeprägterer Einschwingvorgang zu beobachten ist und es beim RANS Modell deutlich länger dauert, bis sich aufgrund der Temperaturdifferenz zwischen Zylinder und umgebendem Fluid eine Strömung und eine erhöhte Temperatur an Punkt 2 einstellt, obwohl die Initialbedingungen der beiden Modelle gleich gewählt wurden. Dies lässt sich gut erkennen, wenn man den absolute Unterschied der Geschwindigkeiten am Punkt 2 zwischen beiden Modellen, wie in Diagramm 8.4 dargestellt, betrachtet.
Ergänzend zur Betrachtung der Differenz der Strömungsgeschwindigkeit am Punkt 2 wurde auch der zeitliche Verlauf der Temperaturerhöhung infolge des heißen Zylinders betrachtet. Auch hier wurde mit Hilfe einer Regressionsanalyse der Verlauf einer Trendlinie für die Temperaturentwicklung der LES Simulation bestimmt. Die drei Kurven sind in Diagramm 8.5 aufgetragen.
Bei der Betrachtung der Temperaturdifferenz zeigt sich ebenfalls ein ähnlicher Verlauf zwischen dem k-epsilon und dem LES Modell. Hier liegen die Werte des LES Modells (unter Betrachtung der Trendlinie) im eingeschwungenen Zustand ca. 0,1–0,25 K über den Werten des RANS Modells.
Bei der Verwendung des LES Turbulenzmodells werden die in der Strömung auftretenden Turbulenzen deutlich feiner aufgelöst als bei Verwendung des k-epsilon Modells. Dies spiegelt sich jedoch auch in der benötigten Rechenzeit wider. Diese beträgt für das k-epsilon Modell ca. 2 Tage auf einem Rechner mit Intel Core2Duo E4500 Prozessor, wohingegen die Simula-

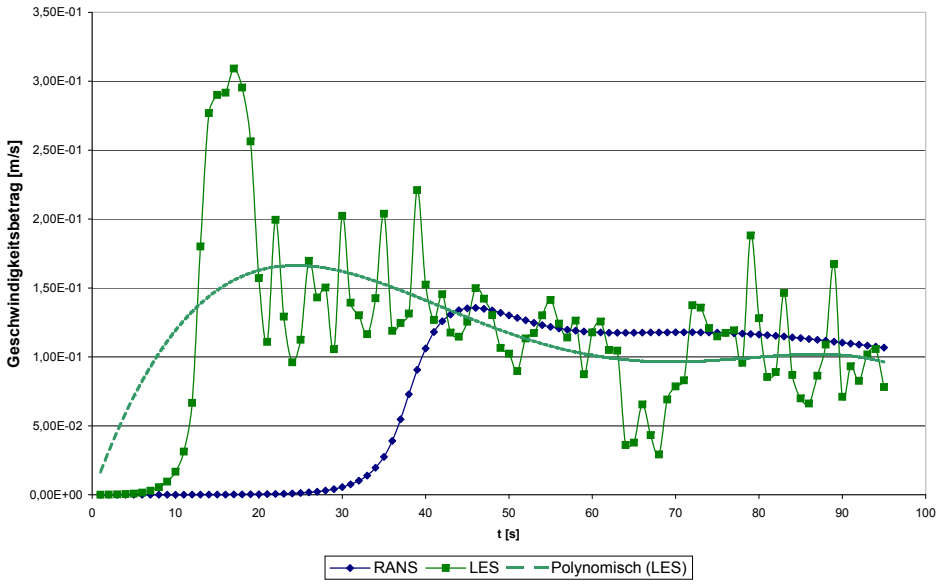


Abbildung 8.3: Zeitlicher Verlauf der Geschwindigkeit an Punkt 2 für die beiden Modelle. Zusätzlich ist ein gemittelter Verlauf der Werte der LES Simulation aufgetragen (Ansys CFX).

tion unter Nutzung des LES Modells ca. die doppelte Zeit benötigt. Die Unterschiede in den gewonnenen Ergebnissen zu einem bestimmten Zeitschritt sind in Abbildung 8.6 ersichtlich. Hier ist die Geschwindigkeitsverteilung zum Zeitpunkt $t = 60s$ im Fluid in einer Schnittebene an $x = 0,5m$ dargestellt. Links die mit dem k-epsilon und rechts die mit dem LES Modell gewonnenen Ergebnisse.

8.1.3 Simulation in `iFluids`

Die Simulationen in `iFluids` wurden auf dem Bundeshöchstleistungsrechner II am LRZ in Garching durchgeführt. Da eine Interaktion mit der Simulation in diesem Fall nicht nötig war, wurde die Visualisierungsapplikation durch eine Dummy-Applikation ersetzt, welche für das Einlesen der Geometrie des Zylinders und für die Ausgabe der Ergebnisdaten auf Datenträger zuständig ist. Die Resultate der Simulation wurden alle n Zeitschritte von der Simulationsapplikation an die Dummy-Visualisierungsapplikation übertragen. An den Messpunkten $P1$ und $P2$ werden an jedem Update-Zeitschritt die Geschwindigkeitskomponenten u, v, w und die Temperatur T aufgezeichnet. Zusätzlich werden an jedem zehnten Update-Zeitschritt die kompletten Ergebnisdaten dieses Zeitschritts gespeichert.
Die Simulationen wurden als batch jobs über das Wartelistensystem des HLRB II gestartet. Als feinste Auflösung wurde eine Gitterweite von $dx = 0,01m$ gewählt. Dies ergibt in x-Richtung 222, in y-Richtung 172 und in z-Richtung 101 Voxel. Da die Parallelisierungsstrategie des Simulationscodes eine Gebietszerlegung in Scheiben vorsieht und eine Mindestdicke der Scheiben von 2 Voxel nötig war, konnte das Problem maximal auf 110 Rechenknoten parallel gerechnet werden.
Neben der feinen Simulation mit einer Gitterweite von $dx = 0,01m$ wurden auch Simulationsläufe mit gröberen Auflösungen durchgeführt. Dies geschah, um Rückschlüsse hinsichtlich der Genauigkeit einer grob aufgelösten interaktiven Simulation machen zu können. Hierbei

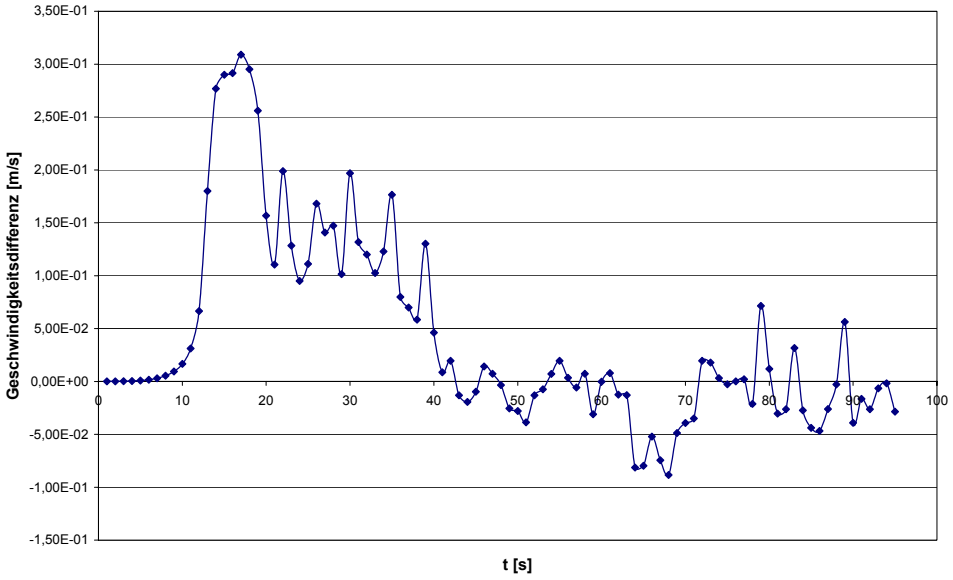


Abbildung 8.4: Absoluter Unterschied zwischen den Ergebnissen des CFX LES Modells und des CFX k-epsilon Modells in [m/s] in Ansys CFX. Hier lässt sich gut der große Zeitversatz zwischen den Modellen erkennen bis sich eine Strömung am Punkt 2 einstellt.

blieben die sonstigen geometrischen Randbedingungen gleich, nur die Gitterauflösung wurde verändert. Wie bei allen Simulationen in `iFluids` kam ein kartesisches Rechengitter mit $dx = dy = dz$ zur Anwendung.
Durch eine Änderung der Gitterweite bei der Lattice-Boltzmann Methode bleiben die dimensionslosen Größen – wie z.B. die Rayleigh-Zahl – gleich, jedoch müssen die LB-Parameter angepasst werden, um weiterhin den Stabilitätskriterien des Verfahrens zu genügen.
In der nachfolgenden Tabelle 8.2 sind die zusätzlich durchgeführten Simulationen mit den dazugehörigen Gitterweiten und der daraus resultierenden Anzahl der Voxel in den betreffenden Modellen aufgelistet:

Gitterweite dx [m]	Anzahl der Voxel in der Simulation
$0,01$	3856584
$0,02$	496944
$0,04$	65208
$0,05$	33075
$0,075$	11160
$0,10$	4554

Tabelle 8.2: Zusätzlich durchgeführte Simulationen mit den dazugehörigen Gitterweiten und der daraus resultierenden Voxelanzahl.

Es wurden maximal 1.200.000 LB-Zeitschritte simuliert. Die Rechenzeit für das feinste Modell betrug 36 Stunden mit 30 Rechenprozessen (32 MPI-Prozesse insgesamt). Die grafische Auswertung der gewonnenen Daten erfolgt mit dem Programm Paraview.
Die in der Simulation gewonnenen Fluiddaten liegen als dimensionslose Größen vor. Diese sind u.a. auch von der gewählten Gitterweite abhängig, weshalb zum Vergleich der gewonnenen Ergebnisse erst eine Rückrechnung in physikalische Einheiten erfolgen muss. Der Umrechenfaktor

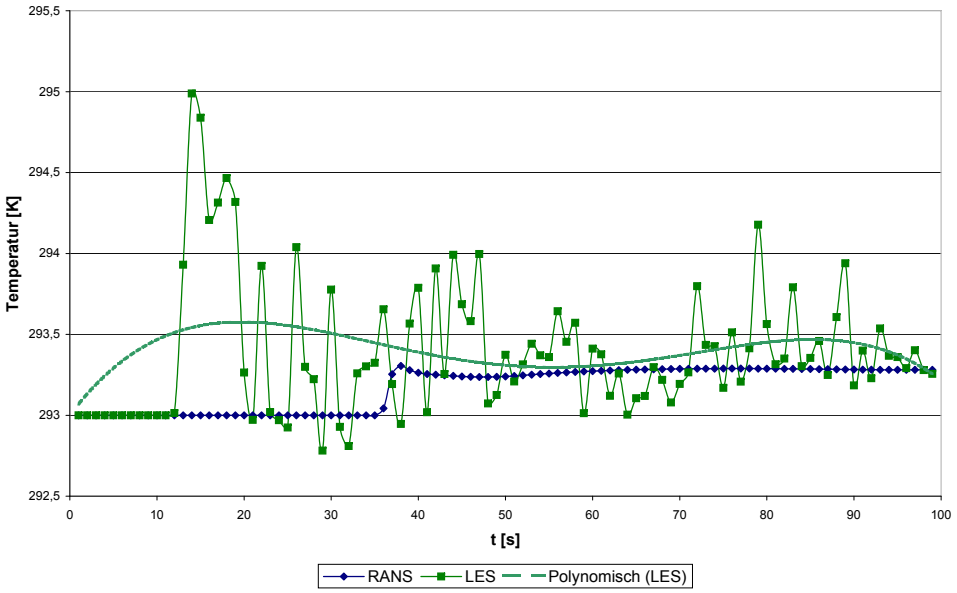


Abbildung 8.5: Zeitlicher Verlauf der Temperaturerhöhung an Punkt $P2$ für die beiden Modelle in Ansys CFX. Zusätzlich ist ein gemittelter Verlauf der Werte der LES Simulation aufgetragen.

für die Geschwindigkeitswerte ist abhängig von den Viskositätswerten im physikalischen und dimensionslosen Raum, sowie von der charakteristischen Länge in physikalischer Einheit und als dimensionslose Größe. Die Umrechnung der Temperaturwerte kann für alle Modelle mit dem selben Umrechnungsfaktor erfolgen, da dieser nicht abhängig von dx ist.
Da es sich bei Lattice-Boltzmann-Zeitschritten um sehr kleine Zeitschritte handelt, wurde nur jeder 1000. Zeitschritt an die Dummy-Visualisierungsapplikation übertragen und auf Datenträger gespeichert. Zusätzlich wurde zur Glättung der Ergebnisse eine polynomiale Regressionsanalyse auf die Daten angewandt. In Abbildung 8.7 ist der zeitliche Verlauf der w-Geschwindigkeitskomponente am virtuellen Messpunkt $P1$ für die beiden Gitterweiten $dx = 0,01m$ und $dy = 0,02m$ dargestellt. Zusätzlich ist auch jeweils der durch gleitenden Durchschnitt gemittelte und dadurch geglättete Verlauf aufgetragen.
In Abbildung 8.8 ist die Temperaturschwankung am virtuellen Messpunkt $P2$ für die beiden feinen Rechengitter über die Zeit dargestellt. Die Temperaturwerte wurden aus den dimensionslosen Größen in die Einheit [K] zurück gerechnet.
Des Weiteren ist zur Veranschaulichung in Abbildung 8.9 die Temperaturverteilung im Raum anhand einer mittig im Raum platzierten x-z-Schnittebene zum Zeitschritt 1.200.000 dargestellt. Links für das feine Modell mit $dx = 0,01m$ und rechts für $dx = 0,10m$. Man erkennt trotz der groben Auflösung und der offensichtlich abweichenden absoluten Werte der Temperatur immer noch das grobe Muster der Strömung.
In den einzelnen Modellen wurde zu jedem Zeitschritt die kinetische Energie E_{kin} im System bestimmt. Sie ist definiert als (siehe auch [58]:

$$E_{kin}(t) := \sum_{k=1}^{N_z-1} \sum_{j=1}^{N_y-1} \sum_{i=1}^{N_x-1} \left[j_x^2(i,j,k)(t) + j_y^2(i,j,k)(t) + j_z^2(i,j,k)(t) \right] \tag{8.1}$$

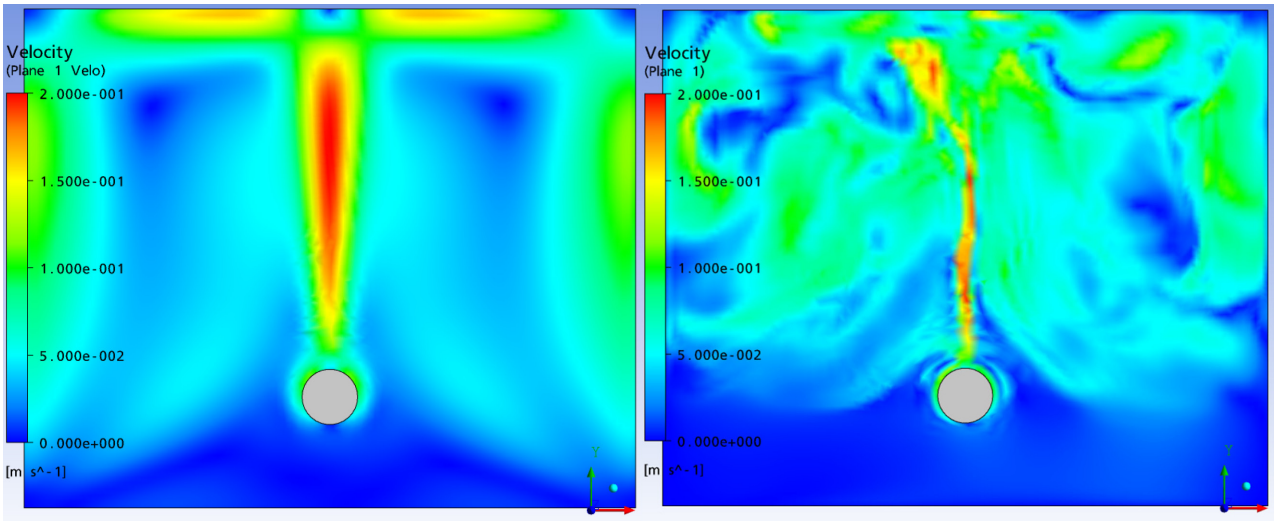


Abbildung 8.6: Gegenüberstellung der gewonnenen Ergebnisse des k-epsilon und LES Modells aus Ansys CFX an einer Schnittebene zur Visualisierung der Strömungsgeschwindigkeit. Hier ist gut der unterschiedliche Charakter der Ergebnisdaten der beiden Verfahren zu erkennen.

Die zeitliche Entwicklung der kinetischen Energie ist für die Simulationen mit den unterschiedlichen Auflösungen im Diagramm (siehe Abbildung 8.10) dargestellt.
Durch den zeitlichen Verlauf der kinetischen Energie lässt sich gut erkennen, ob der anfängliche Einschwingvorgang abgeklungen ist und man mit einer stabilen Lösung des Systems rechnen kann.
Es ist jedoch zu beachten, dass die kinetische Energie in der laufenden Simulation in dimensionslosen Größen errechnet wurde. Da die dimensionslosen Größen jedoch abhängig vom gewählten Knotenabstand dx und somit für jede Simulation unterschiedlich sind, ist nur der qualitative Vergleich zu betrachten. Die absoluten Werte weichen aufgrund der unterschiedlichen Skalierung um Größenordnungen voneinander ab.
Zur Verdeutlichung der Unterschiede in den Ergebnissen der Simulation sind in nachstehender Tabelle die Werte für den gemittelten Geschwindigkeitsbetrag an den virtuellen Messpunkten $P1$ und $P2$ für die verschiedenen Gittergrößen zusammen mit dem relativen Unterschied zur Simulation mit der höchsten Auflösung gegeben. Die Geschwindigkeiten sind in physikalischen Einheiten $[\frac{m}{s}]$ angegeben. Es erfolgte eine zeitliche Mittelung über 60 Sekunden im eingeschwungenen Zustand.

	$P1$		$P2$	
Gitterweite dx [m]	$\lvert\vec{u}\rvert$	Diff. [%]	$\lvert\vec{u}\rvert$	Diff. [%]
$0,01$	$0,1216$	-	$0,1011$	-
$0,03$	$0,0889$	$-26,90$	$0,0810$	$-19,87$
$0,04$	$0,0798$	$-34,44$	$0,0798$	$-21,05$
$0,05$	$0,0805$	$-33,83$	$0,0867$	$-14,19$
$0,10$	$0,0594$	$-51,21$	$0,0969$	$-4,10$

Tabelle 8.3: Geschwindigkeitsunterschiede für verschiedene Gitterweiten an den Messpunkten $P1$ und $P2$ in $[\frac{m}{s}]$.

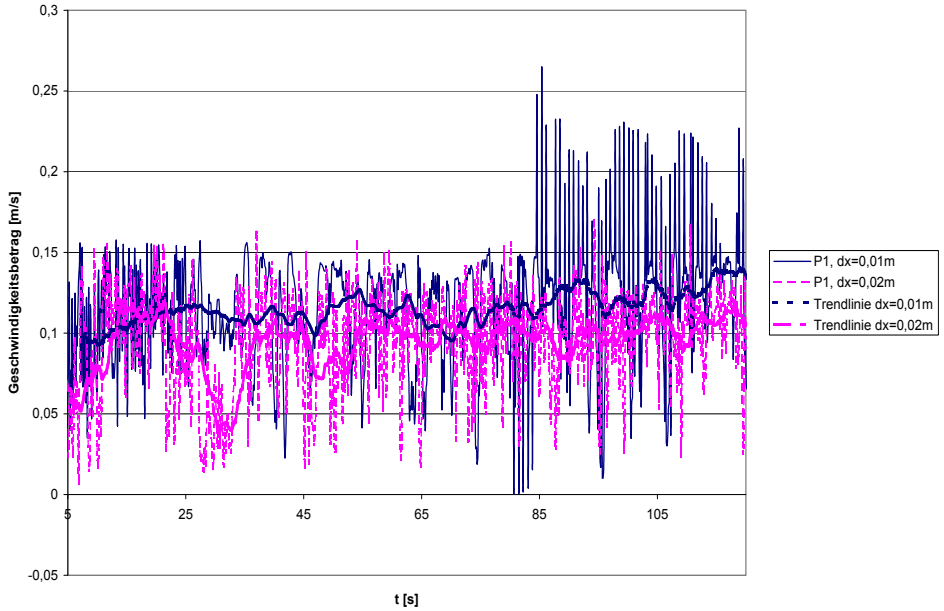


Abbildung 8.7: Zeitlicher Verlauf der Geschwindigkeit am virtuellen Messpunkt $P1$ für $dx = 0,01m$ und $dx = 0,02m$. Zusätzlich ist ein geglätteter Verlauf für die Kurven aufgetragen.

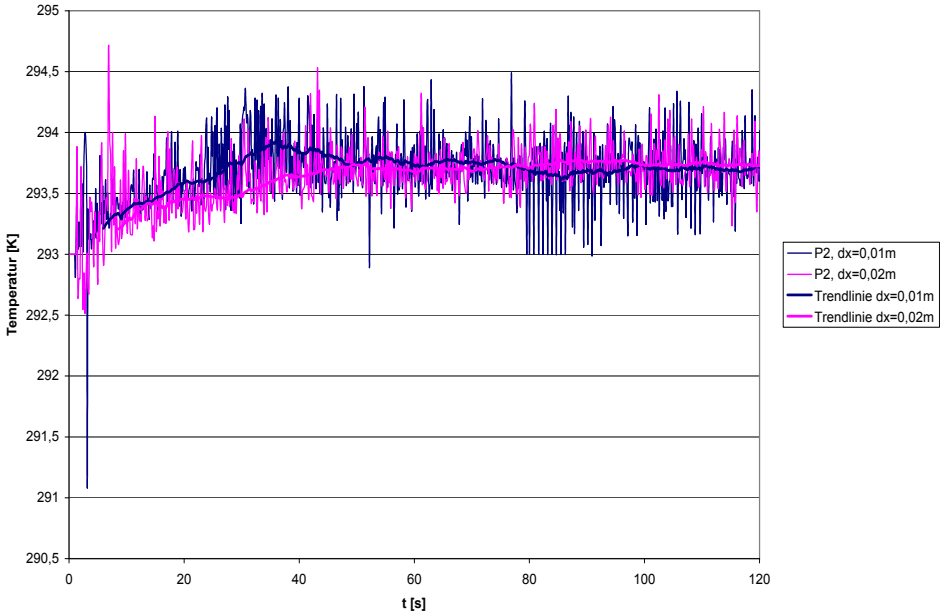


Abbildung 8.8: Zeitlicher Temperaturverlauf am virtuellen Messpunkten $P2$ für $dx = 0,01m$ und $dx = 0,02m$. Zusätzlich ist ein gemittelter Verlauf für die Kurven aufgetragen.

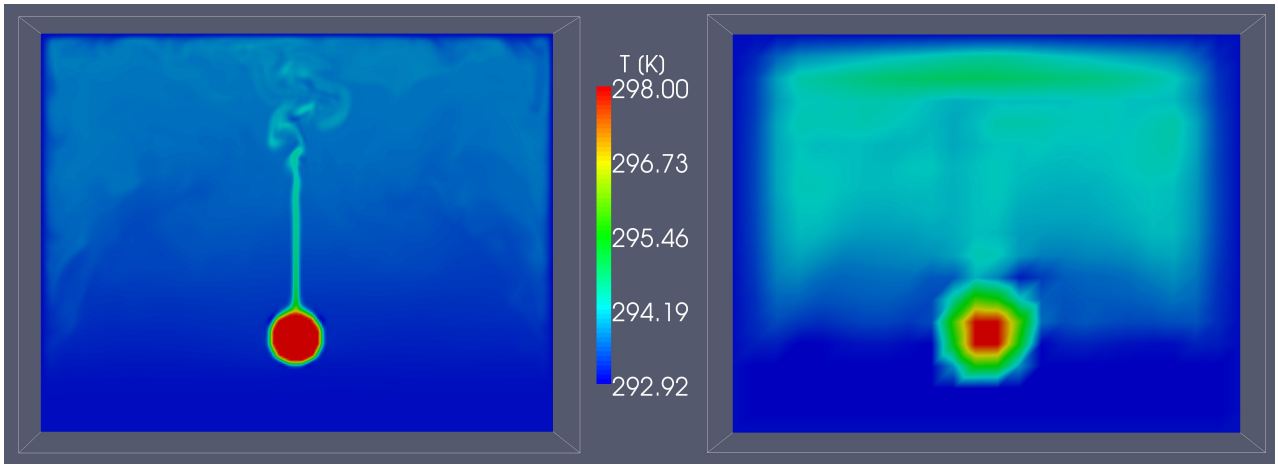


Abbildung 8.9: Temperaturverteilung im Gebiet (Schnittebene bei $y = 0,50m$). Links für $dx = 0,01m$, rechts für $dx = 0,10m$.

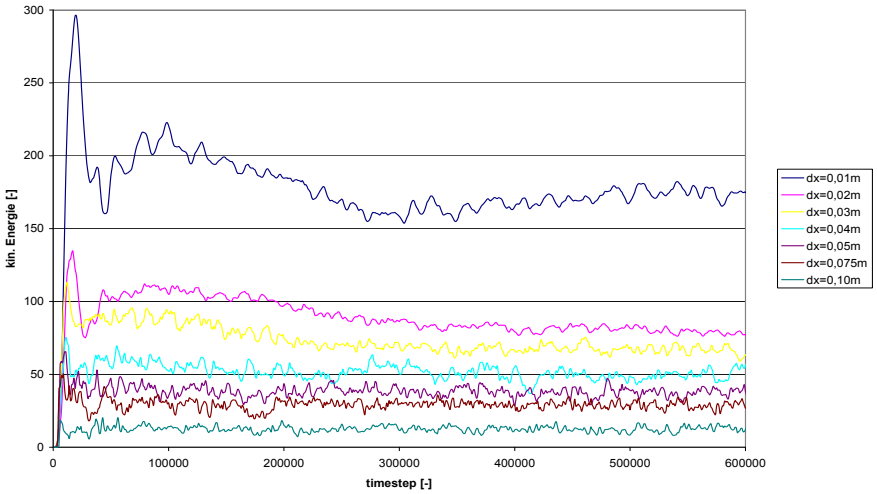


Abbildung 8.10: Zeitliche Entwicklung der kinetischen Energie im System. Dargestellt für die einzelnen Gittergrößen. Die kinetische Energie ist dimensionslos gegeben, weshalb sie für die Simulationen mit den unterschiedlichen Gitterweiten in unterschiedlichen Größenordnungen vorliegt.

Bei dem Vergleich der Ergebnisse der Simulationen mit unterschiedlichen Gittergrößen lässt sich feststellen, dass der Fehler bezogen auf die Simulation mit dem feinsten Gitter, d.h. $dx = 0,01m$, massiv zunimmt. Dies liegt zum Einen daran, dass durch das grobe Gitter das Verhalten des Fluides nicht mehr hinreichend genau abgebildet werden kann. Eine Rayleigh Zahl in dieser Größenordnng erfordert aufgrund der auftretenden Turbulenzeffekte eine feine Diskretisierung des Rechengebiets. Bei einem zu groben Netz werden im Extremfall nicht nur die Ergebnisse sehr ungenau, sondern die Simulationen instabil und liefern somit keine verwertbaren Ergebnisse.
Zum Anderen kann jedoch auch die Geometrie des Modells, in diesem Fall der beheizte Zylinder, aufgrund des groben Gitters nicht mehr hinreichend gut abgebildet werden. Hierdurch entsteht ein zusätzlicher Fehler, welcher sich in den hier betrachteten Modellen besonders negativ auswirkt, da der abzubildende Zylinder die einzige Wärmequelle im Simulationsgebiet ist und somit die über diese Quelle eingebrachte Energie der Antrieb der beobachteten Strömung ist.

8.1.4 Gegenüberstellung der Ergebnisse

Zum Vergleich der Ergebnisse aus den Simulationen mit Ansys CFX und `iFluids` wird auf CFX Seite das LES-Modell, bzw. dessen zeitliche Mittelung, und auf Seiten von `iFluids` das feine Modell mit $dx = 0,01m$ herangezogen. In `iFluids` kommt ebenfalls ein LES-Modell zur Anwendung.
Da es sich bei Strömungssimulationen immer um mehr oder weniger starke Approximationen des tatsächlichen physikalischen Problems handelt und die beiden Programme zusätzlich eine völlig unterschiedliche Herangehensweise aufweisen, sind auch die Ergebnisse nicht sofort eins zu eins vergleichbar. Aus diesem Grund werden aus den Schwankungsgrößen (Temperatur und Fluidgeschwindigkeit) zeitlich gemittelte Verläufe gebildet, die dann vergleichen werden können. Des Weiteren wird hier zusätzlich das Verhalten der Strömung als Ganzes betrachtet und bewertet.

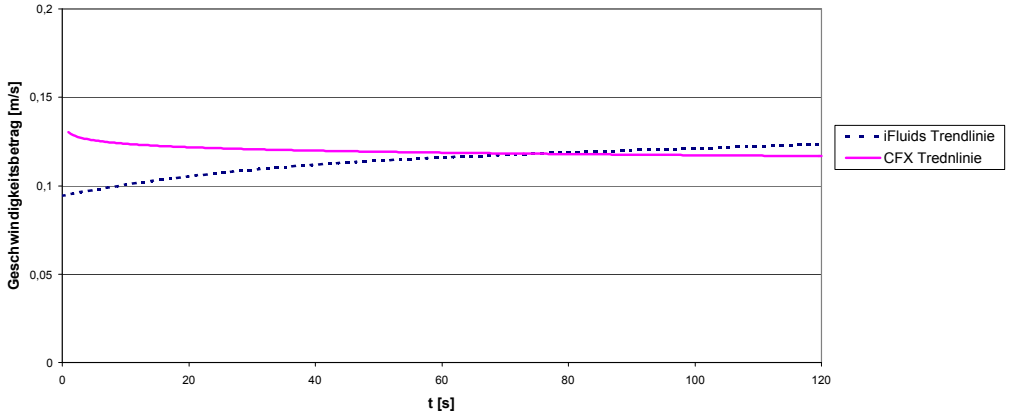


Abbildung 8.11: Zeitliche Entwicklung des Betrags der Geschwindigkeit am Punkt $P1$ für das CFX und `iFluids` Modell. Dargestellt sind die aus einer Regressionsanalyse gewonnenen Trendlinien.

Ein besonderes Problem bei der Gegenüberstellung der Ergebnisse sind die völlig unterschiedlichen Zeitskalen der beiden Simulationen. Zeitschritte in einer Lattice-Boltzmann-Simulation

sind um ein Vielfaches kleiner als Zeitschritte in einem Navier-Stokes-basierten Ansatz. Dies hat zur Folge, dass in einem LBM Ansatz wesentlich mehr Zeitschritte simuliert werden müssen, um die gleiche physikalische Zeitspanne abzubilden. In diesem Zusammenhang soll jedoch darauf hingewiesen werden, dass auch die Rechenzeit, welche zur Simulation eines Zeitschritts aufgewendet werden muss, in der LBM deutlich geringer ist.
Des Weiteren sieht deshalb auch der Einschwingvorgang der Simulationsläufe unterschiedlich aus. Dies ist gut in Abbildung 8.11 zu erkennen, welche die Entwicklung des Betrags der Geschwindigkeit am virtuellen Messpunkt $P1$ über die Zeit zeigt. In diesem Diagramm wurde der Geschwindigkeitsbetrag in physikalischen Einheiten ($[\frac{m}{s}]$) aufgetragen. Hierzu mussten die dimensionslosen Rohdaten der LBM Simulation in `iFluids` in physikalische Einheiten zurücktransformiert werden.
Die zeitlichen Mittelwerte (Mittelung über 60 Sekunden im eingeschwungenen Zustand) für den Betrag der Fluidgeschwindigkeit am Punkt $P1$ der unterschiedlichen Simulationen und der Unterschied der Ergebnisse in Relation zum Ansys CFX LES-Modell sind in folgender Tabelle dargestellt.

Modell	$\lvert\vec{u}\rvert$	Diff. [%]
CFX LES	$0,1092$	-
CFX RANS	$0,1070$	$-1,98$
`iFluids` $dx = 0,01m$	$0,1216$	$11,45$

Tabelle 8.4: Zeitliche Mittelwerte für den Betrag der Fluidgeschwindigkeit am Punkt $P1$ für verschiedene Simulationsmodelle.

In Abbildung 8.12 ist die Strömung zum Zustand $t = 60s$ für das `iFluids` und das CFX Modell abgebildet. Dargestellt ist die Geschwindigkeitsverteilung auf einer x-z-Schnittebene bei $y = 0,50m$. Die grafische Darstellung der CFX Ergebnisse erfolgt im Postprocessing Programm von CFX, während die `iFluids` Daten in Paraview visualisiert wurden.

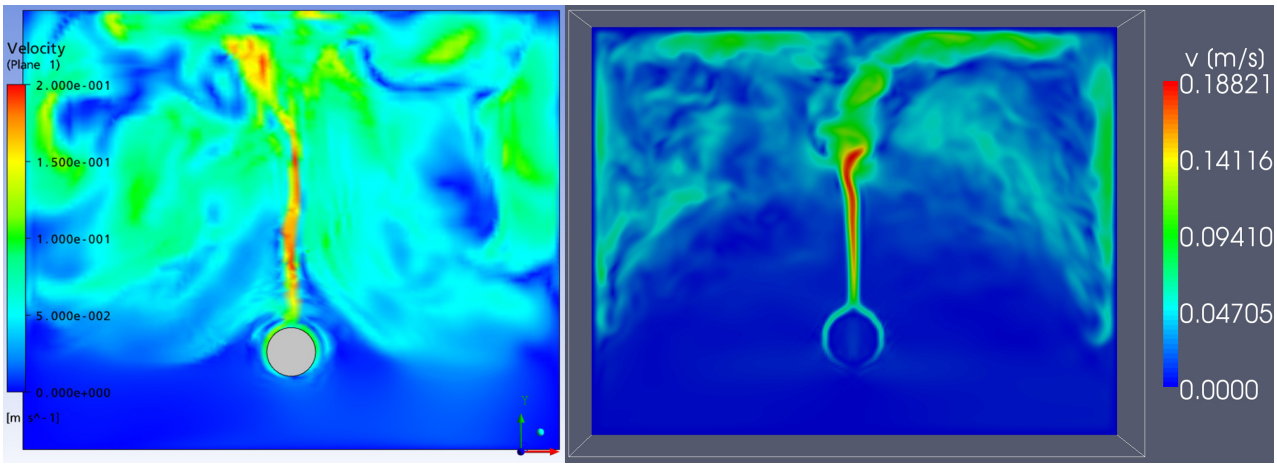


Abbildung 8.12: Geschwindigkeitsverteilung zum Zeitpunkt $t = 60s$. Visualisierug auf einer x-z-Schnittebene bei $y = 0,50m$. Links Ergebnisse aus CFX und rechts Ergebnisse aus `iFluids`.

Dargestellt sind in beiden Fällen die physikalischen Größen in der Einheit $[\frac{m}{s}]$. Die Farbskalen wurden in beiden Programmen gleich gewählt.

Beim Vergleich der Ergebnisse der `iFluids` Simulation mit den beiden Simulationen in CFX (RANS und LES) fällt auf, dass `iFluids` den Charakter der Strömung vergleichbar mit der LES Simulation in CFX abbildet. Dies war so zu erwarten, da auch im Simulationskernel von `iFluids` ein LES Turbulenzmodell zum Einsatz kommt und somit auch hier eine bessere Auflösung der entstehenden Wirbel als bei Nutzung beispielsweise eines RANS Verfahrens gewährleistet ist.

8.2 IGSSE-Projekt-Testraum

Ein weiteres zu simulierendes Problem besteht aus dem modellierten Referenzraum des interdisziplinären Projekts *Building, users, climate*, welches innerhalb der International Graduate School for Science and Engineering (IGSSE) an der Technischen Universität München angesiedelt ist.

8.2.1 Projektbeschreibung

Bei der energetischen und raumklimatischen Optimierung von Gebäuden spielt die Gebäudefassade eine Schlüsselrolle, denn sie definiert die für die Gebäudenutzung wesentlichen Eigenschaften wie Energiebedarf, Tageslichteinfall, Wärmeentwicklung und Luftwechsel und deren Interaktion ([22]). Mittels fall-spezifischer Sensitivitätsanalysen muss dementsprechend untersucht werden, inwieweit die verwendeten Konzepte und Technologien den Nutzungsanforderungen und den standortspezifischen Bedingungen Rechnung tragen. Denn je besser ein Gebäude auf die äußeren klimatischen Bedingungen reagiert, desto weniger Technik und Energie wird benötigt, um ein für die Nutzer angenehmes Klima im Inneren zu erreichen ([22]).
Das Projekt zielt auf die Entwicklung von Planungsstrategien und -methoden für den Bau von Bürogebäuden in unterschiedlichen Klimaregionen, die von Architekten und Ingenieuren gleichermaßen verwendet werden können. Diese Planungshilfen sind dabei besonders für den Einsatz in frühen Entwicklungsphasen eines Bauprojekts gedacht und sollen als Basis für Entscheidungen im weiteren Planungsverlauf dienen. Gerade in frühen Phasen, in denen oft noch keine detaillierten Planungsdaten vorliegen, werden Werkzeuge bzw. Richtlinien benötigt, die eine erste Prognose über das zu erwartende Innenraumklima und die Energieeffizienz eines Gebäudes erlauben und auch die Einflüsse und Verflechtungen zwischen den einzelnen Designvariablen berücksichtigen und auch aufzeigen können.
Als Teilprojekt wird in dieser Arbeit die konvektive Raumluftströmung und die Temperaturverteilung innerhalb des Referenzraumes untersucht. Hierbei werden die Randbedingungen durch Simulationen in anderen Teilprojekten geliefert.

8.2.2 Beschreibung des Raumes

Bei dem betrachteten Raum handelt es sich um einen typischen, rechteckigen Büroraum. Eine Wand ist als Außenwand, und somit als Teil der Fassade, definiert, die beiden daran angrenzenden Wände sind Innenwände zu angrenzenden Büroräumen und die der Fassade gegenüberliegende Wand wird ebenfalls als innen liegende Wand angenommen, welche an eine Kernzone, beispielsweise einen Korridor, grenzt. Die Abmessungen sind in nachstehender Tabelle 8.5 gegeben.
Der Fensterflächenanteil der Fassade ist variabel; der Anwender hat mehrere Alternativen zur Auswahl. Je nach Wahl des Ingenieurs wird das Modell des Raumes entsprechend angepasst. Bei der Raumkonditionierung stehen neben der Bauteilaktivierung, welche über die Oberflächentemperaturen der entsprechenden Bauteile abgebildet wird, auch eine mechanische Lüftung des Raumes zur Verfügung. Hierbei geht der Volumenstrom, und damit die Ausströmgeschwindigkeit, als zusätzliche Randbedingung in die Simulation ein.
Als Randbedingungen werden Oberflächentemperaturen gesetzt, die durch zonale Simulationen in einem anderen Teilprojekt gewonnen wurden.

Breite	$3,90m$
Tiefe	$5,325m$
Höhe	$3,00m$
Fläche	$20,77m^2$
Volumen	$62,30m^3$
Fassadenfläche	$11,70m^2$

Tabelle 8.5: Abmessungen des Referenzraumes des IGSSE-Projekts 2-8.

Für diese konkrete Referenzsimulation wurden Oberflächentemperaturen, die für den Standort Shanghai mit dem Simulationsprogramm *ESP-r*[2] gewonnen wurden, genutzt. Die *ESP-r*-Simulation liefert für den Raum acht Oberflächentemperaturen: Die beiden Seitenwände, die Rückwand, die Außenwand, die Decke sowie den Boden, die Verglasung und den Fensterrahmen. Die Temperaturwerte für diese Flächen liegen für ein ganzen Jahr (365 Tage) im Stundenraster, beginnend am 01.01. um 0 Uhr 30, vor.
Als Fensterflächenanteil wurde ein Wert von 30% gewählt; dies resultiert in einer Fensterfläche von $3,51m^2$. Da das Fenster für den Testraum nicht näher definiert ist, wurden geeignete Annahmen getroffen. Das Fenster wurde somit mit einer Breite von $2,70m$ und einer Höhe von $1,30m$ modelliert. Die Brüstungshöhe wurde auf $0,90m$ festgelegt. Für die Rahmen wurden eine Breite von $0,05m$ gewählt.
Es wurden die gewonnenen Werte für den 30. Juni um 12 Uhr 30 verwendet. Die entsprechenden Oberflächentemperaturen sind in Tabelle 8.6 gegeben.

Seitenwand 1	25,69
Seitenwand 2	25,69
Rückwand	25,69
Außenwand	27,48
Decke	25,73
Boden	26,08
Verglasung	27,60
Fensterrahmen	28,46

Tabelle 8.6: Oberflächentemperaturen für die Simulation in [°C].

Es wurde ein Problem mit mechanischer Lüftung und einer Luftwechselrate von 1,5 h^{-1} gewählt und simuliert. Die Temperatur der Zuluft beträgt 20°C. Es wurde eine Zuluftgeschwindigkeit von $v = 1,0\frac{m}{s}$ gewählt. Zusätzlich ist der Luftstrom um einem Winkel von 20° zur Decke hin gerichtet. Eine Abbildung des Raumes mit zugeordneter Beschriftung der einzelnen Segmente ist in 8.13 gegeben.
Es wurden zwei virtuelle Messpunkte in der Mitte des Gebiets definiert, an denen Ergebnisdaten ausgewertet wurden. Die y- und z-Koordinaten sind jeweils für beide Punkte identisch und liegen bei $1,95m$ bzw. $1,00m$. Der erste Punkt wurde bei $x = 1,775m$ und der zweite Punkt bei $y = 3,55m$ gesetzt. An den Punkten werden sowohl Geschwindigkeiten als auch Temperatur des Fluids erhoben.

[2]Hierbei handelt es sich um ein dynamisches Gebäudesimulationsprogramm (siehe [16]).

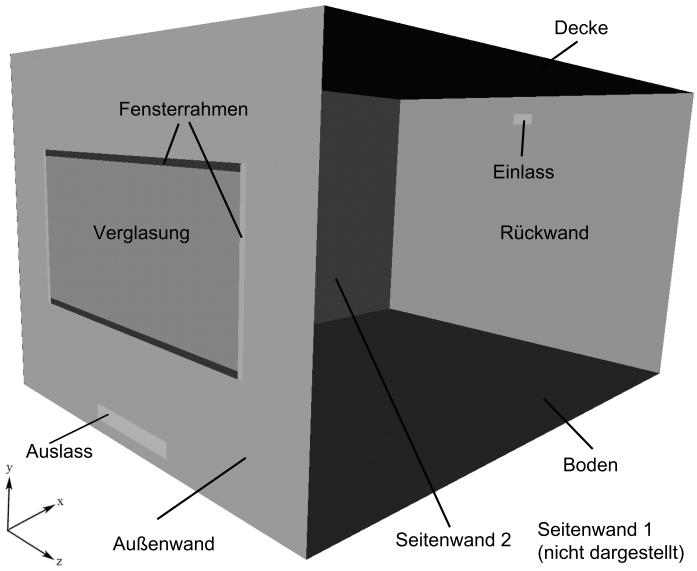


Abbildung 8.13: Referenzraum mit Unterteilung in Segmente.

8.2.3 Aufbereitung der CAD-Daten

Für die Referenzsimulation wurde die reine Geometrie des Raumes zuerst in einem CAD-Programm – im konkreten Fall AutoCAD – modelliert und anschließend als STL-Datei im ASCII-Format exportiert. Durch diesen Exportprozess entsteht eine Datei in welcher das gesamte Modell als ein Objekt („solid“) abgebildet ist. Für die Definition von Randbedingungen auf einzelnen Segmenten ist jedoch die Unterteilung in verschiedene „solid“-Abschnitte innerhalb der STL-Datei erforderlich. Diese werden von der Simulationsumgebung `iFluids` als Unterobjekte des übergeordneten Geometrieobjekts erkannt, wodurch auf diesen separate Randbedingungen definiert werden können.
Zur Unterteilung der facettierten Geometrie in einzelne Regionen und zur grafischen Definition von Randbedingungen kam die *iFluidsAttribute*-Erweiterung des Visualisierungsprogramms Amira zum Einsatz (für eine Beschreibung von *iFluidsAttribute* siehe [32]).
Zusätzliche Arbeiten an den facettierten Geometriedaten war nicht erforderlich, da die in `iFluids` eingesetzte Gittergenerierung sehr robust gegenüber kleinen Klaffungen, etc. ist (siehe 3.6.3).

8.2.4 Referenz-Simulation mit CFX

Für die Simulation mit dem Programm CFX wurde abweichend von der Simulation in `iFluids` nicht das Geometriemodell, welches durch ein CAD-System erzeugt wurde, verwendet, sondern die Geometrie wurde direkt im Präprozessor modelliert.
Das erstellte Modell ist in Abbildung 8.14 dargestellt. Ebenfalls darin dargestellt ist die Lage der beiden virtuellen Messpunkte im Raum.
Die Simulation wurde in CFX mit einer automatischen Wahl der Zeitschrittweite bis zum Erreichen einer auskonvergierten Lösung durchgeführt. Als Turbulenzmodell wurde ein k-epsilon Modell gewählt. Die Netzweite liegt zwischen $0,03$ und $0,05m$. Ein feineres Modell konnte auf der zur Verfügung stehenden Hardware nicht erstellt werden. Zudem hätte eine noch feinere

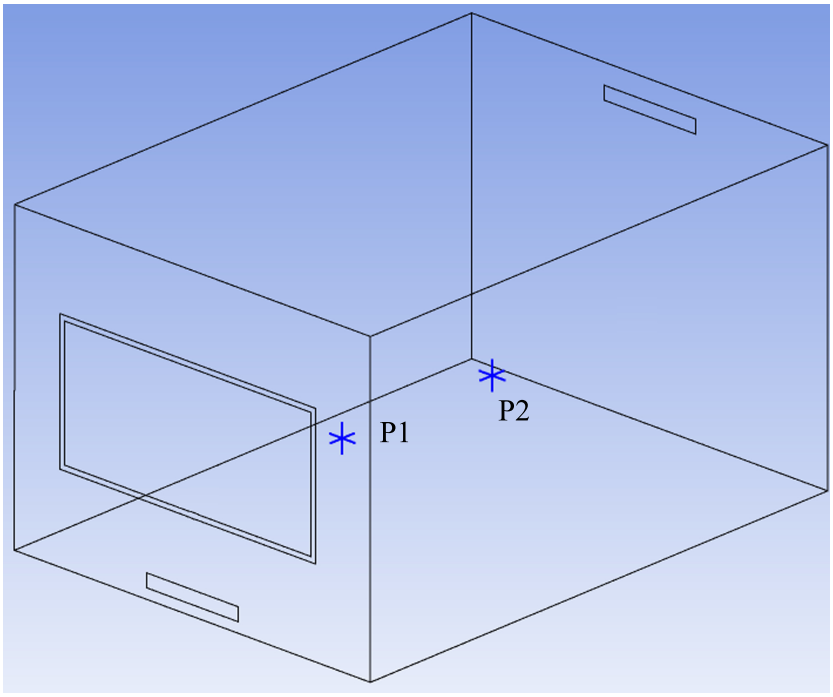


Abbildung 8.14: Modell des Referenzraum in CFX; Lage der virtuellen Messpunkte.

Wahl des Rechengitters die ohnehin schon sehr langen Rechenzeiten (mehrere Tage auf zwei CPU Kernen) nochmal um ein Vielfaches erhöht.
Abbildung 8.15 zeigt die Geschwindigkeitsverteilung im Raum in einer Schnittebene bei $y = 1,95m$.

8.2.5 Simulation mit `iFluids`

Als Rechnerplattform für die Simulationen in `iFluids` wurde ebenfalls wieder der Bundeshöchstleistungsrechner II (siehe 7.2) am LRZ in Garching bei München gewählt. Auch hier ist eine direkte Interaktion mit der Simulation nicht nötig, weshalb die Visualisierungsapplikation ebenfalls durch eine Applikation ohne GUI ersetzt wurde, welche nur für das Einlesen der STL-Datei des modellierten Problems und für die Ausgabe der Ergebnisdaten auf Datenträger zuständig ist. Die Resultate der Simulation wurden alle $n = 1000$ Zeitschritte von der Simulationsapplikation an die Dummy-Visualisierungsapplikation übertragen. An den beiden Messpunkten wurden an jedem Update-Zeitschritt n die Geschwindigkeitskomponenten u, v, w sowie die Temperatur T auf Datenträger ausgegeben. Zusätzlich werden an jedem zehnten Update-Zeitschritt die kompletten Ergebnisdaten dieses Zeitschritts gespeichert.
Die Simulationen wurden als batch jobs über das Wartelistensystem des HLRB II gestartet. Als feinste Auflösung wurde eine Gitterweite von $dx = 0,01m$ gewählt.
Neben der feinen Simulation mit einer Gitterweite von $dx = 0,01m$ wurden auch Simulationsläufe mit gröberen Auflösungen durchgeführt. Dies geschah, um Rückschlüsse hinsichtlich der Genauigkeit einer grob aufgelösten interaktiven Simulation machen zu können. Hierbei blieben die sonstigen geometrischen Randbedingungen gleich, nur die Gitterauflösung wurde verändert. Wie bei allen Simulationen in `iFluids` kam ein kartesisches Rechengitter mit $dx = dy = dz$ zur Anwendung. Bei der feinsten Auflösung ergeben sich 534 Voxel in x-

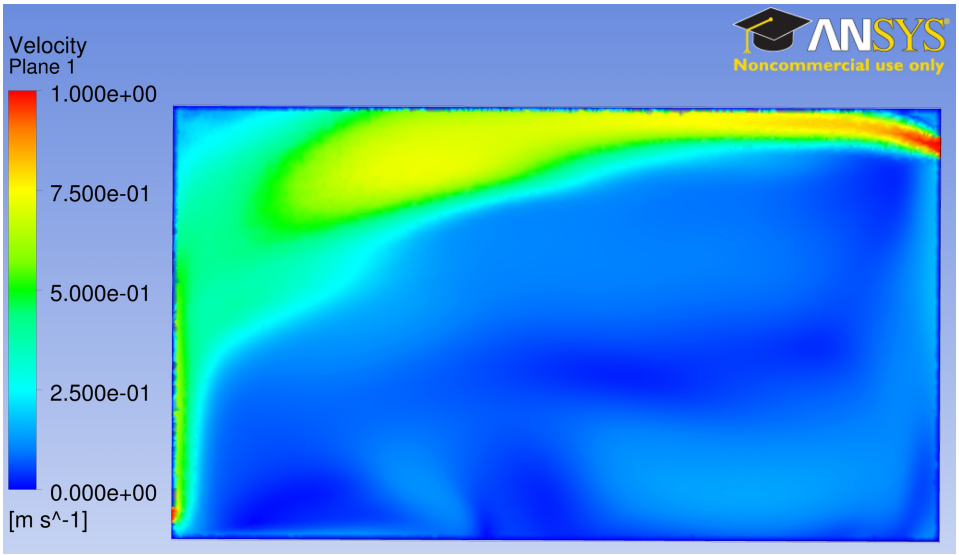


Abbildung 8.15: Schnittebene bei $y = 1,95m$ durch das Simulationsgebiet. Es wird die Geschwindigkeit (Magnitude) dargestellt (Ansys CFX).

Richtung, 392 Voxel in y-Richtung und 301 Voxel in z-Richtung.

In der nachfolgenden Tabelle sind die zusätzlich durchgeführtes Simulationen mit den dazugehörigen Gitterweiten und der daraus resultierenden Anzahl der Voxel in den betreffenden Modellen aufgelistet:

Gitterweite dx [m]	Anzahl der Voxel in der Simulation
$0,01$	63007728
$0,03$	2368349
$0,05$	520452
$0,075$	156456
$0,10$	68200

Tabelle 8.7: Auflösungen der durchgeführten Simulationen und Voxelanzahl in `iFluids`.

Als charakteristische Länge wurde auch aus Gründen der Vergleichbarkeit auch in `iFluids` der automatische Wert aus Ansys CFX eingestellt. Somit gilt: $l_{char} = 3,9643m$.

Zur Parallelisierung wurde das Standard Altix-MPI verwendet. Als Rechenkernel kam der moderat optimierte hTLB3DQ15-Simulationskernel ohne OpenMP-Erweiterung zur Anwendung. Aufgrund der im Optimierungsprozess des Codes gesammelten Erfahrung wurde die Applikation explizit auf einer *bandwidth* Partition des HLRB II platziert.

In Abbildung 8.16 ist die Geschwindigkeitsverteilung in der Fluiddomäne zum Zeitschritt 230.000 mittels einer x-z-Schnittebene in der Mitte, d.h. $y = 1,95m$, des Gebiets dargestellt. Zur grafischen Auswertung der Ergebnisse wurde die Applikation Paraview benutzt, da die OpenInventor basierte Online-Visualisierung der CSE wie bereits beschrieben für diese Rechnungen deaktiviert wurde.

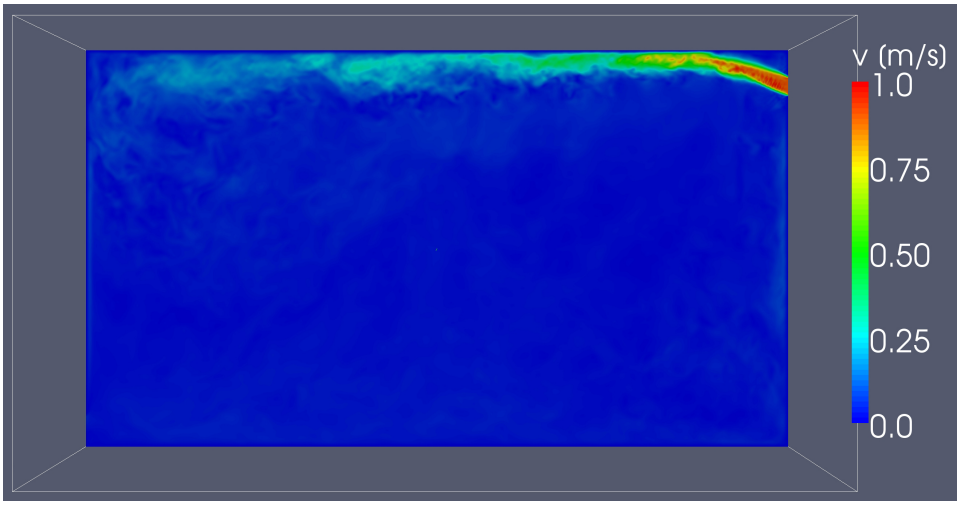


Abbildung 8.16: Schnittebene bei $y = 1,95m$ durch das Simulationsgebiet bei $dx = 0,01m$. Es wird die Geschwindigkeit (Magnitude) dargestellt (iFluids).

Die gewonnenen Simulationsdaten werden LBM-typisch als dimensionslose Größen ausgegeben. Die Rückrechnung in physikalische Größen erfolgt in einem manuellen Postprocessingschritt in der Visualisierungsapplikation.

8.2.6 Auswertung

Bei der Auswertung und dem Vergleich der beiden Simulationen in `iFluids` und Ansys CFX für dieses Beispielproblem ist eine eins zu eins Gegenüberstellung der Ergebnisse aufgrund der unterschiedlichen Simulationstypen schwierig. In Ansys CFX kam ein RANS Modell zur Anwendung, wohingegen in `iFluids` immer ein LES Turbulenzmodell eingesetzt wird. Somit sind die Turbulenzeffekte in den Daten aus `iFluids` wesentlich deutlicher zu erkennen, wie in 8.17 erkennbar ist.

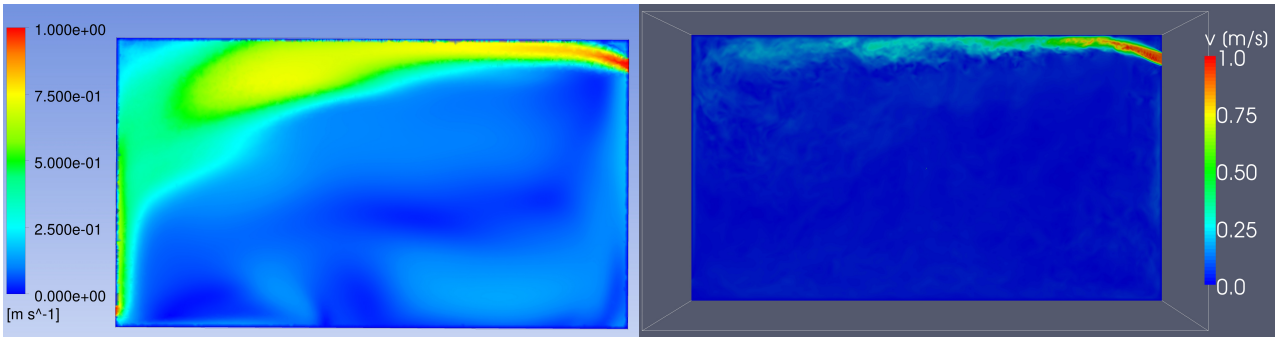


Abbildung 8.17: Schnittebene bei $y = 1,95m$ durch das Simulationsgebiet. Links ist das visualisierte Ergebnis in Ansys CFX und rechts die Ergebnisse aus `iFluids` (visualisiert mit Paraview) bei einer Gitterweite von $dx = 0,01m$ dargestellt. Es ist die Geschwindigkeit (Magnitude) aufgetragen.

In den Daten aus `iFluids` sind kleinere Wirbel im Simulationsgebiet gut zu erkennen, wohingegen in Ansys CFX diese zu großflächigen Strukturen gemittelt wurden.

Besonders hervorzuheben und besonders interessant und wichtig ist die Abbildung des Coandă-Effekts. Hierunter versteht man in diesem Fall das sich Anlegen des Luftstrahl an die Decke. Hierbei handelt es sich um einen (meist erwünschten) Effekt bei der Auslegung von Lüftungssystemen. Die gekühlte Luft dringt somit tiefer in den Raum ein und sackt nicht vorzeitig ab. Bei der erreichten Luftverteilung sind auch keine Zuglufterscheinungen im Arbeitsbereich von Nutzern zu erwarten. Dies ist gut bei der höchsten Auflösung von $dx = 0,01m$ in Abbildung 8.16 zu sehen.
Dieser Effekt lässt sich jedoch schon bei einer Gitterweite von $dx = 0,03m$ in `iFluids` nicht mehr in diesem Ausmaß beobachten. Abbildung 8.18 zeigt die Temperaturverteilung. Links sind die Daten aus Ansys CFX und rechts die Daten aus `iFluids` mit $dx = 0,03m$ abgebildet. Die Visualisierung der Temperaturverteilung aus `iFluids` zeigt deutlich, dass die gekühlte Zuluft wesentlich früher als in Ansys CFX in den Raum abfällt. Somit ist festzustellen, dass schon bei dieser Gitterweite keine korrekte Aussage über die Luftverteilung mehr gemacht werden kann. Wie zu erwarten ist, ist auch in den Daten der noch gröber aufgelösten Simulationen dieser frühe Abfall der Zuluft zu beobachten.

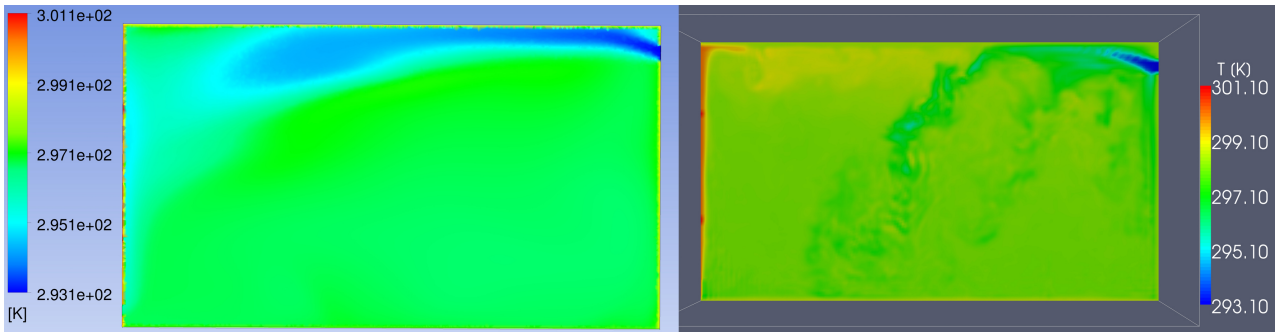


Abbildung 8.18: Schnittebene bei $y = 1,95m$ durch das Simulationsgebiet. Links ist das visualisierte Ergebnis in Ansys CFX und rechts die Ergebnisse aus `iFluids` bei $dx = 0,03m$ (visualisiert mit Paraview) dargestellt. Es ist die Temperaturverteilung aufgetragen.

8.3 Rückschlüsse auf die interaktive Simulation

Basierend auf den oben dargelegten Untersuchungen lassen sich Rückschlüsse für die Nutzung der interaktiven Simulation ziehen. Hierbei ist jedoch immer zwischen den Zielen einer flüssigen Simulation und Visualisierung und möglichst korrekten Ergebnissen abzuwägen. Generell ist zu beachten, dass durch die Nutzung des MRT Verfahrens und durch die Betrachtung der thermischen Effekte die Ansprüche an die Rechenleistung stark steigen bzw. die mögliche Auflösung sinkt. Generell sollte versucht werden, die Gitterweite kleiner als $0,05m$ zu wählen, wobei sich bei der Auswertung der Ergebnisse gezeigt hat, dass in den hier untersuchten Beispielen Auflösungen von unter $0,03m$ nötig waren, um manche Strömungseffekte abbilden zu können. Hierbei handelt es sich jedoch nur um Richtwerte, da die nötige Auflösung auch stark von der zu untersuchenden Strömung und dem Anspruch an die Simulation abhängt. So kann für eine erste Einschätzung der Luft- oder Temperaturverteilung im Raum auch eine Gitterweite von $0,05m$ oder größer ausreichen.
Die benötigte Rechenleistung, bzw. die Anzahl der benötigten Rechenknoten, hängt direkt

von der Größe des Gebiets ab. Kapitel 7 zeigt die Skalierbarkeit des Simulationscodes und gibt so Anhaltswerte für die Anzahl der tatsächlich für ein Problem benötigten Rechenknoten. Generell ist jedoch davon auszugehen, dass vier oder besser acht Knoten das untere Minimum darstellen.

Kapitel 9

Industrielle Anwendungsbeispiele

Ergänzend zu den bereits vorgestellten Benchmark Simulationen wurden zwei weitere Anwendungsbeispiele simuliert. Diese sollen mögliche, weitere Anwendungsfelder zeigen. Zum Einen der Innenraum eines Zugabteils und zum Anderen ein Büroraum, welcher im Kontext des Forschungsprojektes ComfSim ([64]) betrachtet wurde.

9.1 Büroraum mit Fensterlüftung

Bei diesem weiteren praxisrelevanten Anwendungsbeispiel wurde ein neuartiges Fenstersystem mit vertikalem Schiebemechanismus betrachtet. In Deutschland kommen sehr häufig sowohl im Wohnungsbau als auch bei Gewerbebauten Kipp-/Drehflügelfenster zum Einsatz. Diese Lüftungselemente sind auf der einen Seite kostengünstig, weisen aber auch eine Reihe von Nachteilen, wie beispielsweise Zugerscheinungen im Winter und zu geringe Lüftungsintensität für sommerliche Nachtlüftung auf. Um eine Fensterlüftung mit weniger Problemen zu erreichen, wurde von der Fassadenbau-Firma Schindler ein kostengünstiges Fenster mit neuartigem Schiebemechanismus entwickelt, deren Auswirkungen auf den thermischen Komfort in Büroräumen anhand von Simulationen detailliert untersucht wurde.
Hierbei wurden die Untersuchungen im Kontext des Forschungsprojektes ComfSim durch den Industriepartner MüllerBBM durchgeführt. Ergänzend zu diesen Untersuchungen wurde der Raum auch mit der CSE `iFluids` simuliert. Hierbei wurde die Geometrie wie in Abbildung 9.1 dargestellt modelliert. Die Randbedingungen für diese Simulation sind in nachfolgender Tabelle 9.1 gegeben.
Die Abmessungen des Raumes betragen $x = 5,25m$, $y = 2,8m$ (Raumhöhe) und $z = 2,6m$. Die Lüftungselemente befinden sich in der Wand in yz-Ebene bei $x = 0m$.
Es wurde nur die sich einstellende Luftströmung im Innenraum untersucht. Zu diesem Zweck wurde an einer Öffnung des Lüftungselements eine konstante Einströmgeschwindigkeit mit $u = 0,19\frac{m}{s}$ vorgegeben, wohingegen an der zweiten Öffnung eine Druckrandbedingung modelliert wurde.
Die Diskretisierung des Geometriemodells erfolgte mit einer Gitterweite $\Delta x = 0,025$. Dies ergibt ein Modell mit 210 x 112 x 102, d.h. 2.399.040 Gitterzellen. Die Berechnung erfolgte zum Einen interaktiv, wobei die sich ausbreitende Strömung studiert wurde und zum Anderen als offline Simulation, wobei hier der Strömungszustand in einem konvergierten, späten Zustand von Interesse war. Bei dieser Simulation wurden in einem definierten Zeitschrittintervall die

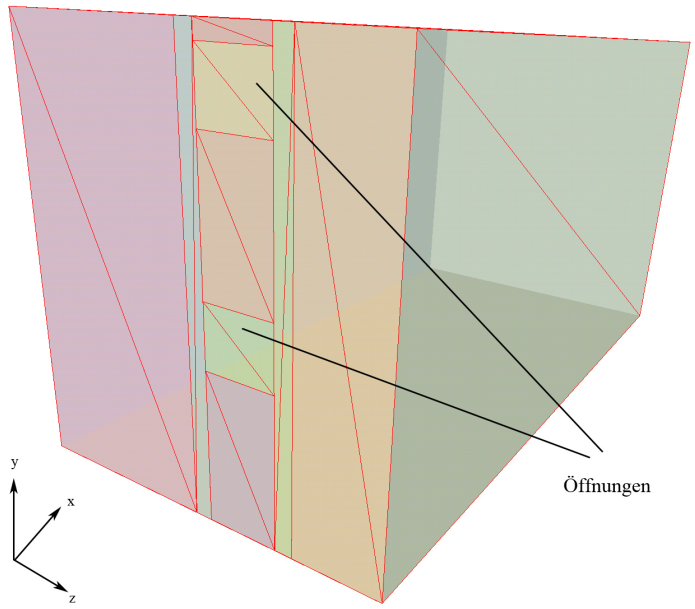


Abbildung 9.1: Geometrie des als Teil des ComfSim Projekts simulierten Büroraums.

Simulationsergebnisse zur späteren Auswertung in Form einer Paraview Datei ausgegeben. In Abbildung 9.2 und 9.3 ist die Strömungssituation nach 500.000 Lattice-Boltzmann-Zeitschritten in einer Schnittebene in xy-Ebene dargestellt. Es ist der Betrag der Geschwindigkeit und die Temperaturverteilung visualisiert. Hierbei ist zu erkennen, wie die kühlere Luft durch das Lüftungselement einströmt und rasch zu Boden sinkt, wo sie sich ausbreitet. Die warme Luft im Raum wird langsam von unten her von kühlerer Luft verdrängt. Die warme Raumluft kann durch die obere Öffnung des Lüftungselements ungehindert ins Freie entweichen. Zusätzlich ist zu erkennen, wie durch die kühlere Decke abgekühlte Luft nach unten sinkt und so zu einer zusätzlichen Kühlung von oben sorgt. Ergänzend führt dieser Abfall von gekühlter Luft zu einer Luftbewegung im gesamten Raum, wie in Abbildung 9.2 gut zu erkennen ist.

Bauteil	**T [°C]**
Boden	30,27
Decke	29,7
Rahmen links	33,22
Rahmen rechts	33,22
Rückwand	31,22
Schiebeflügel 3	33,22
Schiebeflügel 5	33,22
Schiebeflügel 7	33,22
Seitenwand links	37,61
Seitenwand rechts	37,1
Außenluft	25

Tabelle 9.1: Thermische Randbedingungen der Simulation.

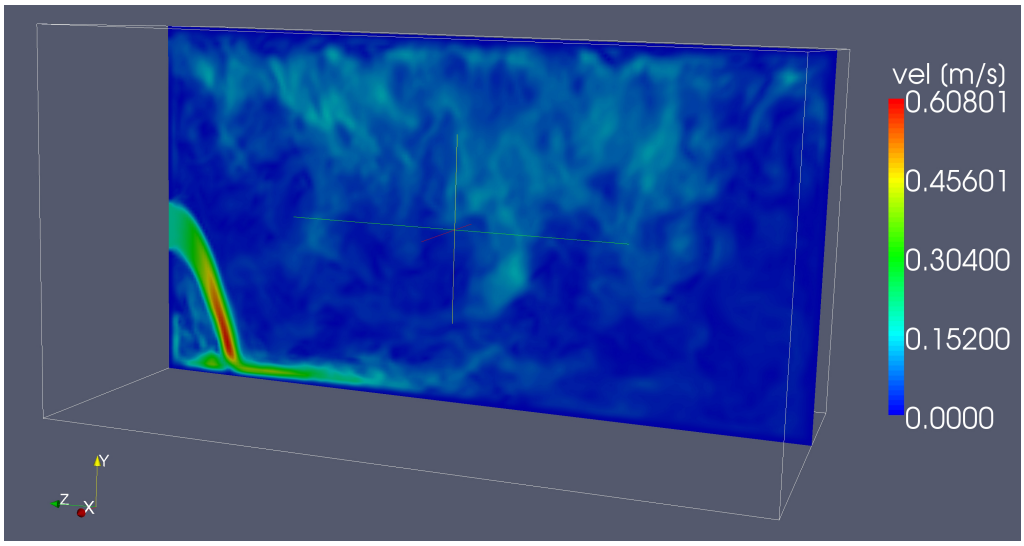


Abbildung 9.2: Schnittebene zur Visualisierung der Geschwindigkeitsverteilung der Raumluftströmung.

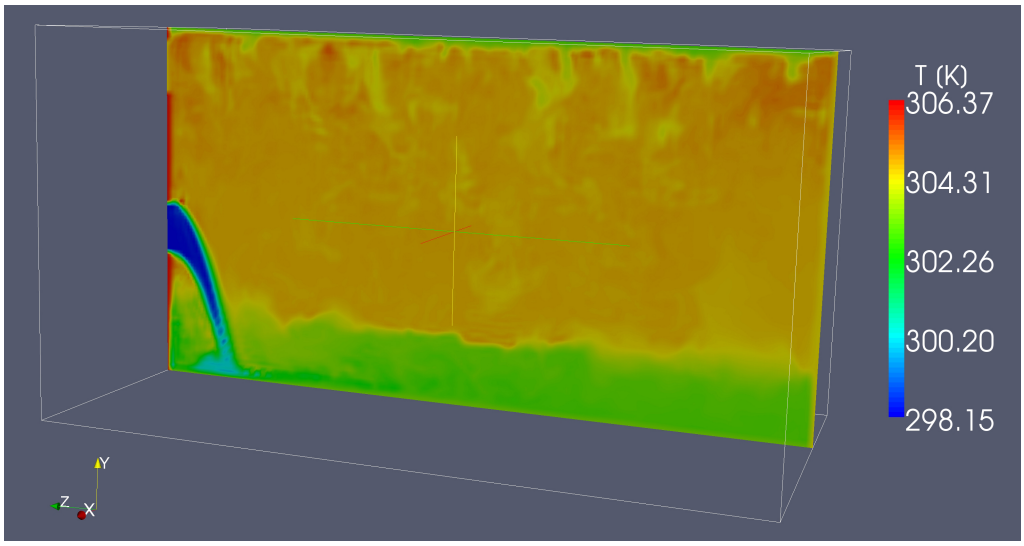


Abbildung 9.3: Schnittebene zur Visualisierung der Temperatur der Raumluft.

9.2 Innenraum eines Zugabteils

Zur Evaluation von industriellen Anwendungsfällen wurde ein Abschnitt eines Siemens Desiro Zuges modelliert und mit `iFluids` simuliert (siehe [66]).
Das Modell besteht aus 173.073 Facetten und die Abmessungen des Geometrie betragen $x = 14,27m$, $y = 2,25m$ und $z = 2,75m$. Es handelt sich um ein Problem mit freier Konvektion, welche durch Heizelemente angetrieben wird. Die Heizelemente befinden sich unter den Sitzen entlang der rechten und linken Außenwand und stellen die einzigen Wärmequellen im Modell dar. Die menschlichen Dummyobjekte geben keine Wärme ab. Die Simulation hat eine Auflösung von $dx = 0,03m$ und umfasst 3.371.436 Voxel.
Die facettierte Geometrie des Innenraums ist in Abbildung 9.4 gezeigt. Zusammen mit der Geometrie wurde die Luftströmung in zwei Schnittebenen visualisiert.

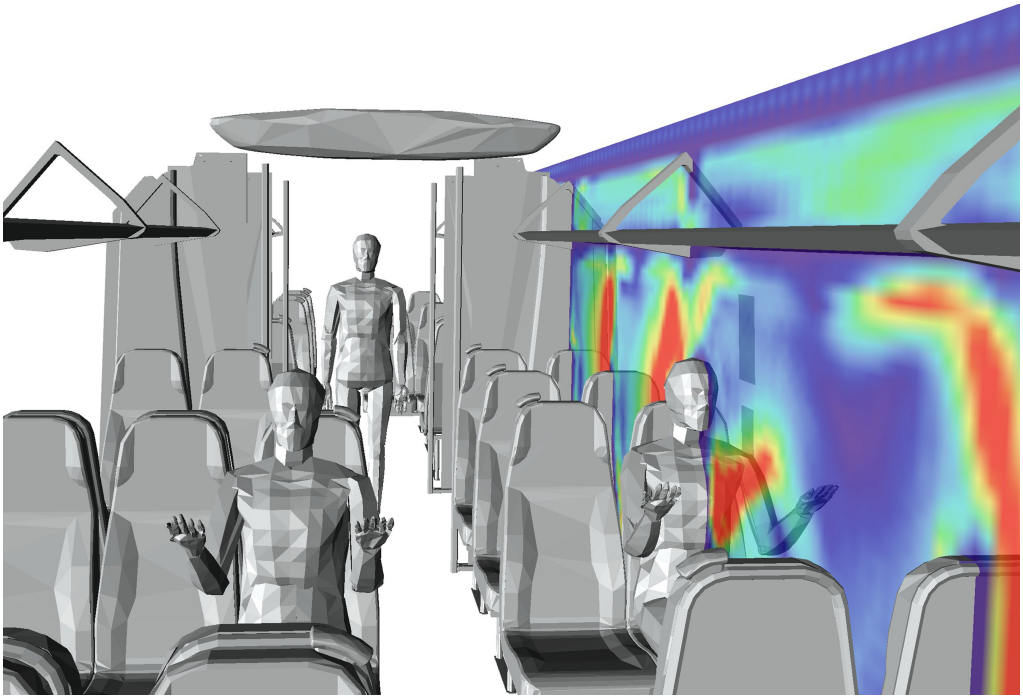


Abbildung 9.4: Teil des Innenraums eines Siemens Desiro Zuges. Visualisierung der Luftströmung mittels zweier Schnittebenen (siehe [66]).

Gut zu erkennen ist die aus der Beheizung resultierende Luftverteilung im Großraumabteil des Desiro Zuges. Hiermit lässt sich gut die Wirkung der Heizelemente analysieren und es werden Rückschlüsse hinsichtlich des thermischen Komforts möglich. Hierzu gehören beispielsweise auch Analysen von sich möglicherweise ausbildenden Temperaturschichten, die von den Passagieren als unangenehm empfunden werden könnten.

Kapitel 10

Zusammenfassung und Ausblick

In dieser Arbeit wurden verschiedene Aspekte der interaktiven, wissenschaftlichen Strömungssimulation beleuchtet. Insbesondere wurde die Nutzung von Hochleistungsrechnern, die wissenschaftliche Datenvisualisierung und die Validierung der Ergebnisse detailliert betrachtet. Im Gegensatz zu anderen CSE Ansätzen, wie sie auch in dieser Arbeit vorgestellt wurden, bietet `iFluids` die Möglichkeit, Geometrieobjekte, wie sie ohne zusätzlichen Aufwand aus CAD Anwendungen exportiert werden können, direkt in die Strömungssimulation zu laden. Diese lassen sich anschließend zur Laufzeit der Simulation noch direkt im Visualisierungsinterface transformieren und mit modifizierten Randbedingungen versehen. Andere Ansätze erlauben bisher, sofern Änderungen an der Geometrie überhaupt möglich sind, nur die Verwendung von vorher fest definierten Geometrieobjekten.
Basierend auf den Arbeiten von Wenisch ([68]) und von van Treeck ([65]) wurde in dieser Arbeit die CSE `iFluids` mit einem verbesserten Simulationskern ausgestattet, welcher nun u.a. auch thermische Strömungssimulationen ermöglicht. Der Simulationskern wurde als Teil dieser Arbeit für die Hardwareplattform SGI Altix 4700 (HLRB II) optimiert. Hierbei wurde auch der Einsatz von OpenMP im Sinne einer hybriden Parallelisierungsstrategie als Ergänzung zur Parallelisierung mittels MPI untersucht. Es wurde jedoch festgestellt, dass eine hybride Parallelisierung auf dem HLRB II mit massiven Leistungsproblemen zu kämpfen hat und deshalb zumindest auf dieser Hardwareplattform nicht vernünftig eingesetzt werden kann. Eine vergleichende Untersuchung der Leistung dieser Parallelisierungsstrategie auf einer Workstation mit acht CPU-Kernen zeigte jedoch, dass dieses Konzept auf anderen Systemen durchaus noch Potenzial haben kann.
Ergänzend wurde ein weiterer Schwerpunkt auf den Vergleich des hTLB3DQ15-Simulationskerns in `iFluids` mit einem komerziellen CFD Tool gelegt. Dies war von besonderer Wichtigkeit, um die Güte der durch die Simulation gewonnenen Daten bewerten zu können. Hierzu wurden zwei konkrete Probleme betrachtet. Bei dem Ersten handelte es sich um ein reines Benchmark Beispiel. Hierbei wurde nur natürliche Konvektion gerechnet. Bei dem zweiten Beispiel wurde die Strömungssituation in dem Standardbüroraum des IGSSE Projekts 2-8 (Buildings, users, climate) unter mechanischer Lüftung und unter gegebenen thermischen Randbedingungen analysiert. Die Referenzlösungen für beide Problemstellungen wurden mit dem kommerziellen und validierten CFD-Programm Ansys CFX gewonnen. Hierbei zeigte sich, dass die Simulationen in `iFluids` den Strömungscharakter gut abbilden und durchaus für praxisrelevante Fragestellungen im Bereich der Raumluftströmung eingesetzt werden können. Bei dem Zylindermodell wurden zur Quantifizierung der Unterschiede zusätzlich die Strömungsgrößen

an zwei definierten virtuellen Messpunkten aufgezeichnet. Hierbei wurde festgestellt, dass die mit `iFluids`[1] gewonnenen Daten nur um ca. 10% von den Werten, welche mit Ansys CFX gewonnen wurden abweichen. In Anbetracht der wesentlich aufwendigeren numerischen Modelle und der besseren Geometrieabbildung auf Seiten von Ansys CFX ist das Ergebnis als durchaus gut zu bewerten.
Am Beispiel des Zylindermodells konnte gezeigt werden, dass die grundlegende Strömungscharakteristik trotz einer groben Auflösung noch abgebildet werden kann. Die hierbei erzielten quantitativen Größen sind zwar mit einem größeren Fehler behaftet, jedoch kann basierend auf den groben Ergebnissen dann eine feiner aufgelöste, nicht interaktive Simulation stattfinden.
Anhand der beiden industriellen Anwendungsbeispiele wurden mögliche Einsatzbereiche für diese Simulationsumgebung gezeigt. Diese reichen also von der Simulation von Innenräumen in Gebäuden bis hin zu diversen Fahrzeugen.
Durch den Trend zu immer leistungsfähigerer Rechnerhardware, zum einen für die Simulation und zum anderen für die Visualisierung, ist für die Zukunft zu erwarten, dass die noch interaktiv zu simulierende Auflösung des Rechengebietes stetig zunimmt. Vor allem auch, wenn dedizierte Visualisierungsserver, wie z.B. die rvs1 am LRZ, mit einer sehr schnellen Anbindung an Hochleistungsrechner zum Einsatz kommen.
Mit der Integration des numerischen Dummymodells in die CSE wurde zudem die Grundlage geschaffen, um thermische Komfortmodelle in die CSE mit einzubinden und die Komfortbewertung innerhalb des Visualisierungsinterfaces als Farbcodierung auf dem Dummy darzustellen.

[1] Es wurde das Modell mit $dx = 0,01m$ betrachtet.

Literaturverzeichnis

[1] M. Bader. *Raumfüllende Kurven, Unterlagen zur Vorlesung „Algorithmen des wissenschaftlichen Rechnens“*, 2004.

[2] R. Belleman. *Interactive Exploration in Virtual Environments*. Dissertation, Universiteit van Amsterdam, 2003.

[3] P. Bhatnagar, E.P. Gross und M.K. Krook. A model for collision processes in gases. *Physical Review*, 94(3):511–525, 1954.

[4] A. Borrmann. *Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken*. Dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, 2007.

[5] A. Borrmann, P. Wenisch, C. van Treeck und E. Rank. Collaborative hvac design using interactive fluid simulations: A geometry-focused collaboration platform. In: *Proc. of the 12th Int. Conf. on Concurrent Engineering*, 2005.

[6] A. Borrmann, P. Wenisch, C. van Treeck und E. Rank. Collaborative computational steering: Principles and application in HVAC layout. *Integrated Computer-Aided Engineering (ICAE)*, 13(4):361–376, 2006.

[7] M. Breuer. *Direkte numerische Simulation und Large-Eddy Simulation turbulenter Strömungen auf Hochleistungsrechnern*. Professorial dissertation, Lehrstuhl für Strömungsmechanik, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2001.

[8] K. Brodlie, J. Wood, D. Duce und M. Sagar. gviz: Visualization and computational steering on the grid. *UK e–Science All Hands Meeting*, pages 54–60, 2004.

[9] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning und A. R. Porter. Computational steering in realitygrid. In: *Proc. of the UK e-Science All Hands Meeting*, 2003.

[10] H.-J. Bungartz, M. Griebel und C. Zenger. *Einführung in die Computergraphik: Grundlagen, Geometrische Modellierung, Algorithmen*. Vieweg Verlag, 2002.

[11] S. Chapman, D. Burnett und T. G. Cowling. *The mathematical theory of non–uniform gases*. Cambridge University Press, Cambridge, 1970.

[12] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter und S. M. Pickles. Steering in computational science: mesoscale modelling and simulation. *Contemporary Physics*, 44(5):417–434, 2003.

[13] D. R. Croft und David G. Lilley. *Heat transfer calculations using finite difference equations*. Applied Science Publishers, London, 1977.

[14] B. Crouse. *Lattice-Boltzmann Strömungssimulationen auf Baumdatenstrukturen*. Dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, 2003.

[15] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand und L.-S. Luo. Multiple-relaxation-time lattice Boltzmann models in three dimensions. *Phil. Trans. R. Soc. Lond. A*, 360:437–451, 2002.

[16] University of Strathclyde Energy Systems Research Unit. *ESP–r*, 2010. http://www.esru.strath.ac.uk/Programs/ESP-r_ger.htm.

[17] D. Fiala. *Dynamische Simulation des menschlichen Wärmehaushalts und der thermischen Behaglichkeit*. Band 41, De Monfort University Leicester, HFT Stuttgart, 1998.

[18] S. Freudiger, J. Hegewald und M. Krafczyk. A parallelisation concept for a multi-physics lattice boltzmann prototype based on hierarchical grids. *Progress in Computational Fluid Dynamics, An International Journal*, 8:168–178, 2008.

[19] A. Grama, A. Gupta, G. Karypis und V. Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2. Auflage, 2003.

[20] R. W. Hamming. *Numerical Methods For Scientists and Engineers*. McGraw-Hill, 1962.

[21] D. Hänel. *Molekulare Gasdynamik*. Springer Verlag, 2004.

[22] G. Hausladen, M. de Saldanha und P. Liedl. *ClimaSkin, Konzepte für Gebäudehüllen, die mit weniger Energie mehr leisten*. Verlag Callwey, München, 2006.

[23] G. Hausladen, M. de Saldanha, W. Nowak und P. Liedl. *Einführung in die Bauklimatik*. Ernst und Sohn, Berlin, 2003.

[24] R. Heiland und M. P. Baker. *Coprocessing: Experience with CUMULVS and pV3*, 1999.

[25] C. Hirsch. *Numerical Computation of Internal and External Flows*. Butterworth–Heinemann, New York, 2007.

[26] HLRS. *COVISE*, 2010. http://www.hlrs.de/organization/av/vis/covise/.

[27] C. Johnson und S. Parker. A computational steering model applied to problems in medicine. In: *Supercomputing '94*, pages 540–549. IEEE Press, 1994.

[28] C. Johnson, S. Parker, C. Hansen, G. Kindlmann und Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, 1999.

[29] S. Kühner. *Virtual Reality - basierte Analyse und interaktive Steuerung von Strömungssimulationen im Bauingenieurwesen*. Dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, 2003.

[30] F. Klimetzek. *Virtual Intuitive Simulation Testbed VISiT*. Technical report, Daimler Chrysler AG, Research and Development, 2001.

[31] J. A. Kohl, T. Wilde und D. E. Bernholdt. Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance. *Int. J. High Perform. Comput. Appl.*, 20(2):255–285, 2006.

[32] M. Kollinger. Definition strömungsmechanischer Randbedingungen für interaktive CFD Simulationen. Diplomarbeit, Lehrstuhl für Bauinformatik, TU München, 2007.

[33] M. Krafczyk. *Gitter-Boltzmann Methoden: Von der Theorie zur Anwendung.* Professorial dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, 2001.

[34] P. Lallemand und L.-S. Luo. Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions. *Physical Review E*, 68(036706), 2003.

[35] M. Landahl und E. Mollo-Christensen. *Turbulence and Random Processes in Fluid Mechanics.* Cambridge University Press, 1992.

[36] B.E. Launder und D.B. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3:269–289, 1974.

[37] K. Y. Lee, M. Price und C. Armstrong. Visicade - a virtual simulation environment for seamless integration of cad/cae into vr. *IEE Conference Publications*, 2004(CP506):437–442, 2004.

[38] PACX-MPI library, 2006. http://www.hlrs.de/organization/pds/projects/pacx-mpi.

[39] LRZ. *How the remote graphics reach your client (Sun Shared Visualization Software)*, 2011. http://www.lrz.de/services/compute/visualisation/visualisation_8.

[40] B.H. McCormick, T.A. DeFanti und M.D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.

[41] J. D. Mulder und J. J. van Wijk. 3d computational steering with parametrized geometric objects. In: *Proc. of the IEEE Conf. on Visualization*, 1995.

[42] R.-P. Mundani. *Lecture notes in Parallel Computing.* Department for Computation in Engineering, Technische Universität München, WS 2009/10.

[43] H. Oertel jr. (Hrsg.). *Prandtl – Führer durch die Strömungslehre.* Vieweg & Sohn, Braunschweig/Wiesbaden, 2002.

[44] Department of Computer Science und UMN Engineering. *METIS — Family of Multilevel Partitioning Algorithms*, 2010. http://glaros.dtc.umn.edu/gkhome/views/metis.

[45] OpenMP, 2010. http://www.openmp.org/.

[46] Suhas V. Patankar. *Numerical heat transfer and fluid flow.* Hemisphere Pub. Corp. ; McGraw-Hill, Washington : New York, 1980.

[47] S.M. Pickles, R. Haines, R.L. Pinning und A.R. Porter. A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1843–1853, 2005.

[48] L. Renambot, H. E. Bal, D. Germans und H. J. W. Spoelder. Cavestudy: An infrastructure for computational steering and measuring in virtual reality environments. *Cluster Computing*, 4(1):79–87, 2001.

[49] P. Sagaut. *Large Eddy simulation for incompressible flows.* Springer Verlag, 2. Auflage, 2001.

[50] N. Satofuka und T. Nishioka. Parallelization of lattice boltzmann method for incompressible flow computations. *Computational Mechanics*, 23:164–171, 1999.

[51] H. A. Schwarz. *Gesammelte mathematische Abhandlungen.* Springer Verlag, Berlin, 1890.

[52] A. Shadavakhsh, L. Möllenhoff und C. Büyükbayram. Visualization interface for ansys cfx. Software lab, Lehrstuhl für Bauinformatik, TU München, 2007.

[53] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond.* Clarendon Press, Oxford, 2001.

[54] THESEUS-FE, 2008. Theory manual, release 2.0, http://www.theseus-fe.de.

[55] G. Vahala, P. Pavlo, L. Vahala und N.S. Martys. Thermal lattice-Boltzmann models (TLBM) for compressible flows. *Int. J. Modern Physics C*, 9(8):1247–1261, 1998.

[56] R. van Liere und J. J. Mulder, J. D. und van Wijk. Computational steering. In: *High-Performance Computing and Networking*, pages 696–702. Springer-Verlag, 1996.

[57] C. van Treeck. *Lecture notes in Parallel Computing.* Department for Computation in Engineering, Technische Universität München, 2002–2006.

[58] C. van Treeck. *Gebäudemodell-basierte Simulation von Raumluftströmungen.* Dissertation, Technische Universität München, 2004.

[59] C. van Treeck. *Introduction to Building Performance Simulation.* Habilitation thesis, Technische Universität München, 2009.

[60] C. van Treeck, J. Frisch, M. Egger und E. Rank. Model-adaptive analysis of indoor thermal comfort. In: *Building Simulation 2009*, Glasgow, Scotland, 2009.

[61] C. van Treeck, J. Frisch, M. Pfaffinger, E. Rank, S. Paulke, I. Schweinfurth, R. Schwab, R. Hellwig und A. Holm. Integrated thermal comfort analysis using a parametric manikin model for interactive real-time simulation. *J Building Performance Simulation*, 2(4):233–250, 2009.

[62] C. van Treeck, M. Krafczyk und M. Schulz. Lattice-Boltzmann Verfahren im Bauwesen – Tutorial eines grafisch-interkativen Strömungssimulators in 2D. In: R. Romberg und M. Schulz, editors, *Forum Bauinformatik, Junge Wissenschaftler forschen, München '01*, Fortschrittberichte VDI, Reihe 4, Nr.169, pages 249–259, 2001.

[63] C. van Treeck, M. Pfaffinger, P. Wenisch, J. Frisch, Z. Yue, M. Egger und E. Rank. Towards computational steering of thermal comfort assessment. In: *IndoorAir2008*, Copenhagen, Denmark, 2008.

[64] C. van Treeck und E. Rank. ComfSim - Interaktive Strömungssimulation und lokale Komfortanalyse in Innenräumen. In: Bayerische Forschungsstiftung, editor, *Jahresbericht 2005*, page 66, München, Germany, 2006.

[65] C. van Treeck, E. Rank, M. Krafczyk, J. Tölke und B. Nachtwey. Extension of a hybrid thermal lbe scheme for Large-Eddy simulations of turbulent convective flows. *Computers and Fluids*, 35:863–871, 2006.

[66] C. van Treeck, P. Wenisch, A. Borrmann, M. Pfaffinger, O. Wenisch und E. Rank. ComfSim - Interaktive Simulation des thermischen Komforts in Innenräumen auf Höchstleistungsrechnern. *Bauphysik*, 29(1):2–7, 2007. DOI: 10.1002/bapi.200710000.

[67] H. Wengle. *Vorlesung und Skriptum Numerische Berechnung Turbulenter Strömungen.* Lehrstuhl für Fluidmechanik, Technische Universität München, 2001.

[68] P. Wenisch. *Computational Steering of CFD Simulations on Teraflop-Supercomputers.* Dissertation, Department for Computation in Engineering, Technische Universität München, 2007.

[69] P. Wenisch. *Interactive Fluid Simulations: Computational Steering on Supercomputers.* Technical report 2006, In: Science and Supercomputing in Europe, 2007.

[70] P. Wenisch, A. Borrmann, E. Rank, C. van Treeck und O. Wenisch. Collaborative and interactive CFD simulation using high performance computers. In: *18th Symposium AG Simulation (ASIM) and EuroSim Frontiers in Simulation*, Erlangen, 2005. SCS PublishingHouse e.V. Erlangen.

[71] P. Wenisch, C. van Treeck, A. Borrmann, E. Rank und O. Wenisch. Computational steering on distributed systems: Indoor comfort simulations as a case study of interactive cfd on supercomputers. *Int. Journal of Parallel, Emergent and Distributed Systems*, 22, 2007.

[72] Petra Wenisch. *Computational Steering of CFD Simulations on Terraflop-Supercomputers.* Dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, 2008.

[73] Wikipedia, 26.05.2010. Floating Point Operations Per Second, http://de.wikipedia.org/wiki/FLOPS.

[74] A. Wilde. *Anwendung des Lattice–Boltzmann–Verfahrens zur Berechnung strömungsakustischer Probleme.* Dissertation, Technische Universität Dresden, 2006.

[75] D.A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models.* Springer Verlag, 2000.

[76] D. Yang. *C++ and object-oriented numeric computing for scientists and engineers.* Springer, 2001.

[77] Z. Yue. *A computational human model for thermal comfort simulation.* Master thesis, Computation in Engineering, TU München, 2008.

[78] Z. Yue. *The realization of highly complex dummy animations in the Computational Steering Environment.* Report, Computation in Engineering, TU München, 2008.