Lehrstuhl für Bauinformatik
Fakultät für Bauingenieur- und Vermessungswesen
Technische Universität München

Computational Steering of CFD Simulations on Teraflop-Supercomputers

Petra Wenisch

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. Michael Manhart
Prüfer der Dissertation:
1. Univ.-Prof. Dr. rer. nat. Ernst Rank
2. Univ.-Prof. Dr. rer. nat. Ulrich Rüde,
Friedrich-Alexander Universität Erlangen-Nürnberg

Die Dissertation wurde am 7. November 2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Bauingenieur- und Vermessungswesen am 7. Februar 2008 angenommen.

für Heike und Hella

Abstract

Computational methods have by now become established techniques of everyday workflow in civil engineering, especially in the field of structural mechanics. Another field of application with increasing importance is computational fluid dynamics (CFD), where it is most prominently used for simulations in the context of hydraulics or for investigations with respect to fluid-structure interactions such as wind loads on bridges, skyscrapers.

In building construction practice the use of simulations for the analysis of indoor air-flow is much less common, because especially during the planning phase of unique constructions like non-standard buildings, the limited time available to the engineer often does not allow detailed simulation series. Hence, these obviously helpful early simulation studies cannot be carried out at all and rules of thumb are still used instead. Should it turn out during the building phase that a redesign is inevitable, the costs are considerably higher than those for a redesign during the planning phase. Taken together, the need for simulations and the lack of time during design phase led to a desire for an alternative tool giving the engineer the possibility of performing case studies of the qualitative flow behavior through short-cycle simulation runs. By providing means of monitoring and interactive steering the running simulation, the engineer can watch the developing flow, adjust parameters according to his needs, and receive instant feedback to his interactions. This kind of steerable simulation and on-the-fly visualization of simulation results represent the defining features of *computational steering*. With the general availability of *affordable* high-performance computing systems and perpetually increasing CPU power, computational steering applications have become possible even in so computation- and resource-intensive fields as computational fluid dynamics in these days.

This thesis focuses on the design of a computational steering framework utilizing supercomputers for the numerical fluid dynamics simulations and high-end Virtual-Reality visualization workstations for online monitoring and steering of the simulation. The computational steering tool *iFluids*, which has been developed based on this framework, opens a whole range of new applications in different fields — not particularly restricted to civil engineering. *iFluids* represents an interactive tool as a means of exploratory short-term investigation for checking ventilation designs in their basic functionality and qualitative behavior — as such meeting at least part of the engineer's requirements during the design phase. In particular, the engineer can explore and watch develop a flow configuration during the running simulation and can interactively adjust global flow parameters, define or modify boundary conditions and even change the geometrical setup without the need to re-preprocess and rerun as with most other simulations. The adaptation of the flow as well as transient effects until a steady state is reached can be watched on a steering and visualization terminal on the fly. In this way, it is possible to perform virtual experiments to quickly gain an intuitive understanding of a given ventilation problem.

iFluids is based on the Lattice-Boltzmann method, which has only been applied for some twenty years in fluid dynamics research. It offers several signif-

icant advantages over other classical methods of tackling fluid dynamics problems. The present thesis describes the specializations and optimizations of the implementation of this method which provide the key to enabling the user to easily change boundary conditions, flow parameters, and the geometric layout of the simulated scene *during* the on-going computation.

However, the simulations often need to be run with a coarser discretization on current mid-level hardware such as workgroup clusters and, accordingly, are not always able to resolve all details of the fluid behavior. Additionally, the computational model must still be kept simple with respect to boundary conditions or turbulence models. Nevertheless, it enables small case-study simulations for feasibility analyses and for testing the behavior in question. The requirements arising when developing a full-grown computational steering application and how these can be addressed represent an important aspect, which is specifically concentrated on in the text. In particular, details are given on how computational steering software can be developed on today's high-performance computers and high-end visualization facilities. Also, the currently achievable performance is benchmarked and limitations with regard to the current hardware technology are pointed out.

To demonstrate the applicability of the computational steering framework to the study of indoor ventilation systems, *iFluids* is used for analyzing a real-case operating room as an example. Results of the interactive simulation tool are compared with more detailed simulations to show what kind of statements can be made through using this tool. Finally, the applicability of the computational steering framework in other fields will be touched.

Zusammenfassung

Rechnergestützte Simulationen haben sich mittlerweile im Bauwesen und insbesondere im Bereich der Strukturmechanik als feste Hilfsmittel im Arbeitsalltag etabliert. Ein weiteres Anwendungsgebiet, in dem die Bedeutung numerischer Simulationen im Bauwesen zunimmt, ist die Fluiddynamik. Hier liegt der Schwerpunkt jedoch mehr auf Fragestellungen der Hydraulik und Gewässerkunde oder Fluid-Struktur-Interaktionen, wie sie bei umströmten exponierten Bauwerken wie Wolkenkratzern oder Brücken auftreten.

In der Praxis des konstruktiven Ingenieurwesens ist die Simulation von Innenraumströmungen noch eher unüblich, da die relativ kurze Entwurfsphase von größeren Bauwerken meistens nicht den zeitlichen Spielraum für detaillierte Simulationsreihen gewährt. Daher werden solche Berechnungen nur sehr selten durchgeführt und meist lediglich Erfahrungswerte für die Planung herangezogen. Stellt sich während der Bauausführung jedoch heraus, dass eine Überarbeitung des Konzeptes unumgänglich ist, fallen die damit verbundenen Kosten deutlich höher aus, als es während der Entwurfsphase der Fall gewesen wäre.

Der trotz des engen zeitlichen Rahmens grundsätzlich bestehende Bedarf an Simulationen erzeugt auf Ingenieursseite den Wunsch nach einem anderen als den bislang zugänglichen Simulationswerkzeugen: Ein derartiges Werkzeug muss

einem Ingenieur die Möglichkeit bieten, eine Reihe von Fallstudien innerhalb kurzer Zeit durchzuführen, um ein qualitatives Strömungsverhalten bestimmen zu können. Durch die gleichzeitige Visualisierung aktueller Strömungsdaten und interaktiver Steuerung während der Simulation soll der Anwender die zeitliche Entwicklung einer Strömung beobachten und durch die Anpassung von Parametern in die laufende Simulation eingreifen können. Die Auswirkungen seiner Veränderungen müssen anschließend in der zur Berechnung parallel laufenden Visualisierung unmittelbar sichtbar werden. Diese Verschmelzung von steuerbarer Simulation und gleichzeitiger Visualisierung nennt man *Computational Steering*. Aufgrund der heute verfügbaren und nach wie vor rasant wachsenden Rechenleistung bei Hoch- und Höchstleistungsrechnern ist es mittlerweile möglich geworden, *Computational Steering* sogar in so rechenintensiven Gebieten wie der Fluiddynamik zu verwirklichen.

Die vorliegende Arbeit beschreibt den allgemeinen Aufbau einer *Computational Steering*-Applikation und zeigt, wie sich eine derartige Anwendung durch die Verwendung von Höchstleistungsrechnern für die numerische Fluidsimulation und Virtual-Reality Visualisierungsanlagen zur Darstellung der Ergebnisse sowie zur Steuerung der Berechnung realisieren lässt. Anhand der Beispielanwendung *iFluids* wird ein interaktives Werkzeug zur Untersuchung von Innenraumluftströmung erläutert, das in kurzer Zeit erlaubt, qualitative Aussagen über ein Belüftungssystem zu treffen und seine grundsätzliche Funktion zu prüfen. Somit kann *iFluids* dem Ingenieur als Hilfsmittel während der Planungs- und Entwurfsphase dienen, das den wesentlichen Teil der oben genannten Ansprüche abdeckt. So kann man während der laufenden Berechnung die Strömungsentwicklung beobachten und interaktiv globale Strömungsparameter, Randbedingungen oder sogar den geometrischen Aufbau der Problemstellung verändern. Im Gegensatz zu den meisten heute verfügbaren Simulationswerkzeugen ist nach derartigen Interaktionen kein Neustart der Berechung mehr erforderlich. Die Anpassung der Strömung ebenso wie transiente Effekte bis zu einem möglichen Gleichgewichtszustand können an der Steuerungs- und Visualisierungsanlage unmittelbar betrachtet werden. Auf diese Weise können "virtuelle Experimente" mit geringem Zeitaufwand durchgeführt werden und dadurch ein intuitiver Eindruck des Strömungsverhaltens bei einer gegebenen Belüftungsfragestellung gewonnen werden.

iFluids berechnet die Strömungssimulation mittels der Lattice-Boltzmann Methode, die ein noch relativ junges Verfahren auf diesem Gebiet darstellt. Ihr Einsatz bietet im Vergleich zu anderen numerischen Methoden einige Vorteile für die *Computational Steering*-Anwendung. Die vorliegende Arbeit geht daher auf die Spezialisierung und Optimierung dieser Methode ein, die die interaktive Veränderung von Randbedingungen, Strömungsparametern und des geometrischen Aufbaus während der laufenden Simulation möglich machen.

Insbesondere auf weniger leistungsfähigeren Rechnern wie z.B. Arbeitsgruppen-Clustern, müssen zur Beschleunigung der Simulationen die geometrischen Objekte meist weniger fein diskretisiert bleiben. So können häufig nicht alle Strömungsphänomene voll aufgelöst werden. Auch hinsichtlich der Randbedin-

gungen oder für die Turbulenzmodellierung müssen oft vereinfachte Modelle verwendet werden. Dennoch können mit dieser Art von Simulation Fallstudien durchgeführt werden, die die grundsätzliche Verwendbarkeit und Funktionsweise von geplanten Belüftungssystemen prüfen. Darüber hinaus geht die Arbeit auch auf die Anforderungen ein, die während der Entwicklung der *Computational Steering*-Anwendung deutlich werden, und zeigt, wie man diesen gerecht werden kann. Insbesondere wird beschrieben, wie *Computational Steering*-Software auf heutigen Höchstleistungsrechnern im Zusammenspiel mit modernen Visualisierungsanlagen entwickelt werden kann. Schließlich wird auch die derzeit mit *iFluids* erreichbare Berechnungsleistung untersucht und die daraus ersichtliche Limitierung bezüglich heutiger Hardware herausgearbeitet.

Um die Anwendbarkeit dieser *Computational Steering*-Lösung im Bereich der Innenraum-Strömungssimulation zu beurteilen, wird mithilfe von *iFluids* das Beispiel eines realen Operationssaals simuliert. Die Ergebnisse aus der interaktiven Simulation werden hierbei mit denen einer herkömmlichen Simulation verglichen, um herauszustellen, welche Art von Aussagen man durch Einsatz von *Computational Steering* treffen kann. Um die Flexibilität der entstandenen Software zu demonstrieren, wird abschließend gezeigt, dass *iFluids* nicht nur auf Probleme aus dem Bauwesen beschränkt ist, sondern ein breites Spektrum verschiedenster Anwendungen abdecken kann.

Contents

Chapter 1

Computational Steering

This chapter shall serve as a short introduction to the topic of computational steering. First, the term 'computational steering' is defined and an overview of the current state of the art is presented by providing sketches of a selection of related work. Then, possible fields of application are given by way of an example followed by a detailed discussion on the application range in civil engineering.

1.1 Introduction

Traditionally, computational engineering studies in high-performance computing are carried out in a sequence of steps. At the beginning one has to supply the geometric modeling (i.e. CAD, mesh or grid generation) and the definition of simulation specific start and boundary conditions, which has become practical on standard desktop machines. This is referred to as the pre-processing step which is followed by the actual simulation and a concluding post-processing step to extract and evaluate particular results (see Figure 1.1).

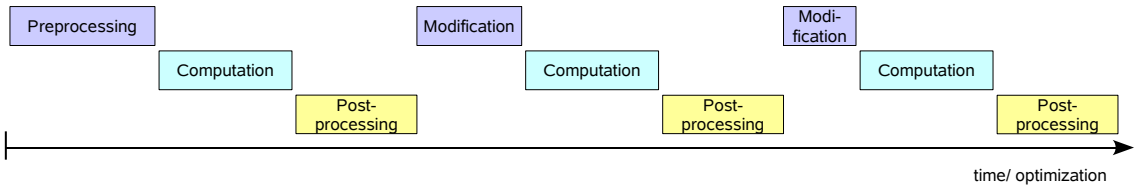

Figure 1.1: *Iteration of work steps for optimization problems in engineering studies: In the traditional approach, the steps of pre-processing, simulation and evaluation of the simulation results are traversed sequentially and iterated in a drawn-out loop. Assuming the best-case scenario, potential waiting times for required resources have been ignored in this scheme. Furthermore it is assumed that a modification of the simulated model does not require the whole pre-processing procedure and is therefore cut short in each optimization cycle.*

Regarding the computational effort, engineering problems still belong to the most challenging numerical simulations with substantial resource requirements

that arise as more and more physical details are taken into consideration. Accordingly, it is often advantageous to perform the computation on a supercomputer or cluster. On this type of machine a simulation most commonly has to be submitted to a queuing system as a batch job which 'has to wait' until the requested resources become available before it can be executed. At the end of the simulation, the results are usually stored on a filesystem and, occasionally, have to be transferred to an adequate post-processing front-end, which may be located on a different, more suitable system. There, the results obtained are finally evaluated in the post-processing step by means of appropriate analysing and visualization techniques. When studying several test cases of a simulation scenario this often tedious and lengthy chain of processes is rather inconvenient for the engineer without the possibility of immediate interaction with his 'experiments'.

According to Johnson et al. (1999), the first published statements indicating the desire for computational steering, which integrates these single steps of a pipeline into one single process cycle, appeared in the late 1980s. McCormick et al. (1987) reflected that scientists want to be able to *interact* with simulations close to real-time. Correspondingly, Johnson and Parker (1994) have defined *Computational Steering* as the 'capacity to control all aspects of the computational science and engineering pipeline'. This comprises the steps of pre-processing, computation, and post-processing as mentioned above (cf. Figure 1.2).

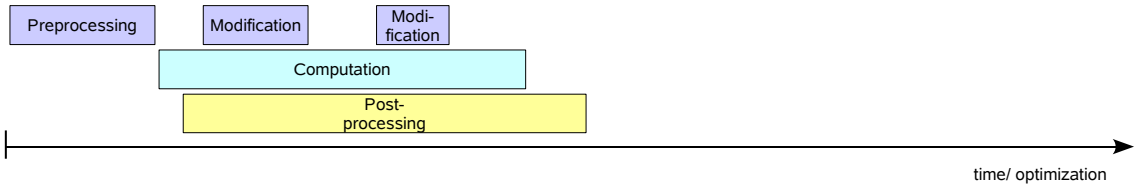

Figure 1.2: *Closing the loop: Compared with the traditional form of sequential steps, as shown in Figure 1.1, the optimization of three iteration steps is speeded up considerably by using a computational steering application. For one thing, preprocessing, computation and postprocessing can be done in parallel. For another, the visualization and computation times are reduced since these processes usually benefit from previous results and do not necessarily need to be restarted from the very beginning.*

Since the articulation of these early visions, the power of computers has increased severalfold and was paralleled by the development of interactive simulation applications. However, most of these focused on the post-processing and visualization step and did not allow direct interaction with the computation (van Liere et al., 1996). As a matter of fact, computational steering — even today — is often mis-interpreted as 'online monitoring' with only basic functionalities like stopping, pausing and resuming a simulation. Despite offering the first step towards real computational steering, these basic features clearly fall short of representing a fully developed interaction with the simulation during runtime. Computational steering in its explicit meaning, however, enables the scientists to directly change some or all of the parameters of the simulation process during its

execution and the availability of a front-end to analyze the effects of these interactions immediately Mulder et al. (1999).

1.2 Related Work

The following section outlines some examples representing the state-of-the-art in the field of computational steering. These projects can be classified into libraries, problem solving environments and application frameworks.

Libraries

The **gViz** library (Brodlie et al., 2004) allows users to visualize data in a post-processing step or on-the-fly during a simulation. Both scenarios support the use of grid computing and collaborative working facilities. In addition to the visualization capabilities parameters can be sent to the underlying simulation in case the changing of parameters during runtime is supported by the simulation kernel.

In the **RealityGrid** (Brooke et al., 2003; Pickles et al., 2004) project an application is typically structured into the client, the simulation and the visualization module communicating by means of a steering library. The library is designed to simplify the changes required to make an existing code 'computationally steerable'. A key requirement for extending the application in this way is the insertion of check- and break-points where modified parameters are fetched and the simulation has to be restarted, respectively. With RealityGrid simulations are either monitored in a read-only mode or steered by modifying parameters or previously registered data sets. The user may pause, resume, detach and stop the simulation or run it from a checkpoint. RealityGrid separates visualization and steering, which makes it possible to investigate current results with arbitrary hardware such as high-end visualization workstations, laptops or PDAs.

Another library has been developed within the **CAVEstudy** project, cf. Renambot et al. (2001). In contrast to the libraries described above the original simulation code does not need to be adapted at all. Instead, information about a simulation's input and operations is placed in a file describing the way commands are issued. The CAVEstudy library enables visualization and remote steering within Virtual Reality (VR) environments like a cave.

Problem Solving Environments

As opposed to the computational steering libraries, **SCIrun** is a so-called problem solving environment (PSE) (Parker et al., 1999). A problem solving environment is a computational system that provides a complete and convenient set of high-level tools for solving problems from a specific domain (Abrams et al., 2007). In such PSEs applications are often composed through a visual programming interface similar to the widely-known AVS front-end (Advanced Visualization Systems Inc (AVS Inc., 2007)), for example. The user has to set up a network of

modules and interacts via their corresponding graphical user interface. Several parameters can be changed during the simulation without the need to stop it. The affected module is re-executed and sends updated output to all connected modules, which react accordingly. Other changes with a deeper impact on the simulation require an automatic cancellation and restart of the simulation.

COVISE (Covise, 2007) is a collaborative visualization and simulation environment. COVISE rendering modules support a wide range of Virtual Reality environments for analyzing datasets intuitively. This distributed environment integrates simulation, post-processing and visualization functionalities, the different processing steps being represented by modules. In Wössner et al. (2005) these modules are extended to support the setup of an interactive CFD simulation. The main focus hereby is to investigate the feasibility of using a tangible interface as an intuitive input device. In a Virtual Reality environment obstacles attached with special markers are placed within the Virtual Reality setup. A set of cameras tracks the position of these markers to determine the position and orientation of the associated obstacles. If the representatives in the real world are moved, the scene has to be remeshed for the following simulation cycle. This has to be triggered by the user by pressing a button.

Application Frameworks

In Georgii and Westermann (2005) an approach to realize interactive simulation on a consumer PC is presented. Here, external forces can be applied to deformable bodies during runtime. The simulation is based on a multi-grid solver running on a PC's CPU while the render engine is run in parallel on the GPU[1] on its graphics card. Within a pre-processing step a certain scenario is set up once and, during runtime, the simulation engine consecutively displaces the underlying finite element mesh of the deformable body according to the user's interaction.

VFReal (Kühner, 2003) was a precursor application to this thesis' approach making the first steps towards an interactive CFD simulation for indoor comfort studies. It is designed as a monolithic application running a Lattice-Boltzmann solver with on-the-fly visualization. The steering possibilities comprise the placement of several basic geometry primitives as fluid obstacles into the scene. However, the underlying simulation grid for these objects is required in a pre-generated form at simulation time.

The insights that have been made during the process of porting the VFReal application onto a supercomputer and connecting it to a Virtual Reality environment played an active part in developing the new approach presented in this thesis. In the present form the user can now interact with each of the three processing steps even when run distributed on a visualization and simulation machine. Boundary conditions can be set or adjusted interactively, as is common in offline preprocessing front-ends. In addition, the geometry of the simulated

[1]The graphics processing unit (GPU) is the dedicated graphics rendering device on modern graphics boards which, in special applications, is also used for general purpose computations.

scene can be modified during runtime. In contrast to Wössner et al. (2005) arbitrary geometries can be inserted throughout the simulation run by loading from the filesystem without pre-meshing or any other kind of pre-processing. Regarding the simulation, the numerical model and method can be adjusted during execution time and, of course, interaction with the visualization of current results for post-processing issues is supported. The event-based framework works fully automatically, i.e the interactions are incorporated without any extra actions like starting a remeshing process, restarting the simulation or updating visualization data. Further details of the application and its features are described in Chapter 2. In Borrmann (2007) this application framework was extended for a multiple client version to support collaborative engineering.

1.3 Fields of Application

Computational steering has a wide, and still increasing range of potential application areas. To show the high versatility of this method a few examples of different fields of interest benefiting from computational steering are presented below.

Non-Invasive Vascular Reconstruction

A relatively new field of application for computational steering can be found in medical engineering. The numerical simulation of blood flowing through blood vessels is a popular matter of interest in this respect. The non-invasive vascular reconstruction as discussed in Sloot et al. (2004) may serve as a typical example from this field.

Arteries and veins are increasingly affected by a growing number of vascular diseases. Two categories of vascular dysfunction have to be differentiated: aneurisms and stenoses. An aneurysmal disease refers to balloon-like swellings in the artery, whereas stenosis represents a narrowing or blockage of the artery. A vascular reconstruction intervention aims at treating these abnormal vessels through surgery. In the case of aneurysms this means adding shunts, bypasses, and placing stents or, for stenosis, applying thrombolysis techniques such as balloon angioplasty, bypasses, etc. It is obvious that finding the optimal treatment is far from trivial and a simulation tool to support the verification of a planned operation may prove to be a valuable supplement to classical approaches. A group of researchers under Prof. Sloot at the University of Amsterdam has developed a grid-based problem solving environment to test several treatments in this respect. However, they still have to process the whole sequential pipeline of pre-processing, computation and post-processing. Nevertheless, the application achieves almost real-time simulations.

Simulation of Manufacturing Systems

Recent developments in the simulation of manufacturing systems also offers possibilities for interactive simulation within a computational steering framework.

For the optimization of manufacturing systems a traditional simulation cycle consists of preparing input variables, selecting simulation parameters and running the simulation, which is followed by reviewing the results after the computation. Because of the complex 'what-if-scenario' analysis, several simulation cycles are needed before any initial results of sufficient interest or value are obtained. Therefore, Sudhir and Kesavadas (2000) propose to utilize computational steering concepts within an interactive virtual environment. In this way, on-the-fly visualization of results delivered by a manufacturing simulation could be used to allow for instant feedback from the system after modification within the virtual environment. A simple conceptual system has already been implemented to study this approach with promising initial results.

Helios - Computer Aided Lighting Technologies

To improve and speed up the developmental cycle, computer-aided technologies (CAx) are extensively used in automotive design and construction and often computational steering can be helpful in various ways. One interesting application is, e.g., the design of automotive lighting in Computer Aided Lighting (CAL) systems, which can be integrated straightforwardly into the development process. CAL may be seen as a virtual lighting laboratory where the work is again divided into the three steps of pre-processing, simulation of the construction and post-processing. All of these issues are addressed by the CAL application *Helios* developed by Hella KG (Hella KG, 2007) for testing automotive lights in a simulated environment (Biermann and Kalze, 1996). The above-mentioned development cycle is repeated until the specific lighting requirements are met so that a prototype can be built. Integrating the virtual lighting laboratory into a computational steering framework could probably improve and speed up the computer-aided testing process and, moreover, would allow for an intuitive modification of lighting designs.

Interactive Physics-Based Simulations

A topic that closely related to computational steering is the more and more popular interactive physics-based simulation of real-time scenarios in computer games or other virtual 'worlds'. This type of simulation does not quite satisfy the definition of computational steering, since in these worlds the goal is not the steering of the simulation itself. In fact, the actual intention is to simulate a real or fictitious world in a realistic way, i.e., to respect physical laws and the behavior of the scenario depending on all influences, also including the user in this world. The remarkably sophisticated genre of computer games is the most prominent representative of this category.

However, there are also academic examples such as the simulation of rivers, as described in Kipfer and Westermann (2006). Here, water can spring from several sources, flow over a height field to form rivers or to cluster into lakes, perpetually influenced by gravitation, wind and, of course, terrain obstacles. The simulation is based on smoothed-particle hydrodynamics with surface extraction running on the GPU.

Related examples can be found in Thürey et al. (2005) and Tölke (2006), where fluids with a free surface are animated based on the Lattice-Boltzmann method. Thürey et al. (2005) mainly presents drops and fluid streams falling into a pool of fluid. The authors state that due to the use of the Lattice-Boltzmann method it is comparatively easy to set boundary conditions. Therefore, it is possible to interactively place drops of fluid above a bowl-shaped obstacle, into which these drops splash in complex shapes driven by gravity. Similarly, albeit not steerable, Tölke (2006) simulate the free surface flow over a barrage with its hydraulic jump and a surf wave.

Another spectacular example of a distributed and parallel graphics application utilizing Virtual Reality is the **FlowVR** application framework (Allard and Raffin, 2006). It is capable of simulating rigid bodies, mass-spring objects and an approximation of fluid behavior by an inviscid, multi-body simulation (Eulerian fluid) using a 32 processor cluster for computation and a 22 processor visualization cluster plus 5 FireWire cameras for tracking in VR. The position of the freely moving user is tracked to consider his influence due to collisions with the virtual bodies in the simulation.

1.4 Computational Steering in Civil Engineering

One aspect of this thesis is the evaluation of the applicability of computational steering for civil engineering problems. After presenting a series of general examples of computational steering, the following section concentrates on the field of civil engineering, which stands out from most other engineering disciplines because of the lack of designing prototypes.

Industry sectors with large-scale production such as the automotive industry usually invest considerable amounts of time and money in the design phase of the prototype for a new product. In this planning and testing phase, extensive simulations are conducted, frequently leading to repeated re-designing of the concept. Nevertheless, prototypes are also built and tested. After a fairly long period, serial production commences and the costs of planning and design are recuperated by high sales figures.

In contrast, the specialty of civil engineering is the construction of unique copies. Therefore, the design phase has to be much shorter and less extensive to be profitable. As shown in Figure 1.3 the possible influence of the design of the project and its cost is significantly higher during the planning phase than during the construction phase. The development pattern of the project costs is the exact opposite. Therefore, a good and coherent concept for the planning phase is required, since later changes will cause difficulties and result in high additional

costs (Seidenschwarz, 1997).

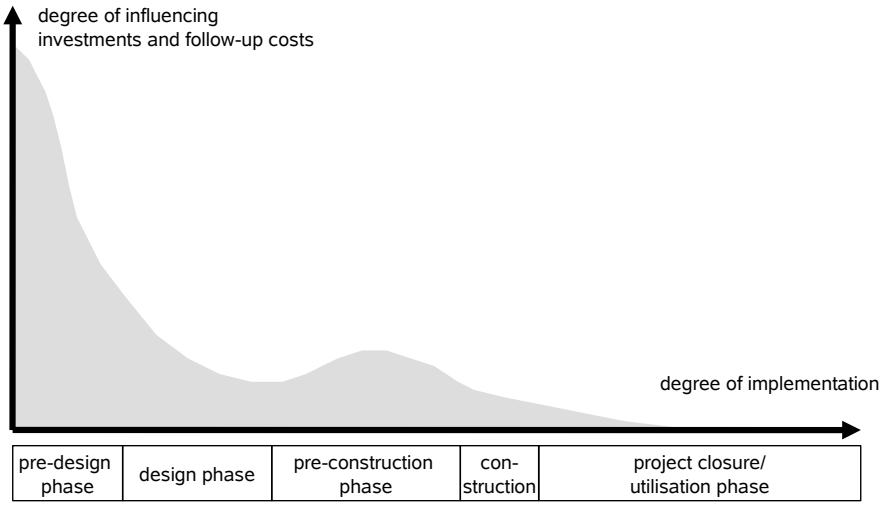

Figure 1.3: *Influences on a building project during the planning and construction phases: This graph shows the high degree of influence on a planned construction in the project preparation phase. The further the implementation of the project has proceeded, the less flexible is the design. Since changes at later phases result in higher additional costs, the importance of a well thought-out concept is evident (Seidenschwarz, 1997). (taken from Diederichs (1984))*

Due to the lack of time for detailed simulations during the planning phase, often only empirical rules are used for designing instead. Nevertheless, it is well known that short-comings still occur and the belated elimination of these is difficult and cost intensive. This leads to the desire for an interactive simulation tool for preliminary investigations, which makes it possible to run short simulation cycles to prove or find the basic concept, possibly followed by a few carefully selected simulation setups for more detailed investigations. This is a classical situation where computational steering can find its appliance.

Conceivable tasks in the field of civil engineering that might benefit from interactive simulations are, for example, the analysis of the spread of pollution due to natural winds, escape route simulation during various emergency scenarios, traffic simulation, flood simulation, or the broadening of fire and smoke in buildings.

The use case for this thesis is the simulation of fluid dynamics for indoor air flow, e.g. for ventilation systems in rooms. In this instance, the points of interest are whether the ventilation affects all parts of the room which need to be circulated, and whether the flow velocities are small enough to prevent uncomfortable air movements or even sickness of the occupants.

The computational steering framework developed in this thesis for the issues of indoor CFD simulation is based on the Lattice-Boltzmann method (Succi, 2001; Kafczyk, 2001). To shorten the computation cycle, the numerical model is simplified compared to current state-of-the-art implementations, which usually include

advanced methods such as spatial and temporal adaptivity, multiple scales, and boundary layer models. An additional, but also the most significant acceleration is achieved by using supercomputers. Therefore, special emphasis is placed on high-performance aspects of computational steering, especially with regard to the systems available during this thesis, namely the Hitachi SR8000-F1 pseudo-vector supercomputer and the SGI Altix 3700/4700 of the Leibiniz Computing Center (LRZ) in Munich.

The computational steering framework *iFluids* for indoor air-flow simulations is introduced in chapter 2. Chapters 3–7 gives a detailed introduction to the development of a distributed computational steering framework, i.e. the theory of the underlying Lattice-Boltzmann method, its parallelization and optimization for supercomputers, followed by a description of the visualization and steering module and the efficient coupling through a suitable communication concept. Subsequently, the applicability of an interactive simulation tool in civil engineering is investigated taking an operating theatre as the main example. Finally, the flexibility of the modular software concept is demonstrated by applying the framework to a completely different problem, namely the interactive blood flow simulation within an artery.

Chapter 2

iFluids - Interactive Fluid Simulations

This chapter serves as an introduction to the computational fluid dynamics application *iFluids*, which will only briefly be presented with regard to its characteristic architecture and features to provide an overview of the main objective in this thesis. The detailed description will be given in later chapters.

iFluids is an application which has been developed mainly as a tool for indoor fluid flow simulations but it is also easy to extend for simulation studies in other fields with a focus on geometric setup. What distinguishes it from other computational fluid dynamics applications is its layout as a computational steering framework for high-performance computers. Its user is able to visualize current, near real-time simulation results on-the-fly and to interact with the simulation while it is still running. Besides basic interaction options such as (re-)starting, stopping, and pausing the computation, the user can modify the geometry of the simulated scene as well as its boundary conditions in a standard desktop or a high-end virtual reality user interface (Fig. 2.1). It is also possible to adapt the computational kernel in terms of the numerical model or the best optimization available for a particular hardware platform.

The following section describes the functional concept of *iFluids*. First, the structural characteristics of the simulation kernel are given followed by a presentation of the steering and visualization terminal. Finally, the special requirements needed to achieve an *interactive* simulation tool providing the required rapid responses to user interactions are summarized.

2.1 Architectural Software Concept

To utilize top-level hardware such as supercomputers for simulation and virtual reality environments for visualization and steering in an efficient way, the *iFluids* framework has been designed as a set of independent modules that can be run (optimized) on many different platforms. An additional benefit of this architecture is the possibility of running *iFluids* distributed on several inhomogeneous hardware setups.

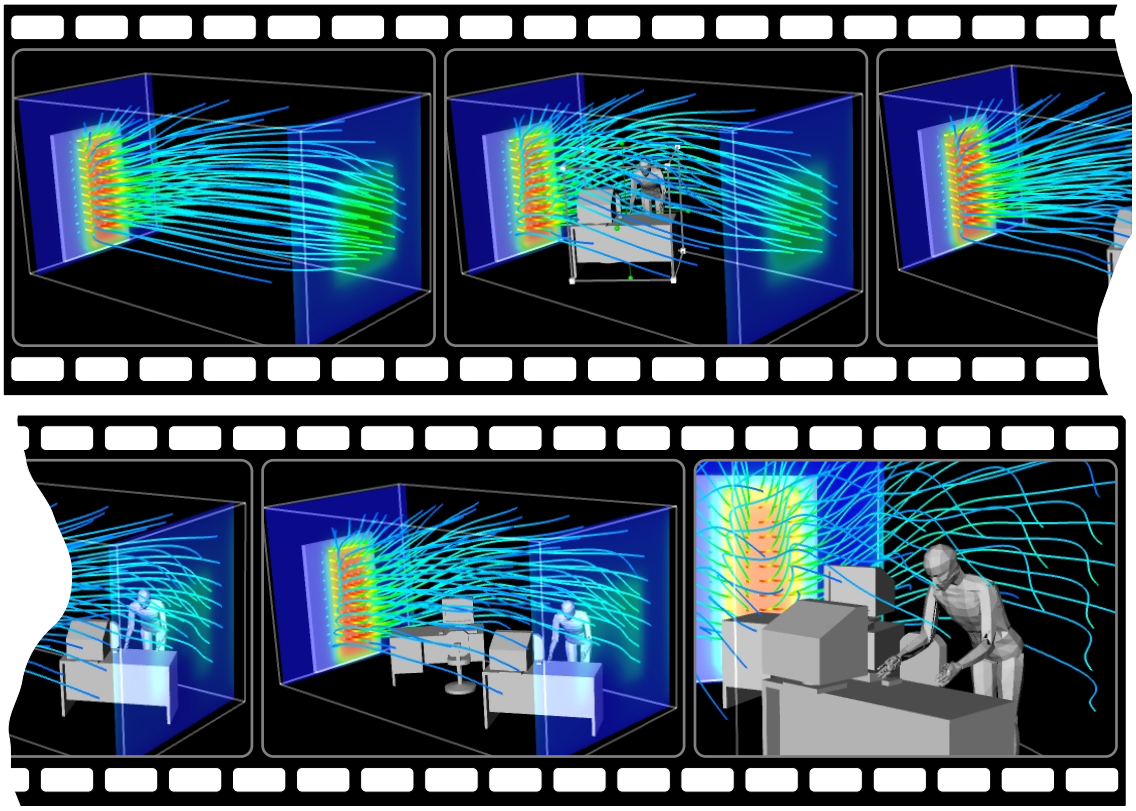

Figure 2.1: *This sequence of snapshots shows the simulated development of air flow in an office room over time. The first frame depicts the steady-state air flow through the room and its underlying boundary conditions, which have been defined beforehand through the application's VR user interface. The flow field is visualized by stream lines while the boundary conditions on the walls are color-coded from blue, representing resting air, to red for air moving with maximum speed. From the left side of the room air enters through an open door streaming directly to an open window in the opposite wall. In the second frame a desk has been added to the scene and one can clearly see how the flow adjusts to the new interior design after only a few seconds of simulation. This enables the user to investigate a series of scenarios interactively and observe their impact on the resulting flow configuration straight away. The investigating engineer can not only change the external view of the scene but also employ different modes of navigation such as walking and flying through the virtual room in his interactive study.*

The schematic diagram in Figure 2.2 shows the module concept of *iFluids*: the visualization and steering front-end is run on suitable selected graphics hardware ranging from standard laptops to high-end visualization environments, receiving results from the simulation kernel that computes the current flow configuration in the background. Optimally, the simulation kernel would not be executed on the visualization machine but on an additional PC, cluster or supercomputer, depending on the model size, level of detail, and the available hardware. The graphics setups used during this thesis were the holobench at the Leibniz Computing Center (LRZ), the Powerwall at the Lehrstuhl für Bauinformatik, and a notebook with good 3D graphics capabilities. The simulation was either run on a single Linux PC, the Bauinformatik Linux Cluster, or the Hitachi SR8000 supercomputer at LRZ, depending on performance requirements and hardware availability. In addition to data exploration, the user can — within the same front-end — intuitively adapt the geometrical setup of the simulated model to precisely the setup he is interested in. The steering terminal forwards the relevant interaction events to the simulation kernel. The kernel immediately incorporates the changes and sends back the adapted flow data set to the user front-end showing an initial trend of the new configuration in the scene almost without delay. As mentioned above, the simulation is preferably run on appropriate 'number crunching' hardware, which is most often different from the visualization and steering terminal.

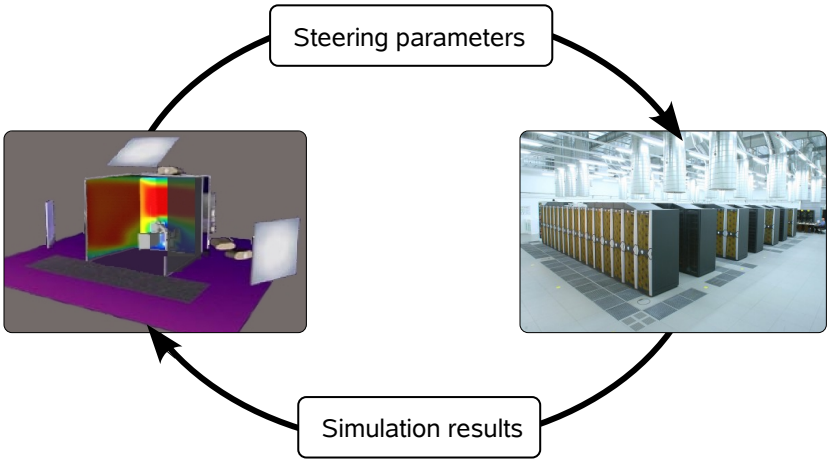

Figure 2.2: *Module concept and data transfer: at the visualization client the user can explore the continuously updated flow data interactively and control the running simulation by modifying geometry, boundary conditions or general flow parameters at the same time. The user's modifications are transmitted to the solver, which immediately incorporates the new setup and, after a single time step of computation, sends back a first trend of the new flow configuration almost without delay. The CFD kernel continues the simulation and transfers current data sets to the visualization and steering client in regular intervals until the next user interaction occurs.*

2.2 Interactive Numerical Kernel

The simulation kernel behind an interactive engineering application has to be designed to meet several special requirements. To provide a convenient level of interactive simulation a specially adapted numerical solver is needed, which has to be able to take into account user modifications fully automatically. Furthermore, the solver must be fast enough to generate useful information describing the flow configuration within a short space of time and give at least an initial trend of the flow behavior within a sub-second time frame. Needless to say, this flexibility comes at a cost and is usually reflected in constraints such as a reduced resolution and complexity of the model.

In order to support different simulation scenarios, the kernel should be able to handle arbitrary geometric setups. In this respect *iFluids* covers a comparatively wide range of applications as it has been used for the simulation of engine rooms of freight ferries, offices, and blood vessels (cf. Fig. 2.3). It is accordingly advantageous that no special data preparation like pre-generation of grids is necessary to permit the immediate use of such geometries in the computational steering framework.

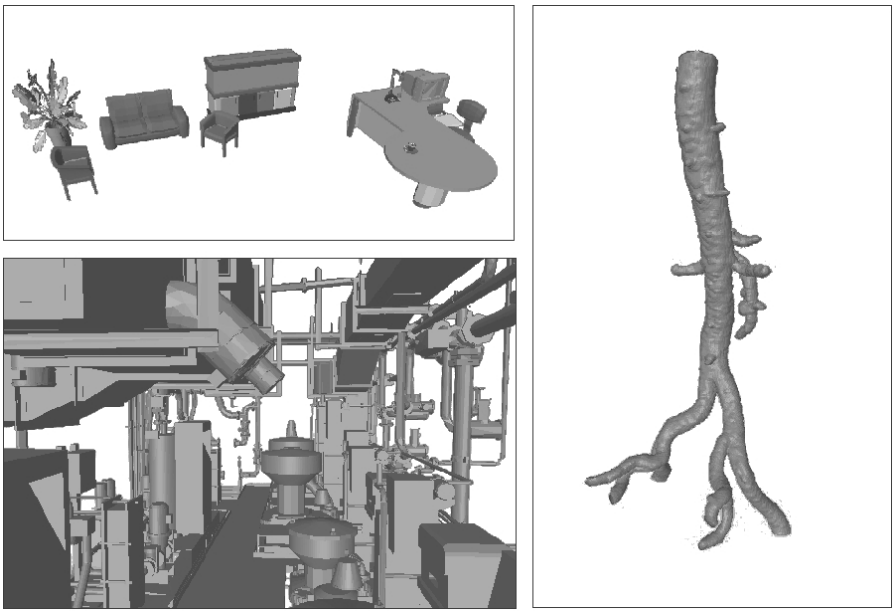

Figure 2.3: *Examples of geometries simulated with* iFluids*: the top picture shows the model of an office, where the flow field develops within the extension of the room. The picure bottom left shows a separator room of a big cargo ferry (Flensburger Schiffbau Gesellschaft, 2007). Here, a comparatively complex geometry dominates the scene. A completely different application of* iFluids *is shown on the right, where an artery 'defines' the region of bloodflow in its interior (data kindly provided by the University of Amsterdam: Section Computational Science (2007)).*

While the simulation is running, the user can enable or disable the accounting

of a turbulence model, he can change the simulated scene with regard to its geometry, its boundary conditions and all relevant flow parameters. Advanced users can also interactively tune the performance of the kernel by choosing adequate optimization strategies depending on the hardware architecture in use.

The solver in *iFluids* is based on the Lattice-Boltzmann method, which has emerged as a complementary technique for the computation of fluid flow phenomena (see e.g. Succi (2001); Kafczyk (2001); Wolf-Gladrow (2000)). Typically, the Lattice-Boltzmann method is implemented on cartesian grids representing the spatial discretization of the simulation domain, thus permitting a fast and fully automatic grid generation. In each time step the Lattice-Boltzmann algorithm computes the *collision* of microscopic, 'virtual particles' modeled statistically by a number of distribution functions at each grid point and 'migrates' the distribution functions of these particles to neighboring lattice sites in the so-called *propagation* step. Fortunately, the computation of the collision does not require any data exchange with neighboring grid nodes and only directly adjacent cells are affected through propagated distributions. Therefore, the Lattice-Boltzmann method allows for an efficient parallelization of the simulation kernel.

2.3 The Visualization and Steering Front-End

Within a computational steering application like *iFluids* the visualization and steering front-end as the central interface between user and simulation is of particular importance. To provide a natural and intuitive way to explore the simulation data and to interact with the running simulation, the user interface for visualization and steering has been combined into a single client front-end. Results are visualized with both the geometry of the investigated scene and its boundary conditions to allow for a better understanding and to support accurate and well-directed interaction. In addition to the conventional desktop interface the visualization client has been implemented to support virtual reality environments with multiple projection screens and tracking of the positions of the user's head and of input devices to allow a better immersion into the simulated scene.

Mercury's OpenInventor (Mercury Computer Systems, Inc., 2007b) is used for visualization and scene-graph manipulation of object geometry and position, navigation, and menu handling. It supports various modifications of objects, such as translation, rotation, and scaling, as well as the transformation of mapped data such as the seed points of a particle tracing for example. Data visualization is realized with the help of the DataViz extension libraries of OpenInventor, which provides a range of standard techniques like streamlines, iso-surfaces, and cross sections.

The main task of the steering interface is to provide user interaction facilities to manipulate the simulation run. Another optional feature allows the initial setup for fluid parameters and start-up geometry to be predefined together with the corresponding boundary conditions before the simulation starts. In this case, the preprocessing module integrated in the computational steering framework can be run as a stand-alone application (Kollinger, 2007).

During the simulation a context-based 3D menu for improved usability in VR environments supports direct access to object parameters like boundary conditions and intuitive interactions with post-processing objects (Marcheix, 2004). New fluid obstacles can be imported from the file system and existing objects can be transformed or removed from the simulated scene. Analogously, it is possible to add and modify boundary conditions during the ongoing simulation.

The visualization and steering clients are again realized as encapsulated modules interacting via clearly defined interfaces to simplify the exchange with other visualization tools. This interface concept is also the prerequisite for extensions such as the adaptation of the single-user application to collaborative multi-client sessions.

Finally, the performance of the communication initiated by forwarding a user interaction from the steering interface to the simulation kernel is of vital importance for a responsive computational steering application.

2.4 Requirements of Interactive Simulation Software

As mentioned, computational steering may be defined as the fusion of the traditionally separated steps of preprocessing, computation and postprocessing into a closed loop. The computational steering application should enable a user to run a simulation and monitor current results to estimate the state or trend of the simulation. In addition, the minimum interaction options should comprise pausing, stopping, and restarting the simulation. An advanced level of interaction is provided by modifying some (fairly simple) simulation parameters. Finally, true computational steering supports a multitude of repeated modifications without the need to restart the simulation. This implies, in particular, that the geometry of the simulated scene can be modified together with boundary conditions, and general flow parameters. To realize a computational steering project of this description, several fundamental prerequisites are identified, as described in brief below.

- **Numerical method:** The underlying numerical method should allow for the easy incorporation of user-initiated modifications. Accordingly, it should be based on a grid or mesh that is suitable for fast generation and modification. Furthermore, the feasibility of on-the-fly visualization of simulation results must be ensured.

- **Computation:** To allow the user to watch the adaptation of the fluid to his geometrical manipulations, the simulation needs to be fast enough. This requires a highly optimized solver running preferably in parallel on adequate hardware architecture.

- **Data visualization:** Current results have to be available and must be displayed with minimum delay. The data visualization module should enable *interactive* exploration of the resultant data. Displaying the geometry of the

simulated scene along with the simulation results makes for a better understanding of the physical behavior.

- **Steering and problem definition:** Steering the application requires an interface which should be combined with the visualization, preferably in a single terminal to provide a more intuitive interaction with the simulation. The main part of the steering process is the problem definition comprising the geometric setup and manipulation as well as setting the boundary condition.

- **Communication:** Regarding the usability of the computational steering application, the communication coupling the modular building blocks has to be flexible and efficient as the user not only expects to be shown continually updated simulation results during data exploration but also wants to have instant responses to his modifications.

In the following chapters the modules making up the computational steering framework will be introduced with special attention paid to the requirements listed above.

Chapter 3

Computational Fluid Dynamics Using the Lattice-Boltzmann Method

This chapter gives a short general introduction into computational fluid dynamics (CFD) followed by a brief derivation of the Lattice-Boltzmann method (LBM) which has been used for the numerical simulations in this thesis. The third section of this chapter describes the simulation kernel implemented for the computational steering project *iFluids*.

3.1 Computational Fluid Dynamics

The conventional form of describing fluid phenomena in CFD is based on the famous Navier-Stokes equations. They can be summarized as two sets of equations to express both mass and momentum conservation. Often, only the simplified Navier-Stokes equations for describing an incompressible Newtonian fluid, i.e. $\rho = const$, are used. They can be found in literature as

$$\frac{\partial u_\alpha}{\partial x_\alpha} = 0 \tag{3.1}$$

$$\frac{\partial u_\alpha}{\partial t} + u_\beta \frac{\partial u_\alpha}{\partial x_\beta} = \frac{1}{\rho}\frac{\partial p}{\partial x_\alpha} + \nu \frac{\partial}{\partial x_\beta}\left(\frac{\partial u_\alpha}{\partial x_\beta}\right) \tag{3.2}$$

with the density ρ, pressure p, velocity $\mathbf{u}$ and the kinematic viscosity ν. Note, that terms containing repeated Greek indices have to be interpreted according to the Einstein summation convention, i.e. $x_\alpha y_\alpha = \sum_{\alpha=1}^{D} x_\alpha y_\alpha$.

There are numerous methods of solving these equations numerically, some classical approaches are for example (wikipedia, 2007a):

Finite Differences Method (FDM): Applying this method the derivatives in the transport equation are approximated by Taylor expansions at discrete points usually on a regular grid. Accordingly, the discretization error is given by the difference between the values on the discrete grid and the exact solution. It is possible to reduce this error by taking terms of higher order

in the Taylor approximation into consideration (Noll, 1993). A problem of this method is that the solution of the difference equations is not necessarily conservative, i.e. it can happen that inward and outward fluxes of a given domain are not in balance (Schönung, 1990).

Finite Volume Method (FVM): FVM may be seen as the classical or standard approach as it is used most often in commercial software and research codes. Here, the computational domain is discretized into so-called control volumes over which the differential equations are integrated. From these volume integrals or their corresponding surface integrals one can derive balance equations which guarantee a conservative discretization. The advantage over the FDM lies in the conservative discretization, which allows non-equidistant and curvilinear meshes (Noll, 1993; Schönung, 1990).

Finite Element Method (FEM): This method is particularly popular for structural analysis of solids, but it is also applicable to fluid dynamics (Schönung, 1990). Instead of solving the partial differential equations directly, solutions to a 'weak formulation' of the equations are sought by minimizing an integral. This method can be applied to unstructured grids consisting of triangles or quadrangles, the latter also being preferred for fluid dynamic problems.

The above-mentioned methods are all based on the discretization of the Navier-Stokes differential equations. An alternative approach to simulate fluid dynamics is the Lattice-Boltzmann method (LBM) (Wolf-Gladrow, 2000). Instead of solving the Navier-Stokes equations directly, LBM can be seen as a discrete microscopic model which conserves mass and momentum by construction. The corresponding macroscopic quantities are obtained through a multi-scale analysis.

3.2 Lattice-Boltzmann Method

The Boltzmann equation is the central equation of transport theory in statistical mechanics and is used, e.g., to describe the distribution of particles in a fluid. It describes the time evolution of the distribution function $f(\mathbf{x}, \mathbf{u}, t)$ of the number of particles in the phase-space volume $\mathrm{d}^3x\ \mathrm{d}^3u$ at time t, where $\mathbf{x}$ and $\mathbf{u}$ are position and velocity, respectively. Considering changes in the particle distributions due to external forces $\mathbf{F}$ or through internal collisions Ω between particles, the Boltzmann equation reads

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial x_\alpha} u_\alpha + \frac{\partial f}{\partial u_\alpha} F_\alpha = \Omega(f). \tag{3.3}$$

Beginning with a discretized version of the Boltzmann equation, the LBM approach computes the dynamics of such statistical particle distributions for a discrete number of velocities on a computational grid or lattice (cf. Fig. 3.1). The

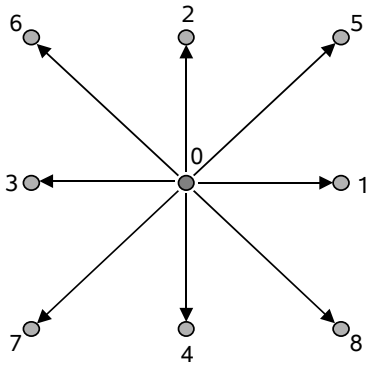

Figure 3.1: *Two-dimensional lattice site in the LBM: This scheme shows nine discrete velocity vectors for a grid point in a 2D lattice, which represent the 8 velocities to the neighboring nodes and the resting 'velocity' 0 in the center of the cell. This is just one example in a number of lattice models $DkQb$, the most popular ones being $D2Q9$, $D3Q15$, and $D3Q19$. In this notation introduced by Qian et al. (1992), k denotes the spatial dimension of simulation space and b refers to the number of lattice velocities. In choosing an appropriate lattice model it is important to bear in mind that a sufficient symmetry of the lattice is guaranteed, otherwise the LBM cannot correctly reflect the Navier-Stokes equations (Frisch et al., 1986).*

statistical description also represents the main improvement of the LBM over its historical origin, the Lattice-Gas automata (LGA)[1].

By design, the LBM method conserves the quantities of mass and momentum to fulfill the hydrodynamic laws. The Lattice-Boltzmann algorithm computes the 'collision' of microscopic, virtual particles and updates the velocity distribution functions in each simulation time-step followed by a 'propagation' step where the migration of these distribution functions to neighboring cells takes place. Typically, the LBM is implemented on uniform Cartesian grids, which makes it particularly well-suited for taking advantage of parallelization and/or vectorization capabilities of high-performance supercomputers and allows to handle complex geometries.

Although the LBM represents a relatively modern approach it has already been extended in many ways. There are (mainly research) codes for multiphase or free surface flow (Ginzburg and Steiner, 2003; Thürey and Rüde, 2004; Tölke, 2001; He et al., 1999; Shan and Chen, 1993), thermal fluid simulations (van Treeck, 2004; Lallemand and Luo, 2003; Mezrhab et al., 2004; van Treeck et al., 2006), acoustic simulations (Lallemand and Luo, 2003; Haydock and Yeomans, 2003; Neuhierl, 2006), and medical simulations (Bernsdorf et al., 2006; Hirabayashi

[1]A lattice gas automaton is a special type of cellular automaton, which is defined by a lattice of cells with a local update rule determining each cells's state. This update rule is applied simultaneously to all cells and only uses information of a cell's current state and that of certain neighboring cells. Each cell can either be empty or occupied by a single particle, this restriction being the characteristic difference from LBM.

et al., 2003; Artoli, 2003; Götz, 2006; Sloot et al., 2004). By now, LBM is a well understood and accepted method of simulating fluid dynamics and is also realized in the commercial product *PowerFLOW* developed by Exa Corporation (Exa Corporation, 2007).

As mentioned above, the LBM has been developed from the Lattice-Gas automata. The main motivation for the transition from LGA to LBM was the desire to remove the statistical noise by replacing the Boolean particle number in a lattice direction by its ensemble average, the so-called density distribution function (wikipedia, 2007b). This replacement has to be accompanied by a consecutive modification of, the discrete collision rules to a continuous function — the collision operator.

There are numerous ways of introducing the Lattice-Boltzmann equation (LBE). Following Chen and Doolen (1998), the starting point is the discrete version of the kinetic Equation (3.3) for the particle distribution function neglecting external forces:

$$f_i(\mathbf{x} + \mathbf{e}_i \triangle x, t + \triangle t) = f_i(\mathbf{x}, t) + \Omega_i(f_i(\ \mathbf{x}, t)), \qquad (i = 0, 1, .., N) \tag{3.4}$$

where f_i represents the particle velocity distribution function along the ith direction (cf. Figure 3.1); Ω_i is the collision operator expressing the rate of change of f_i due to collision. $f_i(\mathbf{x}, t)$ is the probability density of particles in x at time t. Therefore, the macroscopic density $\rho(\mathbf{x}, t)$ can be computed as the zero-th order velocity moment

$$\rho(\mathbf{x}, t) = \sum_{i=0}^{N} f_i(\mathbf{x}, t). \tag{3.5}$$

Moreover, the fluid momentum is the first order velocity moment

$$\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \sum_{i=0}^{N} f_i(\mathbf{x}, t)\mathbf{c}_i \tag{3.6}$$

with the macroscopic velocity $\mathbf{u}(\mathbf{x}, t)$ and the mesoscopic lattice velocity $\mathbf{c}_i$.

Applying the conservation of mass and momentum to the equations, two constraints on the collision operator are found:

$$\sum_{i=0}^{N} \Omega_i(f) = 0 \tag{3.7}$$

$$\sum_{i=0}^{N} \mathbf{c}_i \Omega_i(f) = 0 \tag{3.8}$$

To transform the discrete LBE into a continuous equation accurate up to second order in $\triangle t$, a Taylor expansion is applied:

$$\begin{aligned} f_i(\mathbf{x}+\mathbf{c}_i\triangle t, t+\triangle t) &= f_i(\mathbf{x},t) + \frac{\partial f_i(\mathbf{x},t)}{\partial x_\alpha} c_{i_\alpha}\triangle t + \frac{\partial f_i(\mathbf{x},t)}{\partial t}\triangle t \\ &+ \frac{1}{2}\frac{\partial^2 f_i(\mathbf{x},t)}{\partial x_\alpha \partial x_\beta} c_{i_\alpha} c_{i_\beta}\triangle t^2 + \frac{1}{2}\frac{\partial^2 f_i(\mathbf{x},t)}{\partial t^2}\triangle t^2 \\ &+ \frac{\partial^2 f_i(\mathbf{x},t)}{\partial t \partial x_\alpha} c_{i_\alpha}\triangle t^2 + O(\triangle t^3) \end{aligned} \tag{3.9}$$

After that, the particle distribution functions are expanded by the equilibrium distribution function f_i^{eq} using the Chapman-Enskog multi-scale expansion, i.e.

$$\begin{aligned} f_i &= f_i^{eq} + \varepsilon f_i^{(1)} + \varepsilon^2 f_i^{(2)} + O(\varepsilon^3) \\ &= f_i^{eq} + \varepsilon f_i^{(neq)} \end{aligned} \tag{3.10}$$

with the non-equilibrium distribution function $f_i^{(neq)} = f_i^{(1)} + \varepsilon f_i^{(2)} + O(\varepsilon^2)$.

In analogy to Equations (3.5) and (3.6), f_i^{eq} should satisfy

$$\sum_i f_i^{eq} = \rho, \qquad \sum_i f_i^{eq}\mathbf{c}_i = \rho\mathbf{u}, \tag{3.11}$$

requiring for the non-equilibrium parts $f_i^{(k)}$ with $k = \{1, 2\}$ that

$$\sum_i f_i^{k} = 0, \qquad \sum_i f_i^{k}\mathbf{c}_i = 0. \tag{3.12}$$

Taylor expansion of $\Omega_i(f)$ about f^{eq} assuming $f^{neq} \ll f^{eq}$ yields

$$\Omega_i(f) = \Omega_i(f^{eq}) + \sum_{j=0}^{N} \frac{\partial \Omega_i(f^{eq})}{\partial f_j} f_j^{neq} + \frac{1}{2}\sum_{j=0}^{N}\sum_{k=0}^{N} \frac{\partial^2 \Omega_i(f^{eq})}{\partial f_i \partial f_k} f_j^{neq} f_k^{neq} + O(|f^{neq}|^3) \tag{3.13}$$

From Equation (3.4) we see that $\Omega_i(f^{eq}) = 0$ for $\varepsilon \to 0$. By keeping only terms linear in f_i^{neq} Equation (3.13) can be simplified to the linearized collision operator

$$\Omega_i(f) = \sum_{j=0}^{N} \frac{\partial \Omega_i(f^{eq})}{\partial f_j} f_j^{neq} = \sum_{j=0}^{N} M_{ij}(f_j - f_j^{eq}) \tag{3.14}$$

where $M_{ij} \equiv \frac{\partial \Omega_i(f^{eq})}{\partial f_j}$ satisfying the constraints (Benzi et al., 1992)

$$\sum_{j=0}^{N} M_{ij} = 0, \qquad \sum_{j=0}^{N} c_i M_{ij} = 0. \tag{3.15}$$

Assuming that the local particle distribution relaxes to an equilibrium state with a single rate $1/\tau$ we have $M_{ij} = -\frac{1}{\tau}\delta_{ij}$ and the BGK collision term (Bhatnagar et al., 1954)

$$\Omega_i = -\frac{1}{\tau}(f_i - f_i^{eq}), \tag{3.16}$$

which leads to the LBGK equation

$$f_i(\mathbf{x} + \mathbf{e}_i, t+1) = f_i(\mathbf{x}, t) - \frac{f_i - f_i^{eq}}{\tau}. \tag{3.17}$$

The following section shows that the macroscopic velocity **u** obtained from the solution of this equation fulfills the Navier-Stokes equation up to second order accuracy.
In addition to Equation (3.10), the Chapman-Enskog expansion is employed to obtain

$$\frac{\partial}{\partial t} = \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2}, \qquad \frac{\partial}{\partial x} = \varepsilon\frac{\partial}{\partial x_1}. \tag{3.18}$$

Combining the continuous Taylor-expanded LBE (3.9) and the linearized collision operator (3.14) we get

$$\frac{\partial f_i}{\partial t} + \frac{\partial f_i}{\partial x_\alpha}c_{i_\alpha} + \frac{1}{2}\triangle t(\frac{\partial^2 f_i}{\partial x_\alpha \partial x_\beta}c_{i_\alpha}c_{i_\beta} + \frac{\partial^2 f_i}{\partial t^2} + 2\frac{\partial^2 f_i}{\partial t \partial x_\alpha}c_{i_\alpha}) = \frac{1}{\triangle t}\sum_{j=0}^{N} M_{ij}(f_j - f_j^{eq})). \tag{3.19}$$

Applying the Chapman-Enskog expansions (3.10) and (3.18) now leads to

$$\frac{\partial f_i^{eq}}{\partial t_1} + \frac{\partial f_i^{eq}}{\partial x_{1_\alpha}}c_{i_\alpha} = \frac{1}{\triangle t}\sum_{j=0}^{N} M_{ij}f_j^{(1)} \tag{3.20}$$

to order ε^0 and to

$$\frac{\partial f_i^{eq}}{\partial t_2} + \frac{\partial f_i^{(1)}}{\partial t_1} + \frac{\partial f_i^{(1)}}{\partial x_{1_\alpha}c_{i_\alpha}} + \frac{1}{2}\triangle t(\frac{\partial^2 f_i^{eq}}{\partial x_{1_\alpha}\partial x_{1_\beta}}c_{i_\alpha}c_{i_\beta} + \frac{\partial^2 f_i^{eq}}{\partial t_1^2} + 2\frac{\partial^2 f_i^{eq}}{\partial t_1 \partial x_{1_\alpha}}c_{i_\alpha}) = \frac{1}{\triangle t}\sum_{j=0}^{N} M_{ij}f_j^{(2)} \tag{3.21}$$

to order ε^1. After some algebra the first order equation can be simplified to

$$\frac{\partial f_i^{eq}}{\partial t_2} + \sum_{j=0}^{N}(\delta_{ij} + \frac{1}{2}M_{ij})(\frac{\partial f_j^{(1)}}{\partial t_1} + \frac{\partial f_j^{(1)}}{\partial x_{1_\alpha}}c_{i_\alpha}) = \frac{1}{\triangle t}\sum_{j=0}^{N} M_{ij}f_j^{(2)}. \tag{3.22}$$

Now the zero-th order velocity moments of Equation (3.20) and (3.22) are calculated and recombined with respect to their time scale, i.e.

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_\alpha}{\partial x_\alpha} = 0, \tag{3.23}$$

which is the mass conservation satisfied by the LBE up to second order accuracy.

In the next step the first order velocities are calculated accordingly and after recombining the scales we arrive at

$$\frac{\partial \rho u_\alpha}{\partial t} + \frac{\partial}{\partial x_\beta}\Pi = 0 \tag{3.24}$$

with the momentum flux density tensor

$$\Pi = \sum_{i=0}^{N} c_{i_\alpha} c_{i_\beta} (f_i^{eq} + \sum_{j=0}^{N} (\delta_{ij} + \frac{1}{2} M_{ij}) \varepsilon f_j^{(1)}). \tag{3.25}$$

Finally, we need to specify the equilibrium distributions corresponding to the lattice structure. To simplify the derivation without loosing generality, we take a look at a two-dimensional square lattice (cf. 3.1, Chen and Doolen (1998)). Therefore, nine lattice velocities are defined:

$$\begin{aligned} \mathbf{e}_i &= (\cos(\frac{\pi}{2}(i-1), \sin(\frac{\pi}{2}(i-1)), && \text{for } i = 1,2,3,4; \\ \mathbf{e}_i &= \sqrt{2}(\cos(\frac{\pi}{2}(i-1) + \frac{\pi}{4}), \sin(\frac{\pi}{2}(i-1) + \frac{\pi}{4})), && \text{for } i = 5,6,7,8; \\ \mathbf{e}_0 &= 0 \qquad \text{for the zero-speed velocity.} \end{aligned} \tag{3.26}$$

According to Chen et al. (1992) the general form of the equilibrium distribution function can be written to $O(u^2)$ as

$$f_i^{eq} = \rho(a + b\mathbf{e}_i \cdot \mathbf{u} + c(\mathbf{e}_i \cdot \mathbf{u})^2 + du^2 \tag{3.27}$$

with the lattice constants a, b, c, and d. It is only for small Mach numbers and by obeying the constraints in (3.11) that these constants can be determined as

$$f_i^{eq} = \rho\omega_i(1 + \frac{c_{i_\alpha} u_\alpha}{c_s^2} + \frac{1}{2c_s^2}(\frac{c_{i_\alpha} c_{i_\beta}}{c_s^2} - \delta_{\alpha\beta}) u_\alpha u_\beta) \tag{3.28}$$

with a sound speed $c_s = \frac{1}{\sqrt{3}}$ and $\omega_0 = \frac{4}{9}, \omega_{1..4} = \frac{1}{9}$, and $\omega_{5..8} = \frac{1}{36}$.

Inserting the equilibria into the flux tensor (3.25) we obtain

$$\begin{aligned} \Pi_{\alpha\beta}^{(0)} &= \sum_{i=0}^{N} c_{i_\alpha} c_{i_\beta} f_i^{eq} = p\delta_{\alpha\beta} + \rho u_\alpha u_\beta, \\ \Pi_{\alpha\beta}^{(1)} &= \sum_{i=0}^{N} c_{i_\alpha} c_{i_\beta} \sum_{j=0}^{N} (\delta_{ij} + \frac{1}{2} M_{ij} \varepsilon f_j^{(1)}) = \nu(\frac{\partial \rho u_\alpha}{\partial x_\beta} + \frac{\partial \rho u_\beta}{\partial x_\alpha}) \end{aligned} \tag{3.29}$$

where the pressure $p = \frac{\rho}{3}$ and the kinematic viscosity $\nu = \frac{(2\tau - 1)}{6}$.
The resulting momentum equation is now

$$\rho(\frac{\partial u_\alpha}{\partial t} + u_\beta \frac{\partial u_\alpha}{\partial x_\beta}) = \frac{\partial p}{\partial x_\alpha} + \nu \frac{\partial}{\partial x_\beta} \left(\frac{\partial \rho u_\alpha}{\partial x_\beta} + \frac{\partial \rho u_\beta}{\partial x_\alpha} \right), \tag{3.30}$$

which resembles the Navier-Stokes equation as long as the density variation δp is small enough.

3.3 Implementation of the LBM Solver

The Lattice-Boltzmann kernel implemented for the computational steering application *iFluids* is a BGK solver based on a $D3Q15$ lattice.

The 15 velocities are defined as

$$c_{i=0,..,14} = c_s \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \end{pmatrix},$$

and the corresponding equilibria can be found to be

$$\begin{aligned} f_0^{eq} &= \frac{2}{9}\rho\left(1 - \frac{3}{2}\mathbf{uu}\right), \\ f_i^{eq} &= \frac{1}{9}\rho\left(1 + 3\mathbf{e}_i\mathbf{u} + \frac{9}{2}\left(\mathbf{e}_i\mathbf{u}\right) - \frac{3}{2}\mathbf{uu}\right), \qquad \text{for } i = 7,..,14 \\ f_i^{eq} &= \frac{1}{72}\rho\left(1 + 3\mathbf{e}_i\mathbf{u} + \frac{9}{2}\left(\mathbf{e}_i\mathbf{u}\right) - \frac{3}{2}\mathbf{uu}\right), \qquad \text{for } i = 7,..,14, \end{aligned}$$

where $e_i = \frac{1}{c_s}c_i$.

For high Reynolds numbers (see (3.31)) an optional turbulence model is integrated into the solver to take unresolved sub-grid phenomena into account. To decide whether a turbulence model is needed or not the Reynolds number must be estimated as

$$Re = \frac{uL}{\nu} \tag{3.31}$$

with u the mean fluid velocity, L the characteristic length, and ν the kinematic fluid viscosity.

Laminar flow occurs at low Reynolds numbers (i.e. Re $\lessapprox$ 2100), while turbulent flow occurs at high Reynolds numbers (i.e. Re $\gtrapprox$ 4000). The transition between laminar and turbulent flow is often indicated by a critical Reynolds number which depends on the exact flow configuration.

Assuming a typical indoor ventilation setup, the air velocity can be estimated as ≈ 0.13m/s to keep the room climate comfortable. A ceiling height of 2.75m is taken to be the characteristic length, which leads to a Reynolds-number of $Re \approx 23800$ for a kinematic fluid viscosity of $1.5 \cdot 10^5$ (at a temperature of 20^oC).

A popular turbulence model is the Large Eddy Simulation (LES) using the Smagorinsky sub-grid model. This model has also been implemented for the Lattice-Boltzmann solver within the *iFluids* computational steering application. The advantage of this model is its relatively low computational cost, compared to the large parameter stability improvement of the solver. Simulations on comparatively coarse grids, in particular, are made possible with this enhancement (Thürey and Rüde, 2005).

The key idea of the Smagorinsky sub-grid model is the concept of an eddy viscosity ν_t as a synthetic indicator of the damping effects on large scales caused by small scales. ν_t is related to the local strain tensor as (Succi et al., 1995; Kafczyk, 2001; Hartmann, 2005)

$$\nu_t = C_s^2 \sqrt{S^2} \tag{3.32}$$

with

$$S_{\alpha\beta} = \frac{1}{2} \left(\partial_\alpha u_\beta + \partial_\beta u_\alpha \right). \tag{3.33}$$

C_s is the empiric Smagorinsky constant. An obvious advantage of the LBM using this turbulence model is the local availability of the strain tensor at each lattice site and is obtained by (cf. Thürey and Rüde (2005); Artoli (2003)):

$$S_{\alpha\beta} = \sum_{i=0}^{14} e_{i_\alpha} e_{i_\beta} (f_i - f_i^{eq}). \tag{3.34}$$

Chapter 4

Fluid Simulation on Supercomputers

After a short introduction into High-Performance Computing (HPC) in general, this chapter describes the optimization and parallelization strategies for the Lattice-Boltzmann solver in the computational steering framework *iFluids*. The solver has mainly been optimized for the Hitachi SR8000 system at LRZ but has also been ported to SGI Altix 3700/4700 systems. Therefore, the architecture of the Hitachi SR8000 and its features will be introduced first, followed by a detailed discussion of the methods applied for parallelization and optimization. Then, the architecture of the SGI Altix will be sketched to be able to motivate the main differences in the parallelization and optimization approach. Finally, the simulation performance of both machines will be compared.

4.1 High-Performance Computing

High-Performance Computing refers to the utilization of supercomputers for performing numerical simulations. In the late 1980s, the US Government defined supercomputers as systems having one or more processors capable of achieving an aggregate performance of more than 100 MFlops/s[1] (Standish, 2006). In an industrial sector developing so fast that its cumulative performance doubles almost every year (see Figure 4.1) it is obvious that definitions like this are untenable. Even today's PCs (2006) already reach up to 5 GFlops. Another approach defines supercomputers as computing resources which provide more than an order of magnitude higher computing power than is usually available on modern PCs (Standish, 2006).

Even with this definition, HPC systems span the range from departmental clusters of off-the-shelf workstations up to the largest and fastest supercomputers in the world. Traditionally, HPC is used in scientific and engineering fields such as Computational Fluid Dynamics, Molecular Dynamics, and Quantum Chromo-Dynamics. Due to their scientific origin HPC codes are still mainly written in Fortran, much fewer in C. Newer implementations also use object-oriented languages such as C++ and even Java.

[1] MFlops = Mega Flops = 10^6 Floating Point Operations per Second

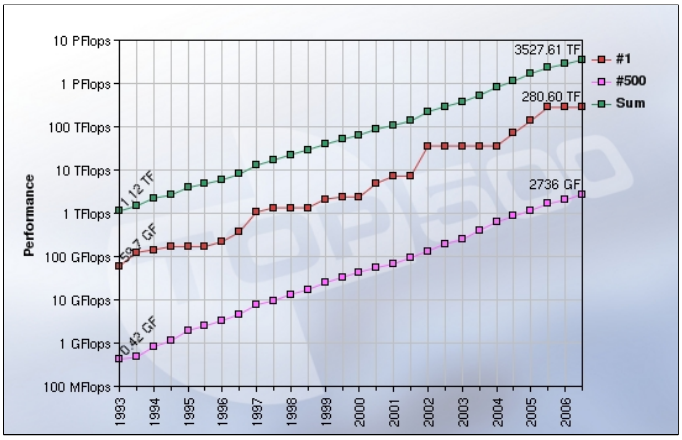

Figure 4.1: *This chart shows the performance development of the top 500 supercomputers during the last decade. The peak performance of the fastest supercomputer on the list is marked by the red line, while the performance the weakest is represented by the purple line. The green line shows the accumulated peak performance of all supercomputers on the list (taken from TOP500 (2007)). Evidently, the performance of these systems nearly doubles every year.*

As described in Standish (2006) and Dowd and Severance (1998), high-performance computers are usually classified into three categories according to their processing architecture:

- **Vector computers**: The CPUs of vector machines provide instructions with vector operands that allow the CPU to efficiently exploit the vector hardware with its multiple arithmetic units. By organizing memory in banks and using fast memory technology running at CPU clock speed the bandwidth between CPU and memory can be increased dramatically. Optimizing compilers are able to transform certain data access patterns into series of vector operations, thus allowing some legacy code to be used without change. The downside of such sophisticated memory subsystems and custom processors is their cost. Vector computers are 20 to 40 times more expensive per peak MFlop than commodity processors. One also has to keep in mind that vector systems are highly proprietary architectures and as such depend on specialized software. Due to the resulting portability issues they suffer from a restricted range of available software.

- **Symmetric Multi Processor (SMP)** and **Shared-Memory Systems**: SMP computers use commodity or RISC-based processors sharing the main memory of the system. By having multiple processors attempting to access memory concurrently, the memory bottleneck of commodity workstations is compounded, so that modern SMP designs arrange the connection between memory and CPU via so-called crossbars. Here, memory is laid-out in 'local' segments for each CPU which other CPUs can access only 'remotely' through the crossbar network with a small communication overhead. In

addition, technical measures need to be implemented to ensure memory consistency between the processors' caches. Besides the increased bandwidth, the biggest advantage with this technology is that large amounts of memory are available to single processors. However, there is also a possible performance penalty with regard to data placement: For optimal performance a program does not only need a high locality of (cache) references, the amount of data moved between CPUs also has to be minimized due to the overhead of the cache-coherence mechanism.

- **Clusters of workstations or PCs** or **Distributed-Memory Systems**: In principle, clusters offer ultimate scalability — if you need more CPU power, just add another PC at commodity cost[2]. However, the number of networking components per CPU will increase logarithmically with the total number of CPUs in the cluster, and will eventually dominate the cost. In practice, machines with 1000s of CPUs are possible in this way. The downside of distributed-memory machines is that exploitation of parallelism must be handled explicitly by the programmer.

Besides the different kind of high-performance resources in use, the performance gain achievable through parallelization depends on the simulation problem. As a quantitative measure the *parallel speedup* has been introduced, which refers to the factor indicating how much faster the parallel program version can be executed as compared to its serial counterpart. Accordingly, the speedup is defined by

$$S(n_p) = \frac{t_1}{t_{n_p}} \tag{4.1}$$

where t_1 and t_{n_p} represent the running time of the serial application on one processor and the parallel version using n_p processors, respectively. From Equ. (4.1) the *parallel efficiency* is derived

$$E(n_p) = \frac{S(n_p)}{n_p} = \frac{t_1}{t_{n_p} n_p}. \tag{4.2}$$

It assesses the per-processor utilization for the parallel program as a fraction of the serial execution speed.

Different algorithms usually are not equally suited for parallelization, depending on their fraction of non-parallelizable code. To estimate the maximum theoretical speedup which can be achieved with a given amount of resources, *Amdahl's Law* can be used, namely

$$S(n_p) = \frac{1}{\alpha + \frac{1-\alpha}{n_p}}. \tag{4.3}$$

[2] At the current state of the art the scalability in systems with a commodity interconnect (Gigabit Ethernet) is limited to approximately 1000 CPUs

Here, n_p refers to the number of processors in use and α measures the serial, i.e. non-parallelizable, fraction of time of the code, which is usually non-trivial to determine. In the limit of an infinite number of processors this leads to a maximum speedup of

$$S_{max} = \frac{1}{\alpha}, \tag{4.4}$$

which cannot be exceeded.

In practice, the theoretical maximum speedup through parallelization is further impaired by the so-called *parallel overhead* due to thread creation and scheduling, communication, and synchronization. This overhead introduces further terms into Amdahl's Law. It then typically takes the form

$$S(n_p) = \frac{1}{(\alpha + \beta) + \frac{1-\alpha}{n_p} + kn_p}. \tag{4.5}$$

with positive parameters β and k which depend on the specific communication pattern of the application and the interconnect hardware. Both parameters worsen the speed-up; the k term in fact degrades performance for a sufficiently large n_p.

The quantities introduced above allow to describe the possible improvements due to parallelization. In addition, different kinds of resources can be validated with respect to their suitability for the specific numerical problem.

4.2 Hitachi SR8000-F1 System Architecture

In 2000, the Leibniz Computing Center (LRZ) in Munich installed the first Bavarian High-Performance Supercomputer, a Hitachi SR8000-F1 pseudo-vector machine. This system was targeted to serve as one of three federal German top-level compute servers. At the time of installation it was the fastest supercomputer in Europe and the first TFlop/s machine for general purpose research in the world. It ranked at position 5 of the Top 500 Supercomputer List (TOP500, 2007) in June 2000. In early 2002 the installation was upgraded in a second installation phase and reached rank 14 of the Top 500 list in June 2002 (TOP500, 2007) with a LINPACK performance of 1.65 TeraFlop/s (2 TeraFlop/s peak performance).

Although LRZ's Hitachi SR8000 was shut down in 2006, this machine and the optimizations with regard to the implementation of *iFluids* will be described in the following, because insights gained on this machine can easily be transferred to current vector or pseudo-vector machines. To this day, this class of supercomputers is still the best-suited for Lattice Boltzmann applications (Wellein et al., 2006).

Hardware Description

The Hitachi SR8000 at LRZ consisted of 168 nodes, each being a RISC-based[3] Symmetric Multiprocessor[4] derived from IBM's PowerPC architecture with eight compute CPUs and one CPU dedicated to the operating system. Each CPU offered a peak performance of 1.5 GFlop/s producing a theoretic total of 12 GFlop/s per node (8 CPUs with 1.5 GFlop/s), while the real application performance of the SR8000 reached on average 1.5 to 2.5 GFlop/s per node. The nodes were equipped with either 8 or 16 GBytes of shared memory and could be accessed with the (still) striking bidirectional bandwidth of 32 GBytes/s.

The SR8000 was a representative of the rather unusual pseudo-vector architecture. This type of machine enables the use of both the vector (PVP mode) and the (parallel) scalar programming paradigm (COMPAS mode) for suitably structured codes, especially codes designed for classical vector systems (Leibniz Rechenzentrum München, 2007). The different paradigms can easily be activated by the programmer through compiler directives. In the following these two special modes are described in more detail.

Co-Operative Micro Processors in Single Address Space

Co-Operative Micro Processors in Single Address Space (COMPAS) is Hitachi's name for the automatic distribution of computational work among the 8 CPUs of an SMP node by the compiler (see Figure 4.2) or manually by the user through insertion of parallelization directives. In order to achieve optimal performance when processing loops — even for loops with comparatively small granularity — a rapid simultaneous start-up of processes is provided. Cache coherency is guaranteed automatically when a fork or a join sequence is executed. This is important, because the data stored by a processor might be referred to by another processor executing a succeeding part (Tamaki et al., 1999).

Pseudo-Vector Processing

Traditionally, a vector CPU executes operations in a vector pipeline which delivers one or more memory references per cycle to the multi-element vector registers of the CPU. In this way, the arithmetic units are continuously fed with data requiring, however, an expensive interleaved and pipelined memory subsystem.

Aiming particularly at scientific computing, Hitachi extended IBM's PowerPC RISC architecture to 160 floating point registers and added pre-load and pre-fetch capabilities to fully exploit the available memory bandwidth and thus alleviate the typical main deficit of RISC-based systems in comparison to vector CPUs. Hitachi refers to this extension as Pseudo-Vector Processing (PVP). Through its PVP

[3] A Reduced Instruction Set Computer (RISC) is based on a processor architecture with reduced chip complexity by using simpler instructions. Thus, the microcode layer with its associated overhead can be eliminated to improve performance (answers.com, 2007).

[4] Symmetric Multiprocessing (SMP) is a multiprocessor computer architecture where two or more identical processors are connected to a single shared main memory (wikipedia, 2007d)

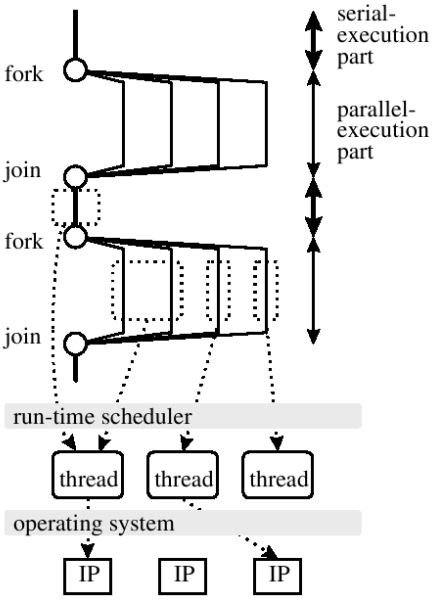

Figure 4.2: *The parallel structure of the DO-loops is a forkjoin, which consists of a serial execution part and parallel execution part appearing one after the other. The serial part is assigned to one thread and executed on one processor, the parallel parts are assigned to multiple threads and executed on multiple processors (taken from Tamaki et al. (1999)).*

features the SR8000 can schedule the fetching of memory in a pipelined manner timely before arithmetic execution, allowing for non-blocking execution in a manner comparable with vector processors.

Pre-fetch transfers requested *cache lines* of data asynchronously from the main memory to the cache, whereas pre-load transfers requested *element* data — again asynchronously — from memory directly to the registers. Both pre-fetch and pre-load (see Fig. 4.3) are useful mainly for codes with small or no cache reuse ratio (for in cache loads indeed than is a small performance penalty), i.e. memory-bound codes. Whether a pre-fetch or pre-load is more efficient depends on how the memory references are organized within the code.

Pre-fetch is the preferred method for referencing contiguous memory areas, whereas pre-load will give better performance for discontiguous (e.g. stride longer than 2) accesses. The reason for this is that, in the latter case, pre-fetch will induce transfers which for the most part deliver unreferenced data to the cache, while for contiguous data pre-fetch will deliver double the bandwidth of pre-load. The compilers choice between pre-fetching and pre-loading can (and sometimes must) be overridden (or suppressed) by a directive in the code.

By accessing the cache when data is in the cache but accessing the main memory in a pseudo-vector manner when data is not in the cache the PVP architecture provides a stable and high data-reference throughput (Tamaki et al., 1999).

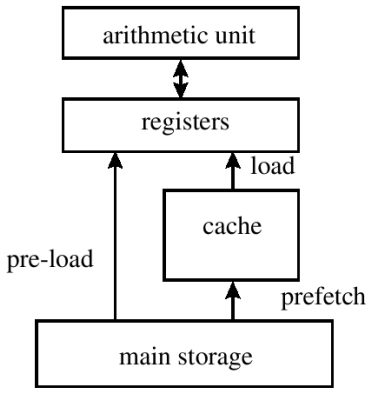

Figure 4.3: *Depending on the memory reference organization within the code, pre-fetch or pre-load is used for data access. Pre-fetch is efficient for contiguous data, since whole cache lines are transferred. For single data pre-load the memory is directly accessed and copied into the CPU registers (taken from Tamaki et al. (1999)).*

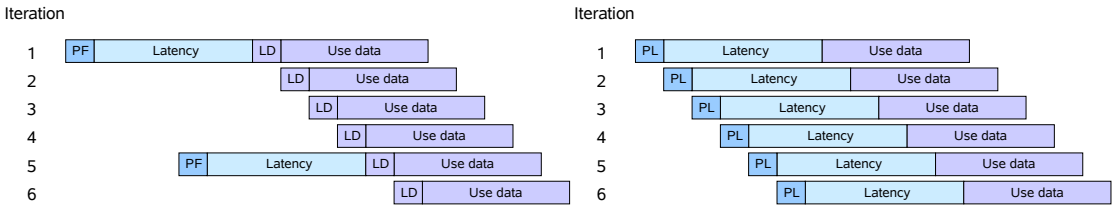

Figure 4.4: *Pre-fetch and pre-load: On the left side of this figure the data use by pre-fetch is sketched, on the right its pre-load counterpart. Pre-fetch loads a complete cache line (of contiguous data) into the cache from where data elements can be loaded and processed in vector manner by the CPU. With a pre-load single data elements are delivered from the memory directly to the CPU (taken from Lanfear (2000)).*

4.3 Parallelization of the Lattice-Boltzmann Solver

As described above, the Hitachi SR8000 offers possibilities of parallelization on several levels (cf. Figure 4.5). Between computing nodes the computation can be distributed using message passing, typically via the MPI library[5]. Within a node the workload can be distributed over the 8 computation CPUs via OpenMP or Hitachi's COMPAS-mode. Finally, after careful manual optimization of the innermost loops the computation can be vectorized to fully exploit the PVP capabilities of the hardware. In this way, a natural hierarchy of parallelization methods is given by the SR8000 architecture with MPI providing work sharing on the coarsest level.

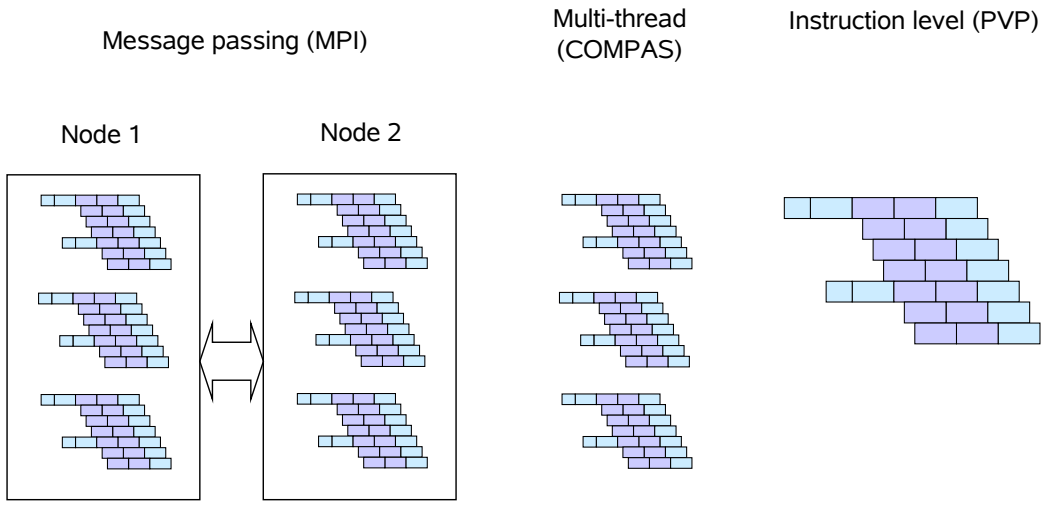

Figure 4.5: *Three levels of parallelization: Communication via MPI exchanges data between nodes of 8 CPUs each. On each node the computational work is processed in COMPAS mode by 8 threads in parallel in a shared-memory environment. Due to the pseudo-vectorization capabilities of the Hitachi SR8000, the loops can be executed vector-wise (taken from Lanfear (2000)).*

In comparison to other approaches to simulate fluid dynamics the Lattice-Boltzmann method is particularly well-suited for parallelization. As described in Chapter 3 the collision step can be computed without the need for interaction with neighboring grid points, since the collision operator requires only data which are locally available. Therefore, the calculation of this step can be conducted without communication between different nodes. In the propagation step the advection of particle distributions comprises only the migration of distribu-

[5]The Message Passing Interface (MPI) is a standardized communication library (MPI-Forum, 2007) covering basic point-to-point (send/receive) and advanced (collective) functionality. It is the most common method of programming multi-processor systems with distributed memory. In basic message passing, the processes are coordinated by explicit communication, i.e., sending and receiving of data (Pacheco, 1996)

tions from one grid point to its next neighbors corresponding to their directions and the Lattice-Boltzmann model in use. The required communication can be kept simple, since only cells at the borders of each domain boundary have to be exchanged (only point to point communication between domain boundaries is required).

Domain Decomposition

Since MPI communication has been designed for parallelization on distributed-memory systems, its use, by nature, requires partitioning of the computational work through domain decomposition. An appropriate domain decomposition results in equal distribution of the workload over all nodes (*load balancing*), and tries to keep inter-node communication as efficient as possible since here is the most sensitive spot for introducing performance-limiting latencies. Depending on the problem characteristics domain decomposition can be far from trivial and, thus, has emerged into a research field in its own right (Domain Decompostion, 2007).

Specifically for the Lattice-Boltzmann method, several approaches for domain decomposition have been taken. An approach presented in Freudiger et al. (submitted 2007) is based on hierarchical grids and uses the free METIS library (METIS, 2007). METIS is a partitioning tool for irregular graphs and FE-meshes. For example, this library has been used by Schulz et al. (2002) for the simulation of porous media with the Lattice-Boltzmann method, since the data reference layout for cell information in their approach is based on lists because a major part of the simulation volume is no longer referenced. When using regular grids as in *iFluids*, cuboidal subdomains are often more advantageous. Satofuka and Nishioka (1999) have found that on a Hitachi SR2201 pseudo-vector machine a 2D domain decomposition into slices is more efficient than into boxes (the authors are not sure about the reason for this). Figure 4.6 shows different cuboidal decompositions, which divide the domain along one, two, or three axes to demonstrate the impact on the number of a process' communication partners and the amount of communication data. *iFluids* supports user-defined domain decomposition, i.e. one can define how many subdivisions should be created along each axis. For determining the best domain decomposition one has to consider many aspects. The efficiency of the decomposition possibilities depends on factors like the total number of processes available, the hardware in use; whether it is possible to exploit optimization features within the reduced problem, whether the memory layout influences performance (latency and bandwith capabilities, distributed or shared memory). Furthermore, it depends on the MPI implementation (e.g. efficiency of derived data types or dependencies between communication partners) and, of course, on the characteristics of the simulation. The three-level parallelization particular to the Hitachi SR8000 suggested a standard decomposition into slices for *iFluids*.

As mentioned above, a decomposition aspect besides the communication volume is load balancing. It is clear that dividing a domain into regular cuboids

does not necessarily achieve optimal load balance. Load balance, however, is not (yet) considered within the *iFluids* computational steering framework, because due to the often-changing geometry during the simulation this would require the domain to be re-decomposed after each geometry modification. Also, the idea of an adaptive decomposition was not considered within this thesis.

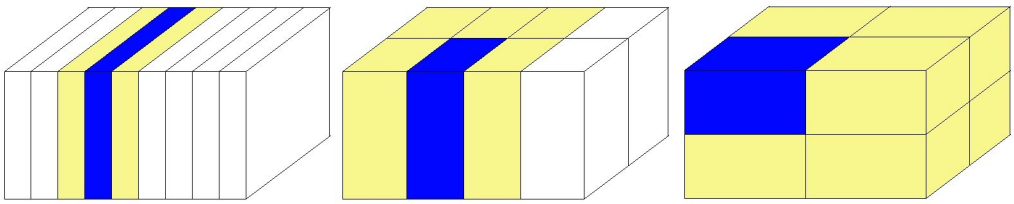

Figure 4.6: *Different layouts of domain decomposition: The yellow-colored domains represent the communication partners of a singled-out process (colored blue) in an LBM simulation (e.g. the D3Q15 model). It is evident that the more cubical the domains are divided up, the smaller is the communication volume and more communication partners are involved.*

MPI Communication

After partitioning the computational work and assigning it to different processes, the next aspect to consider is the MPI layout for inter-process communication. To achieve the best possible communication performance between the SMP nodes of the Hitachi SR8000, its vendor-optimized MPI libraries are used. The interprocess communication consists of multiple sends and receives which have been implemented using non-blocking MPI routines to avoid waiting times as in the case of forced communication order. The data to be exchanged during a propagation step is illustrated for a 2D example in Figure 4.7. Usually, this data is not stored contiguously and has to be specially prepared for sending. There are basically three methods (when using C/C++) to prepare the sending process. One approach, for example, defines derived datatypes as MPI_struct. Here, the memory locations of the relevant data collected for a send are described once and can be used as a "stencil" for each call. On distributed-memory systems MPI copies these data into internal buffers for sending. Another possibility is to use MPI_pack and MPI_unpack, which is, however, less comfortable to implement. Likewise, the data in this case is also usually copied into internal buffers before sending. The third option is to manually copy the data into a continuous vector and send it to the communication partner. Luecke and Wang (2005) investigated three possibilities with regard to their performance and found the use of derived MPI datatypes most efficient. However, this result probably depends on the hardware and the MPI implementation. Within *iFluids* derived MPI data types are used as the standard method.

Domain decompositions into slices offer a fourth option of data transfer with overlapping domains (see Figure 4.8) as has been demonstrated by Pohl et al.

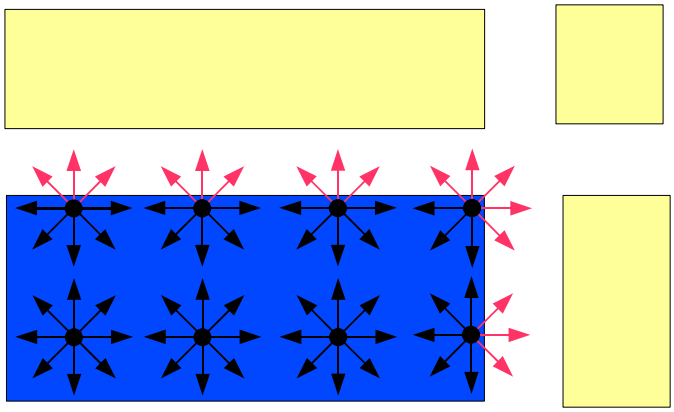

Figure 4.7: *Data Exchange in two dimensions: The red arrows represent distributions of domain boundary nodes which have to be sent to neighboring domains. As there is no simple linear data layout to store these distributions contiguously in memory, one has to design specific mechanisms for sending this data efficiently via MPI.*

(2004). For geometric dimensions of dim_x x dim_y x dim_z and N subdomain slices orthogonal to the x-axis this increases the total number of cells by a factor of $1 + \frac{2(N-1)}{dim_x}$. By overlapping the domains it is possible to exchange the complete data set of all particle distribution as a contiguous data set. Although the communication volume increases approximately by a factor of 2 in this approach, this ansatz still leads to an overall performance gain.

For additional parallelization within an MPI process, the main computing loops were distributed over the 8 CPUs of a node by insertion of COMPAS or OpenMP directives. For exploiting this additional option of parallelization, the code needed to be adapted as described along with further optimizations in the following section.

4.4 Optimization of the Simulation Kernel

Besides parallelizing the computational kernel, the main computing loops in their original form needed rewriting and manual optimization to fully exploit Hitachi's vectorizing and software-pipelining capabilities. Figure 4.9 shows the striking performance gain of an optimized version of the original straight-forward parallel Lattice-Boltzmann kernel that had been used in Kühner (2003).

Collision and Propagation

The LBM is usually divided into the steps of collision and propagation (see Chapter 3), which — at first glance — implicates a code structure of separated loops for collision and propagation. To reduce the data transfer between main memory and processor and for better cache utilization, these steps can be fused into one

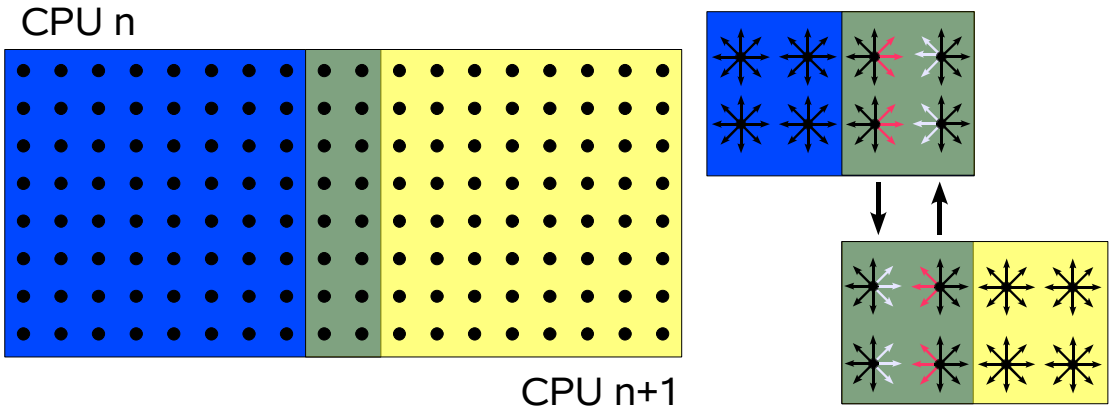

Figure 4.8: *Overlapping domains: Slice subdomains can be overlapped at their boundaries. In the figure white arrows depict distributions which are not available within a process and have to be received from the neighboring domain. There, the required distributions in question are marked red, while black refers to distributions which are computed within the domain independently. By exchanging the complete row before the last one (stored contiguously in memory) to the next domain, black distributions are overwritten by identical values and red ones fill in the gaps.*

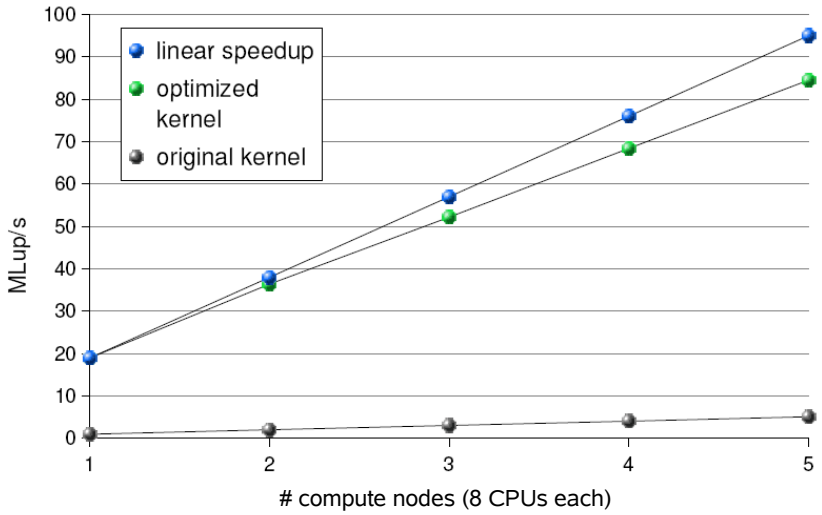

Figure 4.9: *Comparison of performance before and after manual optimizations. By paying special attention to the capabilities and characteristics of the Hitachi SR8000 it was possible to achieve a considerable performance gain over a straight-forward parallel Lattice-Boltzmann implementation.*

loop (Wilke et al., 2003; Wellein et al., 2006). In order to simplify the code structure with respect to propagation, two arrays are used, which hold the distribution densities of successive timesteps t and $t + 1$. Correspondingly, there is no need to care about the order of updating the neighboring cells in the propagation step anymore, and a complex code structure can be avoided.

There are two possibilities of implementing the propagation and collision loop: the so-called *pull* and *push* version. In the push version collision is computed first and the new distributions are propagated to the neighboring cells. In contrast, the pull version collects the relevant distributions from the neighboring cells first and then computes the collision on this basis. For the modeling of boundary conditions the pull-version is advantageous (see Crouse (2003)), but (as claimed by Pohl et al. (2004)) the push version performs better on the Hitachi SR8000 due to the different memory access patterns and is therefore implemented within the Lattice Boltzmann solver presented in this thesis.

Data Layout and Access

Regarding the computation of the collision step it is advantageous to store all distributions of a cell contiguously in memory due to the line-fetching cache access. In C/C++ the last index of an array addresses memory locations in linear succession and should thus be traversed in the inner-most loops. Correspondingly, the array layout reads `f[x][y][z][i]`, where x, y and z are the dimensions of the simulation domain and i refers to the number of distributions per cell (cf. Fig. 4.10). For the immediately following propagation this data layout causes loads from memory locations distant from the location of the current cell distributions. Because of Hitachi's pre-fetch capabilities combined with its high bandwidth from the main memory this disadvantage can be compensated, however.

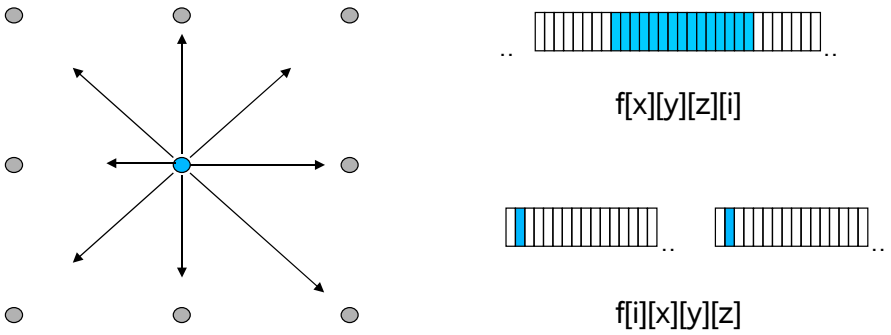

Figure 4.10: *On the left the distribution functions of a lattice site in a 2D example have been sketched. Two different memory patterns of storing a node's distribution functions within an array are visualized on the right. The top pattern has the distribution functions for one node arranged consecutively, whereas the bottom layout stacks the arrays of the different distribution functions.*

In trying to improve the efficiency of data access, pointer arithmetic has been eliminated as far as possible. For example, the three distinct indices x, y and z have been merged into a combined running loop index xyz, i.e., `f[x][y][z][i]`$\rightarrow$`f_ptr[xyz+i]`. As a result, the compiler is able to analyze and optimize the simplified loop structure in this code version much more efficiently with regard to vectorization.

Vectorization and Software Pipelining

Pseudo-vectorization (Fig. 4.11) and software pipelining (Fig. 4.12) refer to special hardware capabilities of the Hitachi SR8000, which are not generally available on other architectures. By these two means code can be sped up considerably on the Hitachi, but only at the cost of careful code tuning and through assisting the compiler.

To benefit from pseudo-vectorization the data layout has been designed such that loops operate on long linear arrays. In particular, long innermost loops are advantageous with regard to pseudo-vectorization (Hager et al., 2003).

To make the code accessible to software pipelining, conditional statements for handling the different boundary conditions have been removed. In case of a predominant number of fluid nodes, the if-statements get replaced by equivalent arithmetic floating point operations, i.e., Boolean expressions are mapped onto real-valued coefficient arrays for multiplication as shown in Figure 4.13. Because of the data locality of the collision (see Chapter 3) and the usage of two arrays for the timesteps t and $t+1$ (see above) no data dependencies between two loops cycles occur, which would decrease the software pipelining's efficiency or even prohibit it completely.

This, of course, causes extra computational cost. Nevertheless, an amazing performance gain can still be achieved with this method, because the avoidance of the even costlier branching instructions — as introduced by the standard implementation of conditionals — turns out to be several times more effective.

Another optimization approach especially suited for fluid scenarios characterized by a high percentage of wall nodes (e.g. porous media) is to introduce lists storing nodes of the same type of boundary condition. This method requires extra memory and, additionally, the velocity distributions of one node and those of the next one in the list may be located far away from each other in memory. An additional performance gain apart from software pipelining and vectorization is achieved by these lists by skipping the large fraction of internal nodes in non-fluid volume areas. The simulation of blood flow within an artery (see Figure 2.3) may serve as an example which benefits strongly from this kind of optimization as here the fraction of fluid nodes to all nodes within the bounding box typically lies in the range from 5% to 15%.

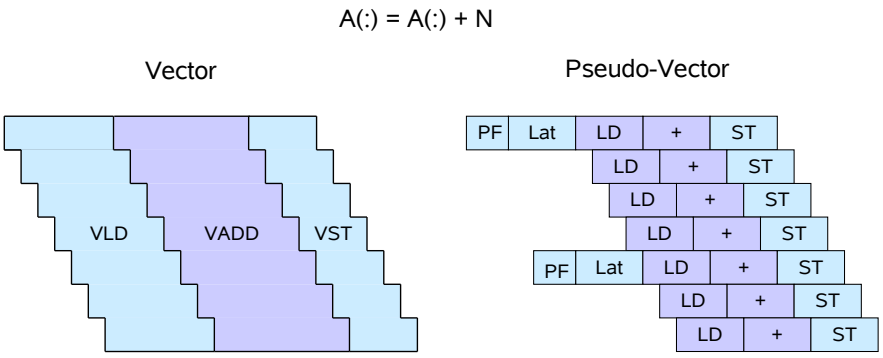

Figure 4.11: *This picture compares the addition of a constant to a vector on a vector machine (left) and on a pseudo-vectorization architecture (right). With its special memory interconnect of pipelined and interleaved memory access the vector computer can load vectorial data to its registers very efficiently. Making use of these large registers, the vector addition is processed in parallel by multiple arithmetic units. Subsequently, the resulting data vector is stored in memory again (Dowd and Severance, 1998). In case of pseudo-vectorization a software-assisted pre-fetch function is used to enable a hardware-based memory lookahead mechanism. In this way, waiting times between successive instructions are eliminated by pipelining data fetches from memory. In the CPU several arithmetic units process the vector addition in parallel. Finally, the results are stored to memory again. (Lanfear (2000))*

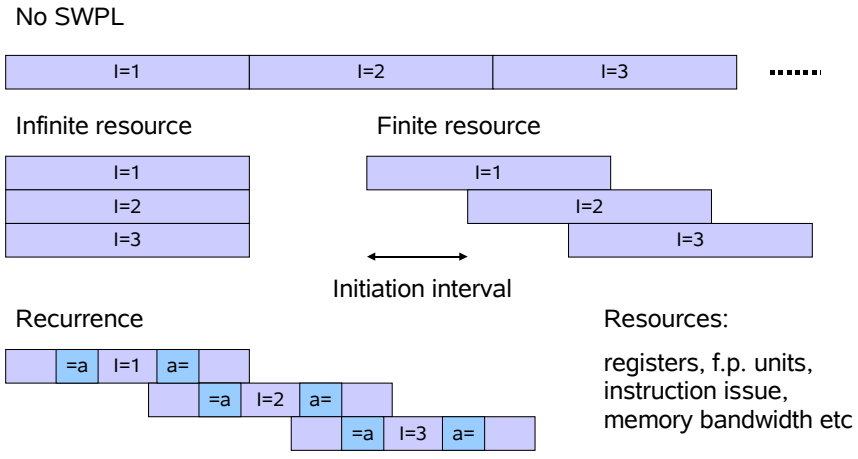

Figure 4.12: *Without software pipelining, each iteration of a loop has to be processed in serial and, only in principle, infinite resources (i.e., registers, arithmetic units, etc.) would allow all loop cycles to be conducted in one step as long as there is no data dependency between iteration steps. In reality, however, the loop cycles can be executed in a pipelined (partially overlapping) manner depending on the available resources and the data dependency between iteration steps (Lanfear (2000))*

	t			t+1
● fluid	*f*	*p(f), v(f)*	*f_eq(p,v)*	*f = coll(f,f_eq)*
● slip	*f = func1(f)*	*p(f), v(f)*	*f_eq(p,v)*	*f = coll(f,f_eq)*
● velocity	*f*	*p(f), v*	*f_eq(p,v)*	*f = f_eq*
● pressure	*f*	*p, v(f)*	*f_eq(p,v)*	*f = f_eq*
● wall	*f*			*f = func2(f)*
○ universal	*f = f·Σ●●● + func1(f) ·Σ●*	*p = p ·Σ ● + p(f) ·Σ ●●●● v = v ·Σ ● + v(f) ·Σ ●●●●*	*f_eq(p,v)*	*f = coll(f,f_eq) ·Σ ●● + f_eq ·Σ ●● + func2(f) ·Σ ●*

Figure 4.13: *This table shows in its left column various boundary conditions, while the three columns in the middle show whether the values of the distribution function f, the local macroscopic density $p(f)$, and the local macroscopic velocity $v(f)$ are directly available or whether they need to be computed at timestep t. The last column shows for which boundary condition 'collision' has to be computed or where only the equilibrium distributions (f_{eq}) have to be defined to get to timestep $t+1$. Using the row with the 'velocity' condition as an example, the table conveys that the distribution functions are available at time t and that the density p is a function of f, while the velocity v needs to be set to a certain value. In general, equilibrium distributions are functions of the local macroscopic density p and of the local macroscopic velocity v. In proceeding to timestep $t+1$, all distribution functions are set to the equilibrium distributions. Finally, to avoid conditional 'if' branches within loops over all grid points of a domain, a universal approach for all boundary conditions is shown in the last row. Depending on whether a boundary condition is set or not, a blending factor of 1 or 0 is introduced, respectively. These blending factors of each boundary condition are multiplied by the corresponding formulae for f, p, v, f_{eq} at time t and f at time $t+1$ to activate them if needed. To improve the clearness of presentation, the blending factors have been substituted by colored dots in the last row of the table.*

4.5 Porting and Optimizing the Solver for SGI Altix Systems

To prove the portability of the computational framework, *iFluids* has been ported to an SGI Altix Linux system. This was done on the Altix 3700 machine at Sara Computing Center in Amsterdam and the whole computational steering application was benchmarked on this system which is very much different from the Hitachi architecture. Experiences made during the porting process could directly be applied on the new Altix 4700 supercomputer at the Leibniz Computing Center in Munich.

Hardware Description

The SGI Altix 3700 machine at Sara Computing Center consists of 416 Intel Itanium2 CPUs (1.3 GHz, 3MB Cache) and hosts 832 GB of main memory. Each CPU has access to 2 GB of local memory, which can be accessed very fast. In addition, all CPUs of a node (theoretically up to 512 CPUs) can access the whole memory of this node via ccNUMA (cache-coherent Non Uniform Memory Access) links. However, memory access through ccNUMA is slower as compared to direct access of local memory [6]. The peak total performance of the Amsterdam SGI Altix 3700 is benchmarked as 2.2 TFlops/s.

Data Layout and Access

In contrast to the Hitachi SR8000 the performance on Altix systems can be increased further by a slightly modified data layout. Donath (2004) compared three layouts of data (`f[x][y][z][i]`,`f[i][x][y][z]`, and `f[x][i][y][z]`) and found `f[x][i][y][z]` performing best, because in the computation the density distributions labelled through i are located closer to each other. The same is true for the locations where the updated density distribution has to be copied to during the propagation.

Wilke et al. (2003); Donath (2004), in addition, suggest *grid merging* and *grid compression*. Instead of using two grids for the timesteps t and $t+1$ an interleaved grid is used to improve spatial locality (see Figure 4.14).

Grid compression, again, increases the spatial locality of memory access and saves main memory. Since for the propagation the computation only requires the direct neighbors, the idea is to shift the grid one unit in each direction. Therefore, one common grid can be used extended by some "ghost layers" (see Figure 4.15). The direction of the shift then determines the sequence in which the cells will need to be updated to avoid overwriting cell information which is still required.

[6] The ccNUMA link allows for data-transfer at a rate of 0.25 to 0.5 % compared to local memory access. Depending on how many router hops need to be taken the latency is increased (about 50 cycles per hop). The main problem of this architecture is a potential congestion of the ccNUMA link in pparallelprograms

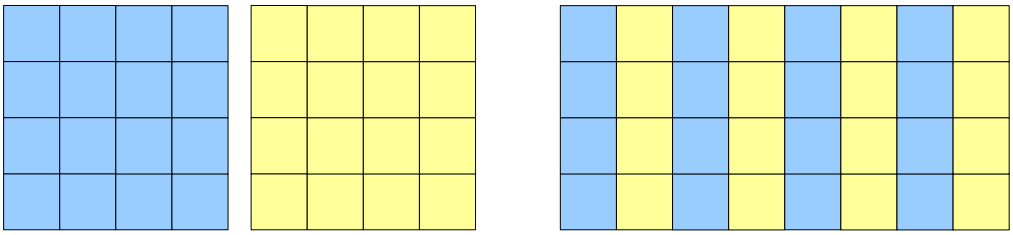

Figure 4.14: *A 2D example of grid merging is shown in this figure: To improve data locality in the propagation step, the two — originally separate — arrays for storing the distribution functions at timesteps t and $t + 1$ are stored as an interleaved array after merging.*

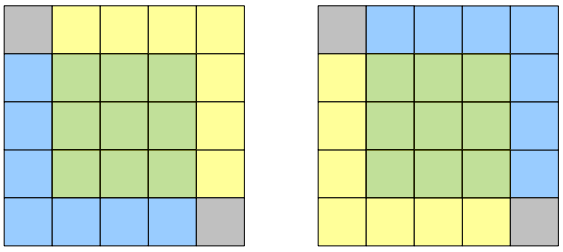

Figure 4.15: *Grid compression is used to further improve spatial locality of the data needed for collision and propagation, and to save memory. The array holding the distributions at timestep t (left) is colored blue, while the yellow array is the target for storing the distributions after the propagation. Since the two arrays are translated by one unit in each direction, no relevant data is overwritten. For the following timestep $t + 1$ (right) the shifted arrays need to be processes in a different order.*

Parallelization Strategy

The hybrid parallelization model as shown in Figure 4.5 was used to optimize *iFluids* on the Hitachi SR8000, combining MPI and COMPAS parallelization. On the SGI Altix only MPI parallelization has been applied and, hence, the shared-memory capabilities of this supercomputer are only exploited where SGI's vendor-optimized MPI library can make use of this architecture.

Besides the usual pitfalls one has to keep in mind when using OpenMP, using it *efficiently* on the SGI Altix NUMA system requires careful programming. To achieve good performance one has to obey the 'first-touch placement' rule when accessing memory or data. Since each processor has its own local memory, which can be accessed very fast, it is strongly recommended to place data that will mainly be used by one processor in its corresponding local memory. The first-touch placement 'pins' the data to that CPU's local memory which touches the data first (usually done during data initialization) (Wellein et al., 2006). Wellein et al. (2006) found good parallel efficiency for AMD Opteron NUMA architectures due to their separate paths to memory. Accordingly, on Intel Xeon multiprocessor systems the flat memory model causes a memory bottleneck. Bella et al. (2002) investigated parallelization of the LBM using OpenMP on an SGI Origin 3200 and found a good speed-up behavior.

Performance Comparison SGI Altix Systems versus Hitachi SR8000

Fortunately, many code optimizations that had been implemented with the Hitachi architecture in mind also performed well on the Altix right from the start, since many optimization aspects are also valid for this system. An example is the fusion of collision and propagation into one loop (Wellein et al., 2006; Wilke et al., 2003). Altix-specific optimizations can be activated via option flags within iFluids. The performance of the application on both machines is compared in Figure 4.16. The performance gain achieved by using the Altix 3700 as compared to the SR8000 lies at about 70%, while both codes show a good speed-up on their machine.

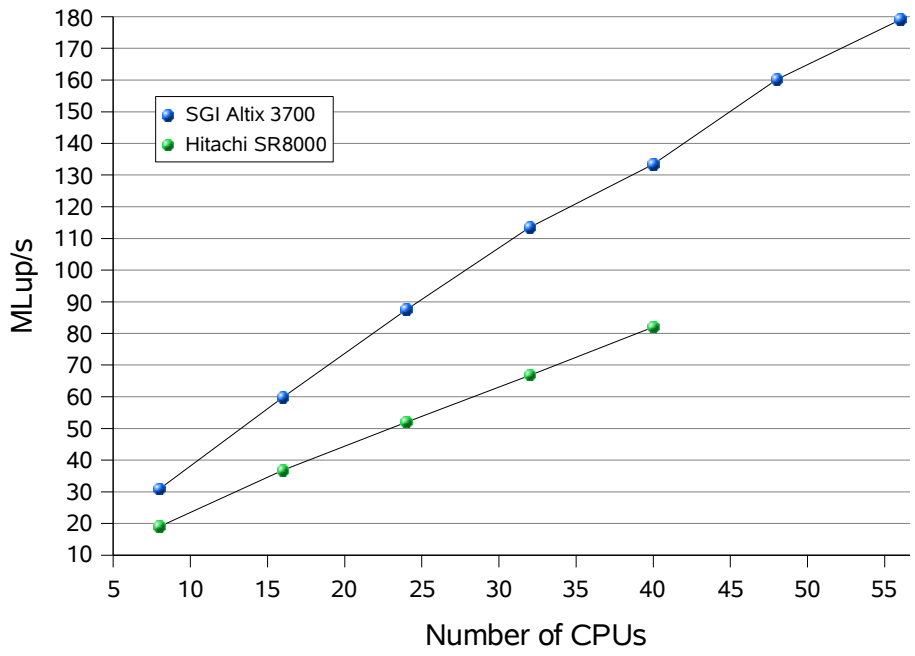

Figure 4.16: *This graph shows the performance of the Lattice-Boltzmann kernel running on Hitachi SR8000 (green) and SGI Altix 3700 (blue). Both codes show good speed-up behavior on the respective machine. The performance gain seen after porting the kernel to the Altix was about 70% when using 40 processors on both machines.*

Chapter 5

Interactive Data Exploration

After the development of a fast CFD solver, the next step towards a computational steering application is the implementation of on-the-fly visualization of data (online monitoring), which will be covered in this chapter. First, the term scientific visualization will be introduced, followed by a brief description of the supported hardware within the computational steering framework *iFluids* and the presentation of the visualization module that has been developed.

5.1 Scientific Visualization

Scientific visualization refers to visualization of scientific data sets. In this context we will now focus on the generation of visual representations from the results of scientific simulations in the field of fluid dynamics.

With increasing computational resources the resulting data sets are growing as well. Especially simulations in the field of CFD frequently produce huge amounts of data that have to be postprocessed. Modern visualization techniques provide powerful tools for data exploration.

According to Bellemann (2003) one has to differentiate between static and dynamic environments. Static environments are used for investigating time invariant data such as in classical postprocessing of precomputed data. In the case of dynamic environments an external process generates new data. For both cases interactive data exploration is a compulsory feature of modern visualization interfaces. It is important to note that in this context user interaction refers to the visualization process only and not to any computational steering interactions influencing a simulation. A prerequisite of achieving interactive data exploration is that the visualization is fast enough (i.e., at least 10 frames per second) without sacrificing important details. In addition, fast responses to user interactions are required to allow an accurate control of the visualization and to avoid uncertainty on the user's side.

A modern visualization environment should offer an *intuitive* means of interaction to permit a user to modify parameters controlling the presentation in order to be able to extract qualitative and quantitative information from the investigated data sets. Accordingly, the interaction methods should not require any ex-

planation or user guides to avoid a possibly long familiarization phase (Norman, 1988; Sanders and McCormick, 1993).

The first breakthrough fulfilling the above requirements is due to Bryson and Levit (1992) who set up a virtual wind tunnel. They developed a virtual-reality environment to explore precomputed unsteady flow fields in 3D (cf. Fig. 5.1). Their virtual-reality environment allowed a user to explore flow data similar to the way it is done in a real windtunnel by, e.g., placing smoke traces into the fluid to visualize streamlines. One important advantage of the virtual windtunnel was the close observation of the flow without disturbances due to measurement facilities as in a real-life tunnel.

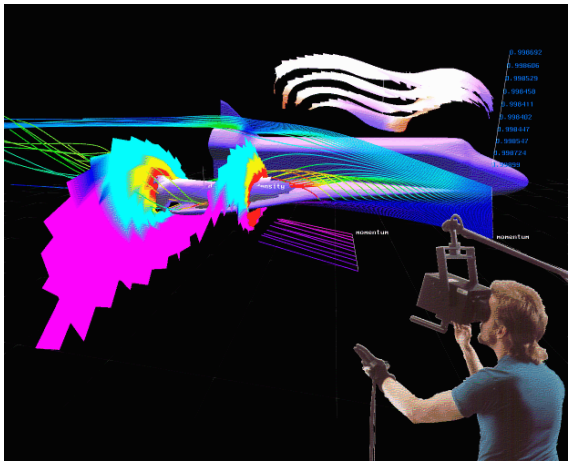

Figure 5.1: *The virtual wind tunnel: Flow visualization around a space shuttle within a virtual-reality environment. Within this virtual windtunnel the user wears a head-tracked stereo display, effectively displaying 3D information, and an instrumented glove for intuitive positioning of flow visualization tools. (taken from Bryson and Levit (1992))*

5.2 Visualization within *iFluids*

As found by Bryson and Levit (1992), immersive visualization is the best suited method for intuitive data exploration. Therefore, the computational steering framework *iFluids* has been designed to allow connections to several virtual environments. Standard displays such as used for workstation PCs and Laptops are also supported (see Fig. 5.2).

Virtual Environments

- **Desktop System:** In this case *iFluids* uses a conventional desktop monitor as a window to the virtual world. Traditionally, this is a 'flat' projection of 3D graphics of a data set. Through the use of a set of LCD shutter glasses, visualization can be extended to support stereoscopic viewing. This method

is called active-stereo as the display on the screen alternates between a left and right eye view of the 3D scene and, in accordance, the shutter glasses switch between opaque and transparent alternating for the right and left eye. The switching occurs fast enough to let the brain fuse the two views to a stereoscopic view.

- **Passive Stereo Projection:** A passive stereo backprojection system installed at the Chair for Bauinformatik (TU München) has also been used for the computational steering setup. It consists of a special-made screen in front of two projectors which are equipped with two orthogonally oriented polarization filters. One projector displays the view for the right eye and the other the corresponding view for the left eye. When using orthogonally oriented polarization filter glasses the two superimposed views from the projectors are filtered for the dedicated eye to receive the corresponding stereoscopic view.

- **Holobench:** The third integrated option for the interactive fluid simulation is offered by the holobench at the LRZ. This is a combination of two projection screens mounted at a right angle to form an L-shaped visualization workbench. Ideally, it is meant for creative teamwork for groups of 2 or 3 people. Again, with LCD shutter glasses a sterescopic image generates active stereo. Additionally, the master user's head is tracked and the view of the virtual world is adapted accordingly to the position of this user. This enhances the impression of 3D objects in front of the spectator.

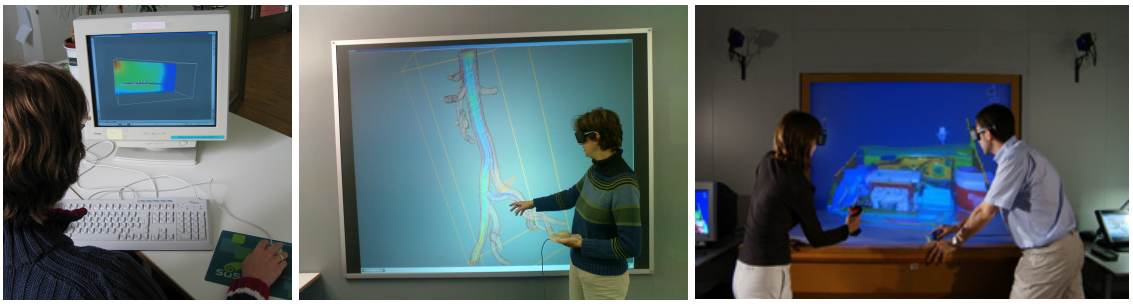

Figure 5.2: *Supported Environments: Within the standard desktop environment (left) conventional 3D graphics and stereoscopic views are supported. A large passive stereo backprojection screen (middle) offers more immersive data exploration. One step further into immersive environments can be achieved by using a 2-screen holobench (right, taken from ConceptCar (2007)) together with head tracking and a tracked input device for improved intuitive interaction features.*

Data Exploration

For a better understanding of the data and to assist the user's orientation in the virtual world data representation objects and CAD-generated geometry of the simulated scene are visualized at the same time. Occlusion of objects is counteracted by providing a mode with transparent visualization.

An intuitive exploration of a scene is enabled via three navigation modes. The most unconstrained is the 'fly mode', in which a user can freely move through the scene in all directions and the viewpoint is adjusted accordingly. The 'walk mode' is thought to support a realistic inspection of rooms and buildings in that the user's point of view is fixed to an adjustable z-level, whereas he can still freely move in x and y direction. In both modes the view representing the user's direction of gaze can be rotated freely so that observation from his current position is not otherwise constrained. The third navigation mode is the classical bird's eye view. Here the user's position is fixed and instead of modifying the user's point of view the scene is transformed by rotation, translation and zooming in and out. During the interactive data exploration the user can switch to any mode at any time, depending on his requirements.

For visualization of fluid data a series of standard techniques for visual analysis have been implemented:

- **Vector planes and fields** visualize the direction of the flow's velocity field at a location in space through an arrow and the velocity value determines it's length and color. Vector planes orthogonal to the room's axes can be inserted, translated and rotated interactively. Vector fields are displayed in a cuboid with interactively definable extensions.

- **Streamlines** are integrated paths along the velocity vectors of the flow field. The starting points of integration are inserted into the scene with the help of geometric 'seed objects' such as rectangles, lines, or circles. These representatives can be transformed arbitrarily and the number of starting points can be adjusted. The velocity is represented through the coloring of the streamline along its path.

- **Motion particles** are animated particles moving along the streamlines from the location where they were inserted into the scene. Inserting the particles is done analogously to the insertion of streamline starting points.

- **Isosurfaces** can be used for the evaluation of scalar quantities. An isosurface is a surface displaying all points in a data field with a certain value. Within *iFluids* an isosurface can be applied to pressure and temperature fields or velocity components. The value determining the surface can be adjusted continuously and the quantity visualized can be changed at any time. The color at a particular point on the isosurface stands for the corresponding magnitude of the displayed value.

- **Cross-sections** also display scalar values using a color map. They can be inserted and transformed in the same way as vector planes.

All data representation objects can be added to or removed from the scene at any time and as often as needed. Figure 5.3 shows some screenshots of example visualizations.

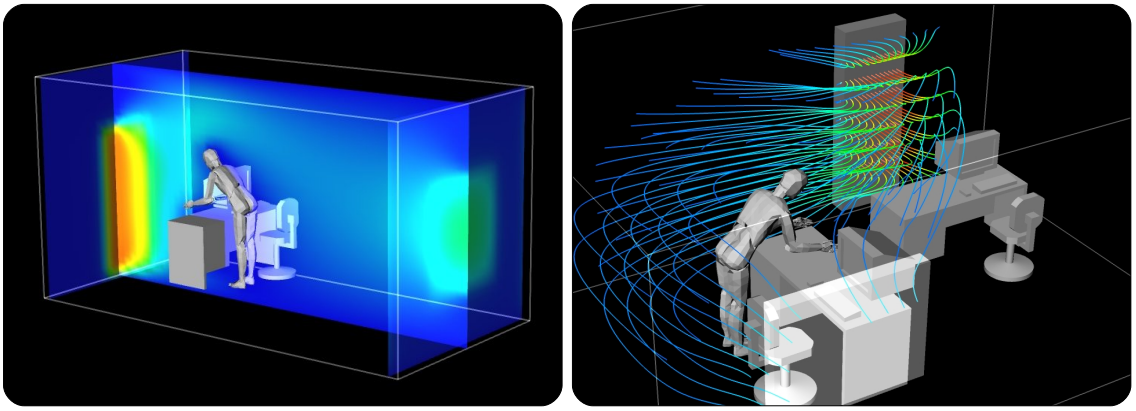

Figure 5.3: *Data exploration: On the left cutting planes depict the x-component of the velocity field, while the right screenshot shows streamlines following the fluid flow. Within both screenshots the coloring ranges from blue to red representing the minimum and maximum values, respectively.*

Besides the transformation all parameters of the visualization objects can be adjusted interactively to facilitate easy and comfortable exploration of the data, in particular, a context-based menu has been implemented for controlling the different functions. For all these interactions different kinds of input devices can be used or combined with each other. So far, conventional desktop keyboards and mice are supported, as are 3D Spacemice and a Wand device, which is used for visualization and navigation especially in virtual reality environments (see Figure 5.4).

Figure 5.4: *3D Spacemouse and the Polhemus 'Stylus' Wand*

Within the computational steering application the data is continuously varying over time (dynamic environment). Therefore, the visualization module needs to update the data fields automatically to allow for an intuitive visualization and

steering terminal. In the following the design of the visualization module will be presented.

Design of the Visualization Front-End

The visualization frontend has been implemented using Mercury's OpenInventor 3D scene graph libraries (Mercury Computer Systems, Inc., 2007b). To guarantee smooth *interactive* data exploration, a multithreaded viewer is used (see Figure 5.5). Multithreading is necessary to allow interaction with data representation objects while, concurrently, the data is processed for visualization in a separate thread. This is especially important for time-varying data sets, which prevail in computational steering applications.

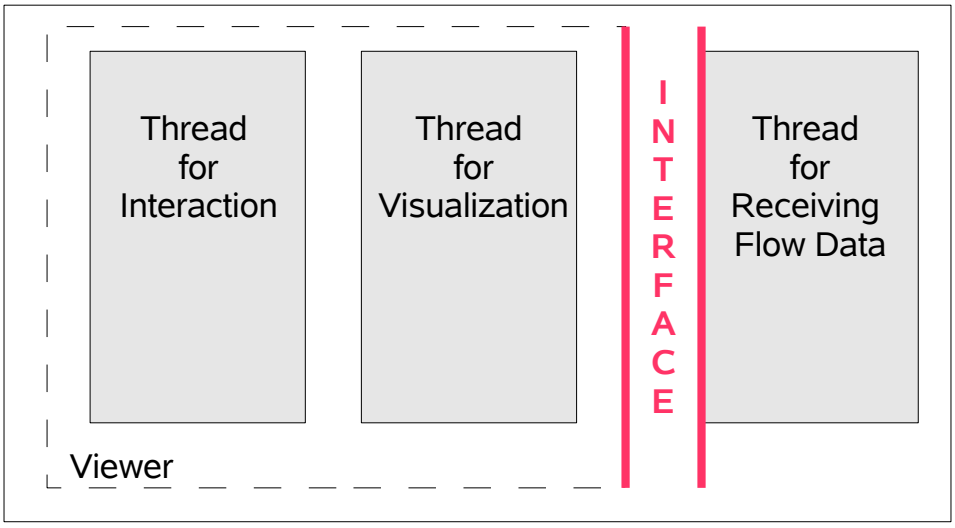

Figure 5.5: *Design of the visualization module: The multithreaded viewer permits concurrent postprocessing of data and user interactions. The visualization thread communicates with the thread receiving the external flow data via a special interface. An internal mechanism takes care of the automatic update of the data representation objects in the visualization.*

In addition, the multithreaded design needs to be extended with a mechanism to guarantee the automatic update of currently received external data. Therefore, the scene graph is built with special visualization nodes, which are connected to so-called *Property Objects*, to avoid copying data sets into these nodes. Through this connection the visualization objects listen on their property links and are automatically updated as soon as new data arrive. In contrast to the data representation objects in the scenegraph, these property classes work with pointers and are notified of changes via calls of an update routine.

The visualization front-end has been designed as a module with interfaces for data exchange and is therefore encapsulated from other structures. This, additionally, enables a connection of the visualization to other data services, as well as making the application ready for multi-client extensions (Borrmann et al., 2006).

Chapter 6

Interactive Problem Definition and Grid Generation

The previous chapters described the simulation kernel and the interactive data exploration of *iFluids*. Those are the central components of an online monitored simulation. To achieve real computational steering and thus allow the user to interact with the simulation during its execution, an additional steering environment is required. This chapter will therefore introduce the important aspects of steering in the context of a CFD simulation, which comprises basic, grid-independent steering options, the definition of boundary conditions as well as the online modification of the geometric model. Furthermore, the user interface of the interactive simulation will be briefly described. Since the incorporation of geometric modifications into the simulation model is central to enabling full interaction, the focus in the second part of this chapter will be put on a specially optimized grid generator.

6.1 Steering of Global Simulation Parameters

The simplest level of interaction with a simulation is to stop, pause or restart the simulation run. This can be extended rather easily to changes of global parameters and physical constants, e.g., the viscosity in the case of fluid simulations. At this point the interaction facilities of most other computational steering frameworks end. The main characteristic of these types of interaction is that their influence on the simulation is grid-independent. Some of the more sophisticated interaction types offered by *iFluids* are the possibility to activate turbulence modeling and to choose between several optimization strategies for the main computation routines so as to be able to adjust quickly and easily to the different strengths of the presently used hardware.

These basic interactions are accessible through a context-based 3D menu which can be used with stereoscopic visualization systems as well as with conventional desktop workstations. The supported input devices are conventional desktop mice with standard keyboards, the Polhemus Stylus wand, and 3D Spacemice, which have been described in Chapter 5.

6.2 Interacting with the Geometric Model

One of the key features of *iFluids* distinguishing it from other computational steering applications is the possibility of interactive modifications of the geometric setup and its boundary conditions during ongoing computation. The geometric model will now be introduced as a starting point of the description of this characteristic.

Geometric Model

The geometric model used in *iFluids* is based on boundary representation objects (Brep), stored in the STL stereolithography file format. This format contains a description of a triangulated surface, namely the coordinates of the vertices and the normal of each facet (wikipedia, 2007c). STL files can be obtained easily since this format is a standard export option in most CAD systems and, in addition, numerous tools allow converting most of the well-established formats such as 3DS, DXF, OBJ, IV, AM, and VRML into STL.

Figure 6.1: *Working place scenario stored in the STL file format: This figure shows triangulated surfaces stored in STL format. One file can contain several distinct objects, which can be loaded and stored as a group.*

An STL file is able to hold the description of several distinct objects and as such permits grouping of objects. For example, an office working place can be described by a group of objects comprising a desk, a chair, a telephone, a desktop PC etc. (see Fig. 6.1). For use within *iFluids* (and between *iFluids* sessions) the standard STL file format can be supplemented by several fluid mechanical attributes without losing its compatibility to standard STL readers or import software. This has been achieved through the use of special 'magic keyword' comments which are only evaluated by *iFluids* and otherwise ignored. With the functionalities of

grouping objects and attaching corresponding fluid mechanical attributes it is possible to define a complete scenario of a room with doors, windows, ventilation facilities and furniture, which can be loaded at application start to generate the initial geometry setup and boundary conditions. This shortens the interactive initialization phase as compared to starting from scratch without any initial start up description, where the hardcoded startup is a simple cuboid room with a door and a window as inlet and outlet, respectively.

Offline Preprocessor *iFluidsPre*

To extend standard STL files with fluid mechanical attributes and to be able to group and split objects for the initial starting scene, the preprocessing tool *iFluidsPre* has been developed (Kollinger, 2007). *iFluidsPre* has been implemented as a module of the general-purpose 3D and VR visualization system Amira (Advanced 3D Visualization and Volume Modeling (Mercury Computer Systems, Inc., 2007a)).

Within *iFluidsPre* a complete scene of a fluid simulation can be constructed. By loading STL files with the geometric configuration for an indoor simulation of a room, the fluid domain and its boundary condition is defined through the room's walls, complemented by more details such as the basic furnishing, ventilation facilities or other inner fluid obstacles.

As already mentioned, the room's geometric configuration is based on object groups stored in a single STL file. The objects building up the group in this context will be referred to as *subobjects* in the following. When double clicking on an object (group), *iFluidsPre* enters the so-called 'Subobject Mode' of such a group and displays the individual subobjects in different colors as shown in Fig. 6.2.

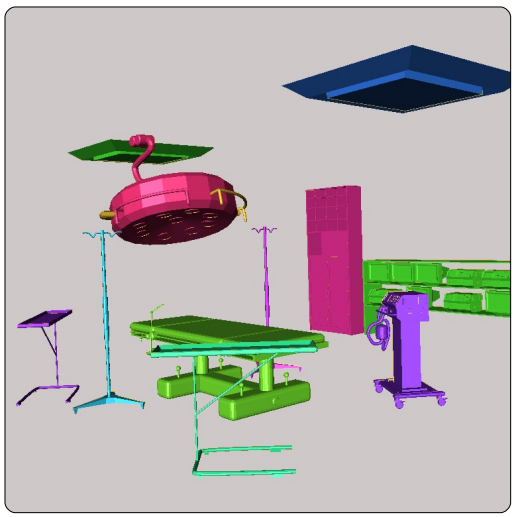

Figure 6.2: *A composition of facilities and devices needed in a surgery room is shown. The different colors indicate the individual subobjects of this grouped set of geometrical representations.*

iFluidsPre allows the user to group several objects from different STL files and to store this group of objects in a new file. Within existing objects or subobjects the user can furthermore define a group of facets which can be stored as a new subobject. Besides defining new subobjects, it is also possible to merge several subobjects into one object or to delete unneeded subobjects or single facets.

As a module of Amira *iFluidsPre*, in addition, offers Amira's full functionality with regard to modifying the geometry of facet surfaces. This comprises coarsening and refining areas of a facet mesh, removing selected triangles, as well as modifying the orientation of their normals or coordinates.

The main objective of *iFluidsPre*, however, is the assistance in defining fluid mechanical attributes for whole objects or groups of facets. Therefore, groups of facets can be defined and tagged by boundary conditions and material parameters (see Figure 6.3). Currently, boundary conditions of the following types can be chosen (one per group):

- At boundaries a **no-slip** condition states that the fluid's velocity vanishes along the boundary. Following He et al. (1997) this has been implemented as a bounce-back condition in the Lattice Boltzmann solver of *iFluids*.
- At frictionless boundaries a **slip** condition is implemented, which does not influence the velocity along the boundary but sets the velocity orthogonal to it to zero.
- At flow inlets as given, for example, at ventilation facilities, a **velocity** condition can be set. To determine the correct value for the Lattice Boltzmann Model, similarity considerations have to be taken into account, e.g., by using the Reynolds number (see Equation (3.31)).
- It is also possible to set a **pressure** condition at a boundary. In case the boundary face is orthogonal to a coordinate axis, the pressure condition can be enhanced by determining missing distributions following Zou and He (1997).
- In addition, a **temperature** and a **temperature gradient** condition can be attached to the surfaces of the investigated model to support currently ongoing developments of *iFluids* which will consider thermal phenomena for indoor fluid simulations and comfort studies (van Treeck et al., 2007).
- To be prepared for future extensions stubs for a **user defined** condition has also been implemented.

Combinations of boundary conditions are naturally supported, however, the user himself must take care to limit himself to meaningful combinations. For example one should consider that the combination of pressure and velocity cannot be set at the same boundary without already knowing the solution of the problem (Tölke, 2001). Furthermore, mass conservation must be guaranteed if only velocity conditions are used. Besides the boundary conditions already mentioned, further attributes and material parameters such as the specific heat or the surface coefficient of heat transfer can be set as well .

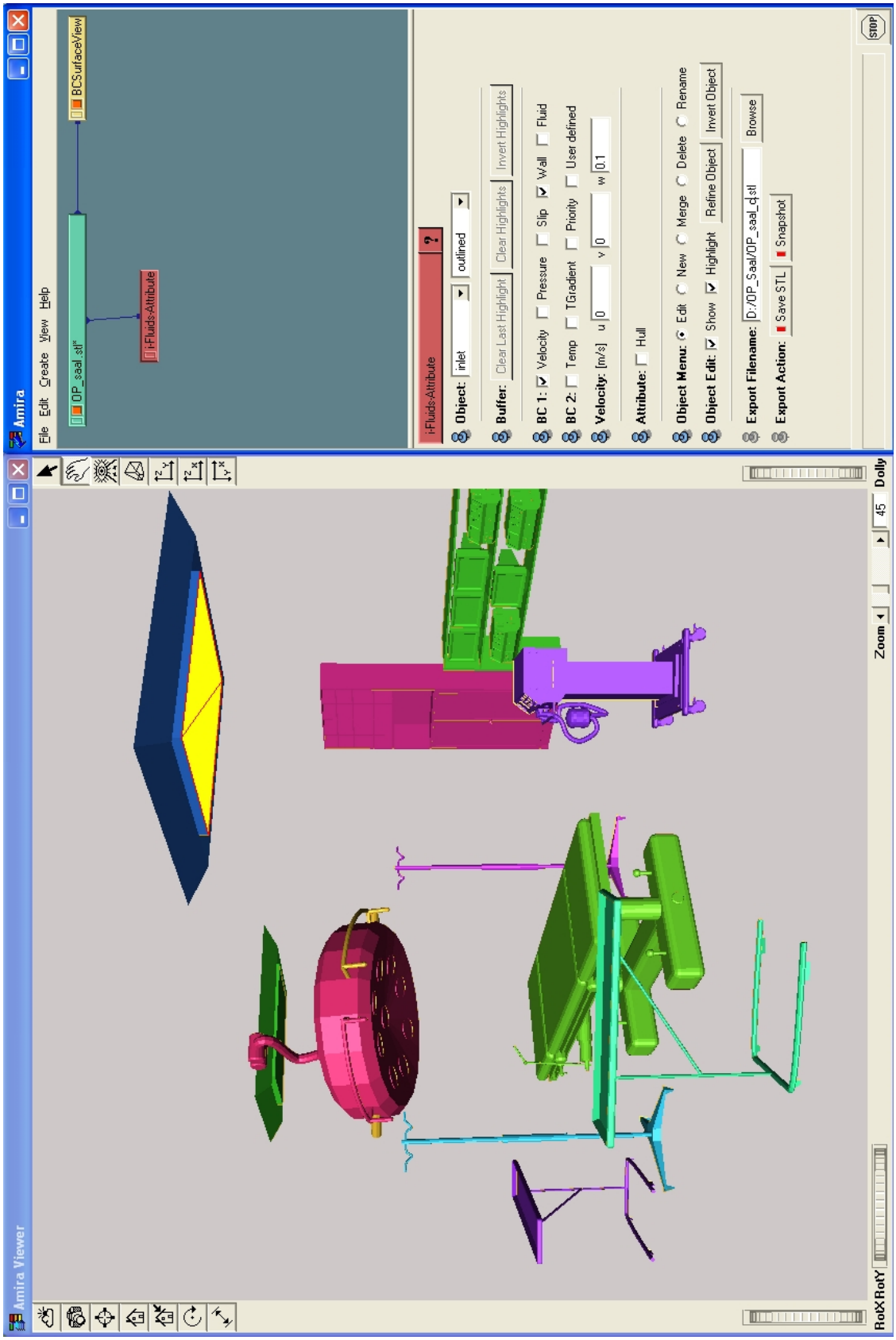

Figure 6.3: *This figure shows the embedding of the iFluidsPre module into Amira. On the left an object set is displayed in a view where its subobjects are represented in different colors. The currently selected subobject (the yellow inlet of ventilation installed at the ceiling) is marked with facets outlined in red. On the right the current settings of the boundary conditions are shown and can be adjusted according to the user's wishes.*

Interactive Geometry Modifications

As mentioned above the key feature of *iFluids* is the possibility to modify the geometric model and to define or change associated boundary conditions during a running simulation. Concerning the interactions with the geometry itself, additional objects can be loaded (i.e., inserted) from additional STL files, which may optionally be prepared with *iFluidsPre*. The context-based 3D menu supports the loading process by displaying previews of the contents of the STL files within the currently selected directory. Objects or whole groups of objects loaded to the scene can be translated, scaled, distorted or rotated within the scene, or they can be added to or removed from it (see Figure 6.4). The same modifications can be done within object groups, i.e. subobjects can be transformed relative to other subobjects of its group, or can be deleted.

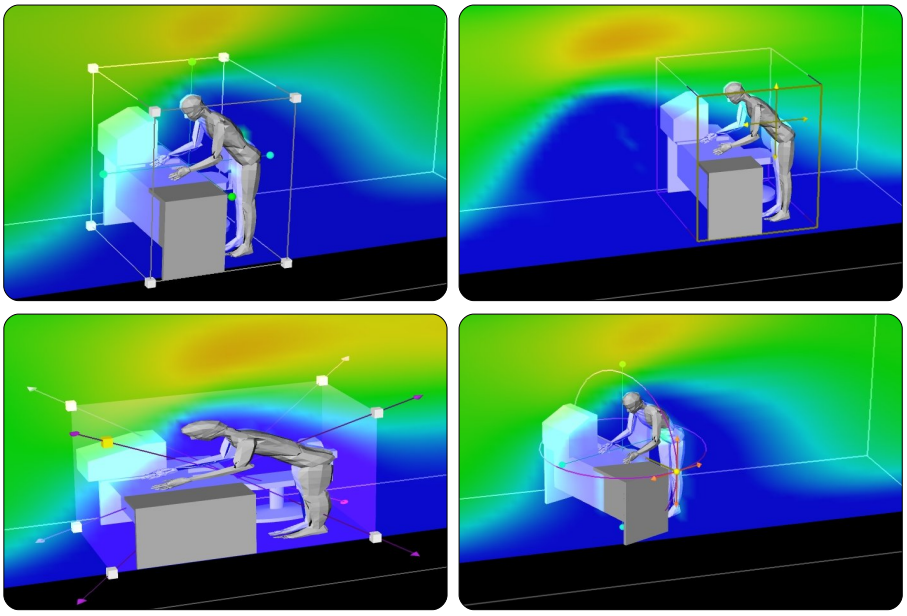

Figure 6.4: *Interaction with the geometric model: The upper left screenshot shows the original object group. Its selection is represented by displaying a bounding box with draggers which enable control of various transformations. In the upper right screenshot a translation in the x-z-plane is shown. In the lower left the object group is distorted in several directions by using the scaling draggers, while the lower right picture demonstrates a rotation transformation.*

Such interactions with the geometry are realized either by using OpenInventor draggers (see Figure 6.4) for all transformations with a conventional mouse or 3D Wand, or by using a 3D Spacemouse to access the objects directly for all possible transformations.

These modifications are immediately and automatically incorporated into the running simulation and the results of the simulation kernel adapt to the updated configuration, accordingly.

Interactive Definition of Boundary Conditions

Besides the modifications of the geometrical layouts a central aspect within the computational steering framework *iFluids* is how boundary conditions are defined or adjusted during a running simulation. The scene can be modified by rearranging inner objects with 'regions' carrying certain boundary conditions, e.g. windows, doors or ventilation inlets in an office room.

The confining surfaces of the simulation domain can be marked with the attribute 'hull' and will then be displayed in a transparent manner whenever they would cover the user's view onto inner parts of the scene as shown in Figure 6.5. As already mentioned, subobjects of a selected object are colored differently and, in addition, an informative pop-up message conveying the current properties of the picked subobject is displayed (see also Figure 6.5).

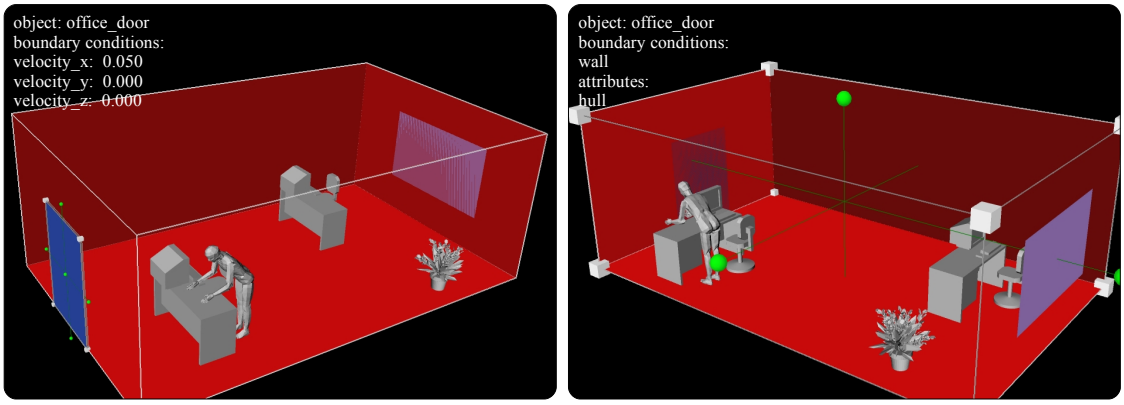

Figure 6.5: *This figure shows a simple setup of an office room. The boundaries of the room are defined through walls, a door and a window, which are assembled into one group of subobjects. In addition, the walls have been marked with the 'hull' attribute, which causes the display of only those parts of the room confinement, which do not hide other parts of the scene. This effect is best observed by comparing two different views of the same scene. Furthermore, the different colors of the subobjects are shown as well as the properties information of either the door (left screenshot) or the walls (right screenshot).*

Frequently, boundary conditions are represented by geometries coplanar to faces of the basic geometry, e.g. windows in a wall. This modeling shortcut is made possible through assigning a priority level to these boundary conditions which can - so to speak - 'overwrite' other settings with lower priority. However, the coplanarity of the facets involved usually makes it difficult for them to be selected unambiguously. Therefore, 'Explosion Mode' can be activated, which temporarily moves apart or explodes all coplanar faces with respect to the object's center and in this manner allows an easy selection as shown in Figure 6.6.

To set new boundary conditions or to modify existing ones, dialogs have been implemented, which can be displayed in either 2D, or 3D for either normal desktop environments, or environments offering stereoscopic views. The dialogs offer

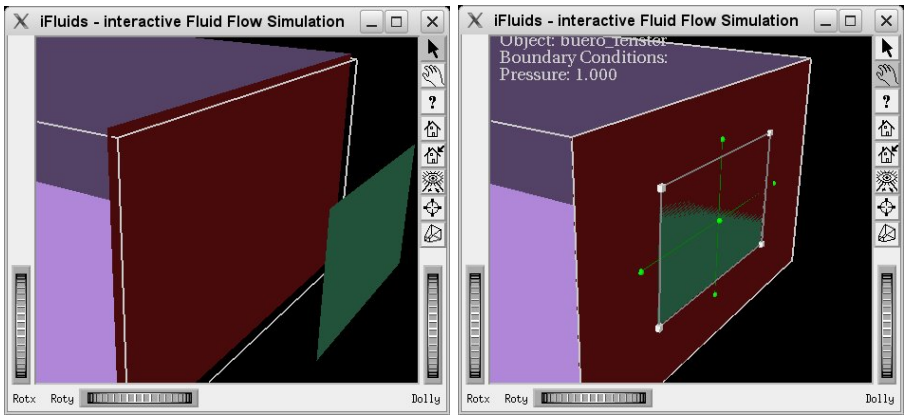

Figure 6.6: *Overlapping planes: These screenshots show exploded planes to simplify selection (left) and the correctly selected but partly hidden subobject (right).*

means of naming the boundary conditions, of choosing their type, and of setting the corresponding values. In addition, the above-described hull attribute can be set, and for overlapping boundary conditions a priority can be defined to specify which condition should overwrite others (cf. Fig. 6.7).

In comparison to the offline preprocessor *iFluidsPre* the only unsupported functions are coarsening or refining the faceted surface or changing the orientation of the triangle normals. Thelatter rely on Amira-internal calls and would have to be separately implemented.

6.3 3D User Interface

As already mentioned in the previous chapter (see Chapter 5), it is advantageous for computational steering applications to be run in a virtual-reality environment. To support an intuitive steering of the simulation in virtual reality a specially adapted user interface is required. Therefore, an immersive and intuitive 3D user interface avoiding classical 2D interaction has been developed. It serves as a context-controlled 3D console assisting a virtual fluid-flow experiment while still preserving the possibility to perform computational steering of the application via a standard 2D workstation mouse and keyboard.

To be able to stick to simple (and possibly also single) interaction devices such as mouse or wand, two major modes of operation have been introduced: The first is called 'direct mode' and provides the possibility of direct interaction with and navigation within the scene by using the standard input devices Wand, Spacemouse, and desktop mouse. In the second mode ('menu mode') a menu panel is displayed in the lower part of the viewport giving access to a variety of control possibilities.

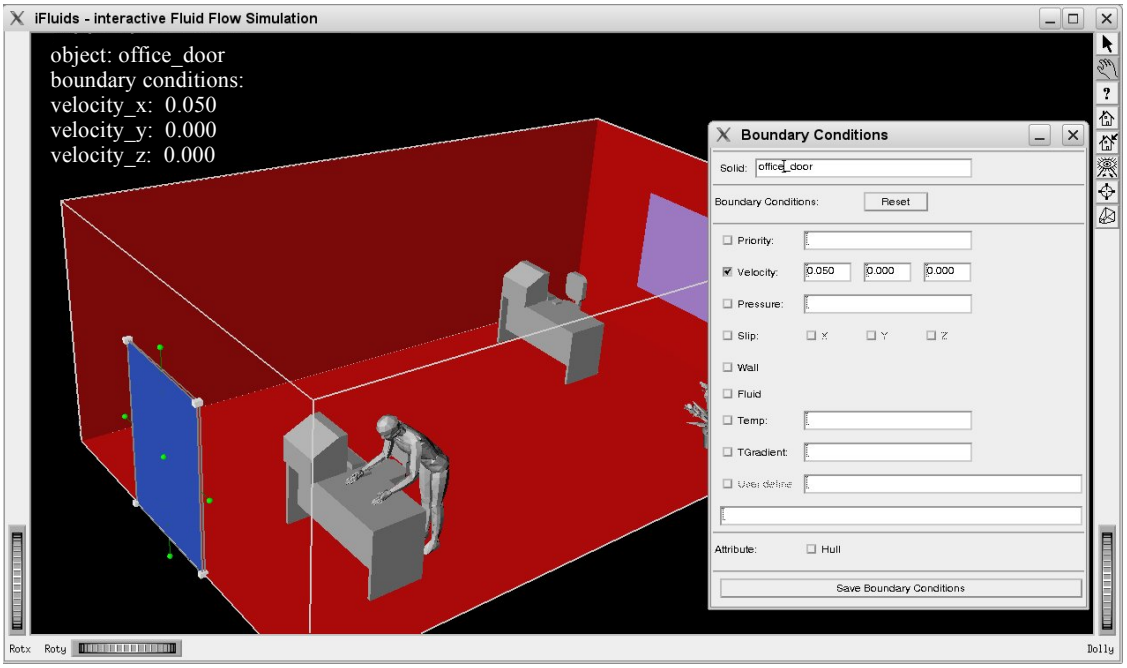

Figure 6.7: *This figure shows the dialog for setting or modifying boundary conditions of a selected object or subobject.*

Direct Mode

The application is started in 'direct mode', where the user - again - can choose between two navigation and two selection modes. To explore the scene, he may fly through or examine it from a fixed point of view, for example, by rotating the corresponding 3D model. The first selection mode allows to choose between the different objects representing the configuration of the simulation, i.e. fluid obstacles and boundary conditions. In the second mode the user can switch the selection between the data visualization objects i.e., cutting planes, streamlines, vector planes, and isosurfaces. Once an object has been selected, it may be moved to a different position. Depending on the current interaction (either navigating within the scene, or selecting data visualization objects or simulation objects, obstacles or boundary conditions), a context-based menu can be invoked, which appears with appropriate options.

Menu Mode

As already mentioned, there are three different menu categories according to the current context. The first one ('Scene Menu') is called while exploring the scene in navigation mode. This menu offers the insertion of new fluid obstacles, new boundary conditions, data representations, or a legend as well as the modification of global fluid flow parameters.

When adding a new obstacle, for example, the menu allows browsing through the file system to load the geometric description of a CAD-generated object. The

file browser makes use of the capabilities of the 3D visualization in that it shows a small rotating preview of the currently browsed object.

A context-based menu offers the advantage that the user can navigate directly through the menu without having to search for the respective submenu. This allows faster access and a more intuitive way of working as compared to user interfaces with deep menu trees. Furthermore, the menu information and graphical representation is reduced to its essential minimum to prevent annoying occlusions of the scene. If the selection of an option calls another menu branch the precedent menu will be replaced with the new one. Since the menu can be activated or left at any time we decided it to be displayed view-fixed regardless of the users current point of view, which may change during navigation. As shown in Figure 6.8, the menu has been arranged in front of the user similar to a console or panel. This preserves the view onto the scene even while controlling the menu in a fully immersive manner.

Another helpful detail is the highlighting of the currently selected menu item to give the user feedback which option would be selected in the menu. The menu options can be selected either by using the arrow keys on the keyboard, rotations of the space mouse knob, or by pointing at and clicking on them with a wand or the desktop mouse.

For complex settings as is the case when defining boundary conditions, the 3D menu switches to a 2D shape (also view-fixed) to be able to see all settings options at one glance in an efficient way (see Fig. 6.7).

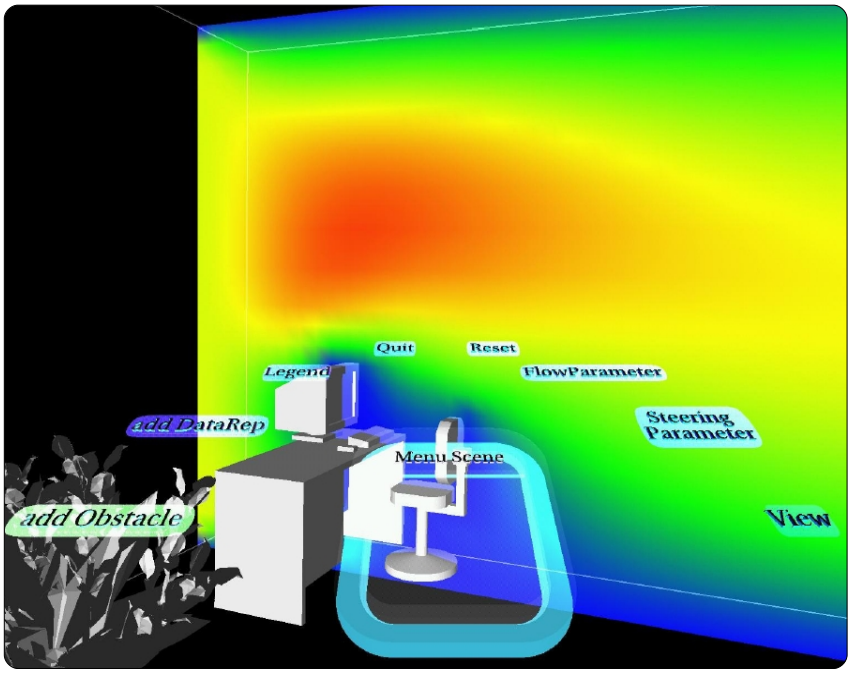

Figure 6.8: *This screen capture shows the 'Scene Menu' offering the user options to add obstacles or data representations, to control flow parameters and other configuration options of the simulation.*

6.4 Grid Generation

As already mentioned, a key-feature of *iFluids* making it stand out from other computational steering applications is the possibility to change the geometry and its associated boundary conditions during a simulation. The few applications which are also capable of handling interaction with the geometry are restricted to predefined and parameterized objects. Within *iFluids* arbitrary geometry can be loaded at any time into the currently investigated simulation scene without particular preparations and can be freely manipulated. This is due to a powerful grid generator which maps the geometry onto the computational grid of the Lattice Boltzmann simulation at comparatively little computational costs. In addition, the lattice sites of the computational grid contain object-specific information about the boundary conditions needed for the fluid simulation.

Voxelization

The process of converting the surface representation of a geometric object or scene to a 3D volume discretization on a grid of a given resolution is referred to as *voxelization*. The elementary unit volume within such a grid is called a *voxel* in analogy to the *pixel* in 2D-scanline conversion for raster graphics displays. Depending on the grid resolution and type of geometry, the voxelized object most often is not an equivalent representation of the original but only an approximation.

Even so, there are several fields of application, in which a volume-based representation of an object is necessary or advantageous. For example, voxelization methods have been used for volume visualization (Jones, 1996)), radiosity and ray tracing (Krumhauer et al., 1999), volumetric model repair (Kolb and John, 2001), and collision detection (Gibson, 1995). Furthermore, measurements in medical, physical, or engineering applications often result in volumetric data. Finally, certain types of numerical simulation are computed on Cartesian grids requiring the transformation of point or surface data into a volume representation.

For fluid simulations based on the Lattice-Boltzmann method Cartesian grids are used quite typically. The handling of complex CAD geometry data is thus comparatively simple, but 'brute-force' voxelization of a scene with several 10 to 100 thousands of surface describing polygons remains a computation-intensive task. It is therefore worthwhile to try to speed up the voxelization process by applying a space-partitioning algorithm to generate a volume representation via an *octree*. An octree is a hierarchical data structure with one root node and — depending on a certain criterion — with either eight or no child nodes. The latter may themselves be father nodes to follow-up trees or end the tree as *leaves* if without children (see Fig. 6.9).

An octree is especially well suited for space partitioning problems in 3D space since it is obvious to identify the eight children of the root node with the eight cartesian octants of an object's or a scene's bounding cube. Subdividing the octants which still contain parts of the geometry can be repeated a specified number

of times, whereby the number of recursions determines the level of refinement of the octree (Fig. 6.9).

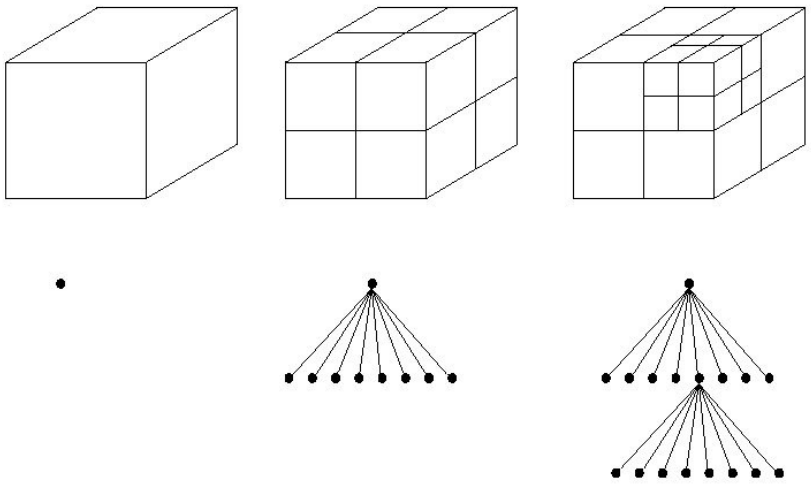

Figure 6.9: *This scheme shows two refinement steps of how to get from a single root-node octant (left) to a level 2 octree (right).*

The voxelization algorithms used in *iFluids* work on triangulated surfaces as input, which most CAD systems offer as a standard export option. In *iFluid's* grid generation module the common STL stereolithography file format (wikipedia, 2007c) is supported. For a given surface the STL file contains a list of its triangles vertex coordinates and normal vector.

There are a series of approaches to voxelization, the three most relevant methods have been presented, e.g., by Stolte and Kaufman (2001), Haumont and Warzee (2002), and Mundani (2006). The method of grid generation applied within the computational steering framework *iFluids* is related to these in so far as surfaces are converted into a volumetric representation using octree data structures. Although Stolte and Kaufman (2001) also concentrate on a fast conversion, their ansatz is limited to objects described analytically through an implicit definition. The other study by Haumont and Warzee (2002) uses polygons as input, however, their focus lies on model repair, visibility determination, collision detection, and complete interior/exterior classification of all voxels with very high computational cost. This leads to a voxelization process too complex for interactive use. The third method (Mundani, 2006) converts surfaces into an octree description using half space partitioning. This algorithm relies on convex objects defined by closed polygonal surfaces and, accordingly, is very efficient for basic geometrical shapes. *iFluids* offers two methods of grid generation, one for creating a uniform Cartesian grid, the other to produce hierarchical Cartesian grids.

Uniform Cartesian Grids

The Lattice-Boltzman solver (described in 3) currently used in *iFluids* is based on a uniform Cartesian grid. Accordingly, the grid generation process starts with an empty grid without any geometry data or boundary conditions. To map the geometrical data with its corresponding boundary condition information efficiently, a root octant is computed for each triangle. Depending on the size and orientation of the facet, these root octants are usually rather small. The size of the root octant and the grid resolution also defines the level of refinement needed. It can be determined through

$$oct_{level} = \lceil log_2 lenght_{max} \rceil + 1, \tag{6.1}$$

where oct_{level} is the maximum level of refinement in the octree and $lenght_{max}$ represents the maximum length of the bounding box of a facet in grid units. Note that this level of refinement creates sub-grid leaves as shown in Figure 6.10 to improve the geometry's approximation.

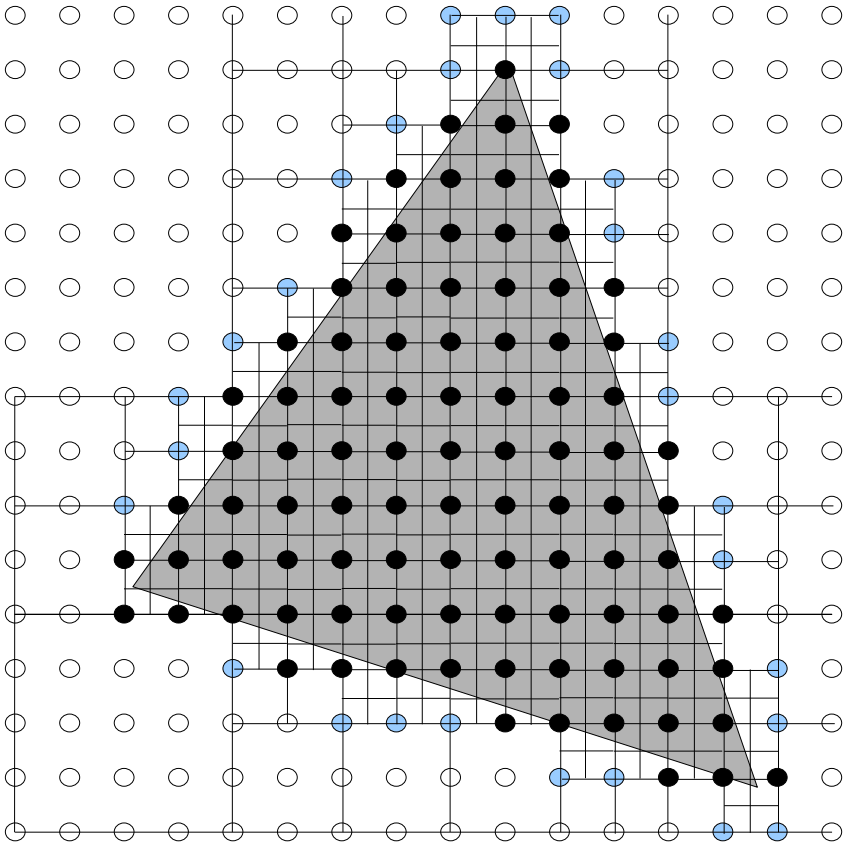

Figure 6.10: *Recursive refinement for a 2D example: The black grid points are set to approximate the triangle while the blue grid points indicate which points would have been set additionally without the sub-grid leaves created by the extra refinement.*

Figure 6.11 shows the octree structure for the described algorithm. For each triangle an individual octree has been created and recursively been refined.

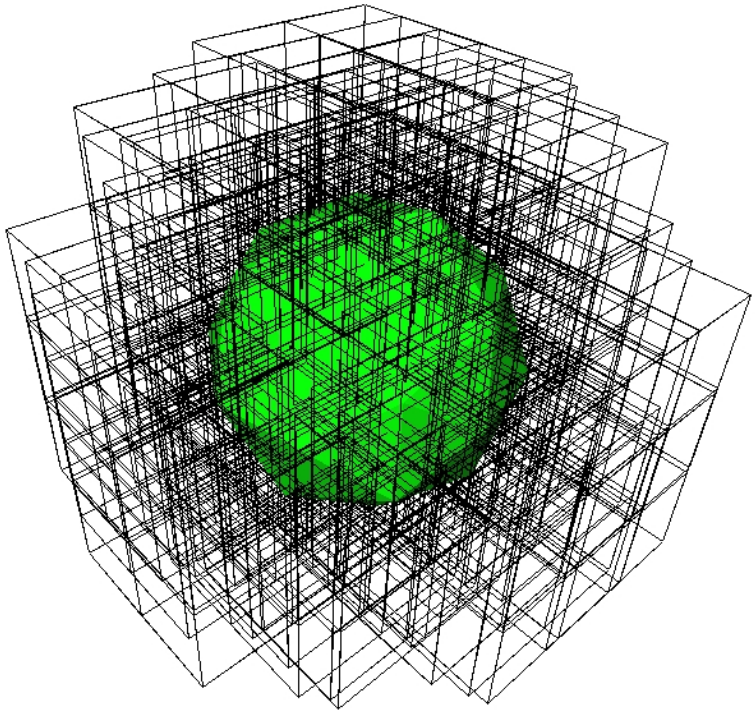

Figure 6.11: *In this example a Pentakis dodecahedron has been voxelized on a uniform Cartesian grid. For each triangle the generated root octants with their tree structure with 64x64x64 grid resolution are displayed.*

Hierarchical Cartesian Grids

Lattice-Boltzmann solvers can also be performed on hierarchical grids as proposed by (Crouse et al., 2003; Tölke et al., 2006; Thürey, 2007). An important option accessible with this kind of grid is the simple realization of patches of higher resolution around critical parts of the geometry. An extreme variant of this can be used for blood flow simulations as proposed in Götz (2006). The octree ansatz can be utilized to identify regions around the geometry (the artery with the interior flow) for setting up the compute cells, while the remaining grid outside these boxes is 'discarded'. To also support such solvers, a modified algorithm has been developed. Here, the algorithm starts with one root octant for the whole simulation scene, which is refined to the same level of refinement as obtained by Equ. 6.1. In contrast to the algorithm described above, all triangles have to be tested for the octants and a list of triangle 'candidates' intersecting the current octant will be given to the next generation of child octants. In addition to the level of refinement, a second criterion to stop the refinement of the current octant, then, is an empty candidate list. Since the triangles have to be tested several times for

intersection with the octants (although it may turn out that they have to be set by another part of the octree), this algorithm is slower than the above-described version for uniform Cartesian grids.

Figure 6.12 shows the octree structure, serving as the basis for the hierarchical grid. To obtain a grid which can be used for a Lattice-Boltzmann simulation, the octree constructed through this method needs to be smoothed, i.e., the level of refinement of neighboring cells must not differ more than a given limit (often required 1).

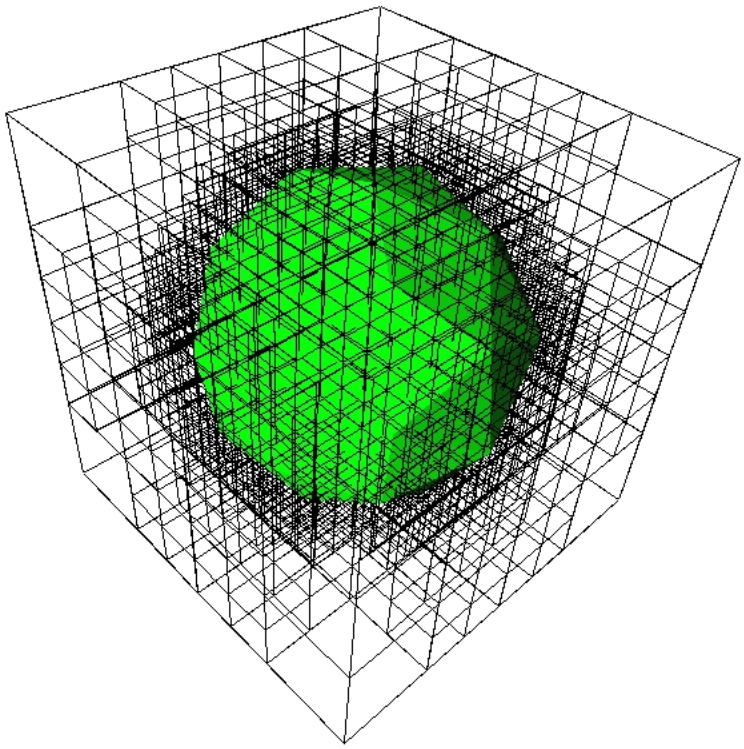

Figure 6.12: *Again, a Pentakis dodecahedron has been voxelized on a 64x64x64 grid. In contrast to Fig. 6.11 one root octant has been created for the whole object and was then recursively refined. The octree obtained can be used as a basis for a hierarchical Cartesian computational grid.*

Optimizations and Performance Evaluation

In both algorithms described above, the main computation is the test of whether a triangle intersects an octant, lies completely in its interior, or outside. For this intersection test a highly optimized routine based on code developed by Akenine-Moeller (2001) has been used.

This test routine relies on the separating axis' theorem (Gottschalk et al., 1996; Möller and Haines, 1999; Eberly, 2000) which states that two convex polyhedra, A and B are disjoint if they can be separated along either an axis parallel to a normal of a face of A or B, or along an axis formed from the cross product of an

edge of A and an edge of B. To simplify these tests, the box and the triangle are transformed in a way that the center of the box coincides with the origin and the faces are axis-aligned.

After the initial transformation, three tests are performed:

- Test the minimal bounding box of the triangle against the octant: Here it is checked whether the minimum and maximum coordinate-components in x-, y-, and z-direction are located outside the octant's minima and maxima or not.
- Test the intersection of the triangle's plane against the octant: Here the plane of the triangle, defined by one vertex and the triangle normal, is tested against the octant according to Möller and Haines (1999) and Haines and Wallace (1994).
- The last test computes the nine cross products of all combinations of the edge vectors of the triangle and the normals of the octant. By projecting the coordinates of the triangle and the octant onto it, it is determined, whether the projected triangle lies outside the projected octant or not.

When all tests are passed, the triangle intersects the octant and the next iteration of refinement can be started. The above tests can be implemented very efficiently (Akenine-Moeller, 2007). For the special case of *iFluids* the algorithm can be further optimized, since the normal of a triangle is already known and the octants are always axis-aligned boxes. Furthermore, vectors computed once for a triangle can be stored temporarily and reused during the following refinement steps.

Compared to the fluid computations, the voxelization of even complex scenes with high grid resolutions takes only a comparatively short time (typically on the order of a second for 256^3 voxels and 10^6 facets). Additionally, voxelization is only required on the occasion of geometry or boundary condition changes due to user interaction. Still, one might want to speed up this process even more and therefore, after the optimization considerations regarding performance of the grid generator on a single processor, the time consumption of the voxelization process can be reduced through parallelization.

Regarding the technical realization of the implementation one can choose between several ways of parallelization such as the MPI message passing library or OpenMP compiler directives as examples of distributed or shared-memory approaches. Since parallelization via MPI always requires a certain effort of adaptation of the source code's structure and, especially on shared-memory architectures, tends to introduce additional communication overhead[1], plain shared-memory communication using the OpenMP programming paradigm was chosen on both machines, the Hitachi SR8000 and the SGI Altix 4700.

[1]If not perfectly optimized ('zero-copy' transfers) MPI communication is slowed down through additional buffer copying as compared to direct shared-memory communication (Gropp and Thakur, 2005)

The parallelization strategy itself firstly depends on whether a uniform Cartesian or a hierarchical Cartesian grid is used. Regarding uniform Cartesian grids, the different threads perform their work on the same, shared grid points and the triangles are distributed among them. In contrast, for the hierarchical Cartesian grid the triangles are shared and the grid is distributed to the threads. With respect to the former approach one has to keep in mind that multiple processes have to write onto the same grid (necessarily in an ordered manner), while the latter variant requires only simultaneous read access. In both cases data placement needs to be taken into account to minimize memory access penalties.

To evaluate the voxelization performance the 'Dragon' test geometry has been used which is quite popular in this context. The respective geometry is shown in Fig 6.13 together with its voxel approximation computed by *iFluids'* grid generator.

The performance of the two algorithms described above has been compared on an AMD Dual Opteron 2.4 GHz system for two different grid resolutions and varying numbers of triangles approximating the Dragon. The single processor voxelization times of both methods are shown in Fig 6.14. Of course, the computation time of the voxel representation depends on the one hand on the number of triangles constituting the geometric object and, on the other hand, on the grid's fineness.

Finally, the first method has also been benchmarked in a parallelized version using two CPUs on the Opteron. Fig 6.15 compares the voxelization times when using one or two CPUs for the same test case as in Fig 6.14.

The performance evaluation reveals overwhelming results particularly for the voxelization on uniform Cartesian grids. This enables *iFluids* to map even detailed and complex geometries onto fine grids within a fraction of a second. To optimize Lattice-Boltzmann solvers, especially the idea of using patches with higher resolution or boxes for a reduced grid could be of interest for interactive applications. Here, a combination of both voxelization methods would lead to the most performant voxelization within *iFluids*.

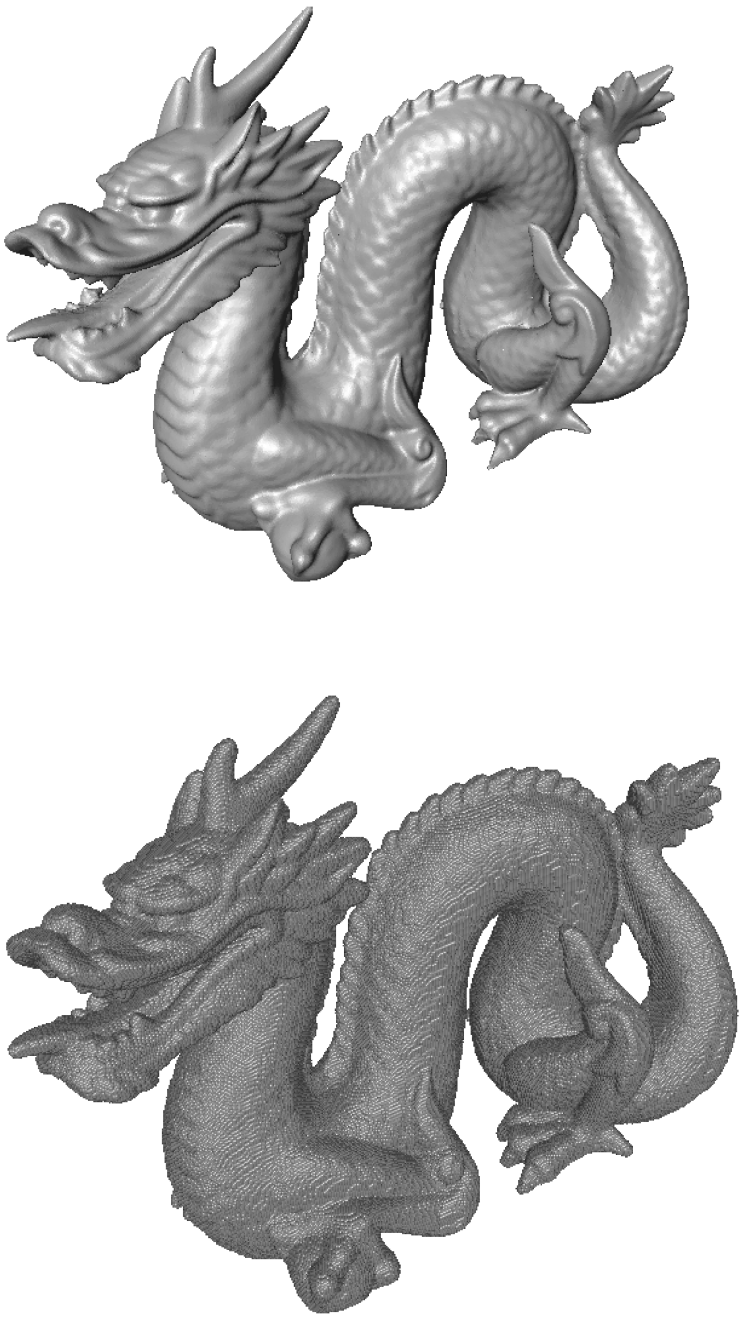

Figure 6.13: *Here, the 'Dragon' test geometry is shown, which has been used for performance evaluation of iFluids' voxelization. On the top a dragon geometry taken from The Stanford 3D Scanning Repository (2007) is shown, which is defined by 499870 triangles. The bottom picture shows the voxel discretization of this model on a 512x512x512 grid.*

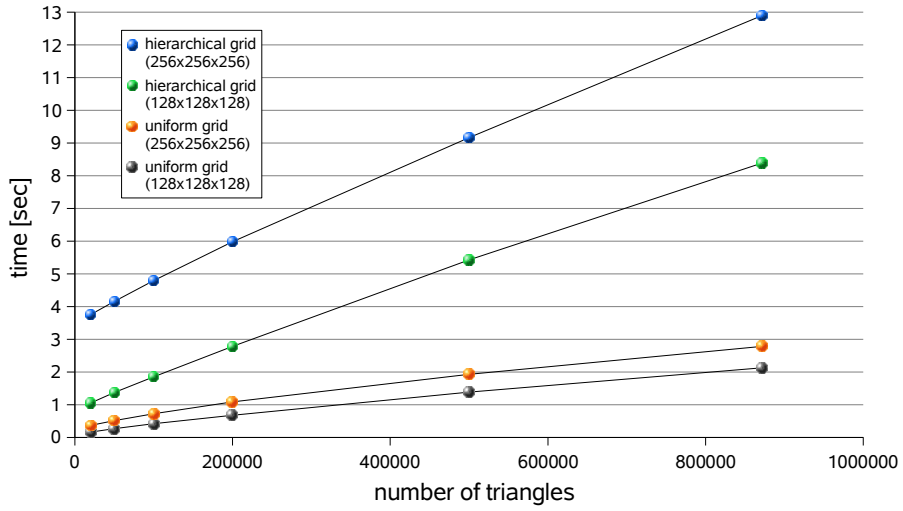

Figure 6.14: *Using the algorithm for hierarchical grids, the upper graphs (blue and green) show the voxelization time for the Dragon test geometry in dependence on the number of triangles for a grid resolution of 256x256x256 and 128x128x128, respectively. Only for uniform grids it is possible to use the faster algorithm and the voxelization time can be reduced to about 22% for the finer (orange) and to 25% of the coarser (grey) grid.*

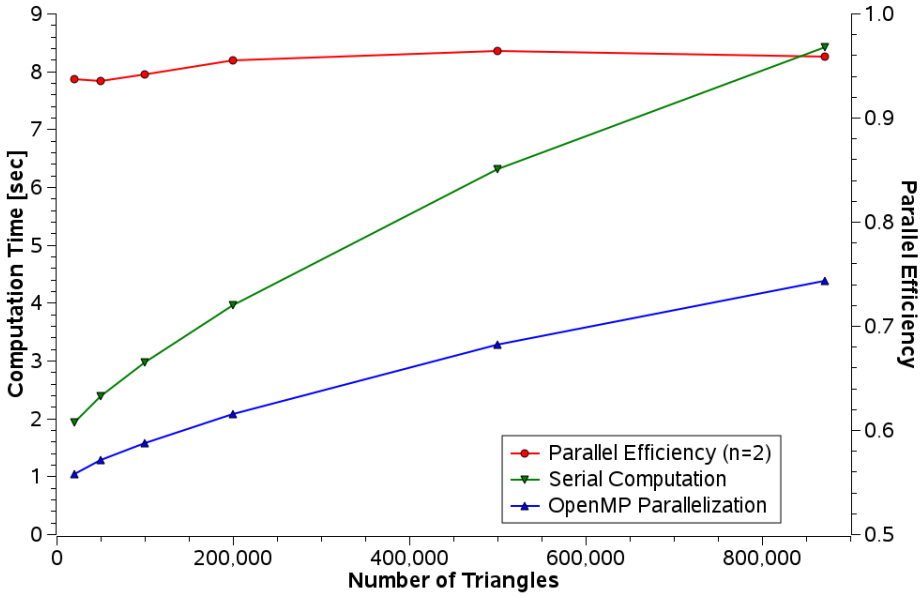

Figure 6.15: *The parallelization graph shown here demonstrates the speedup achieved by performing the voxelization with two as compared to one processor. For this benchmark the 'Dragon' shown in Fig. 6.13 was used, composed of varying numbers of facets. The voxel grid's resolution was set to 512 points in each direction*

Chapter 7

Realization Aspects with Respect to Computational Steering

In the last four chapters the main components of the computational steering framework underlying *iFluids* have been described, namely the simulation kernel, the grid generator and the visualization and steering client.

To facilitate *real* computational steering on a high-end system configuration with a Teraflop supercomputer and an external visualization and steering front-end, these components need to be connected in an efficient way. This chapter will show how the modules have been connected with each other and evaluates the overall performance of the online steerable application *iFluids*.

7.1 Communication Layout

As described in Chapters 5 and 6, the visualization workstation provides the functionality to display the current simulation data and to interact with the simulated scene at the same time. In the following, focus will be put on how an appropriate design of communication between the individual modules can be laid out to meet these special technical challenges of computational steering.

When a user interaction has occurred, the corresponding modifications are sent to the simulation engine, where the information is incorporated into the simulation model immediately. On the supercomputer new results are computed based on the updated simulation configuration. As soon as they are available the data are sent to the visualization client, where the user can observe the adaptation of the fluid in a series of flow updates (see Fig. 7.1).

A closer view onto the details of the models and their data streams is shown in Figure 7.2.

On the visualization (VIS) and steering (STEER) side an additional communication thread (COM) has been included to uncouple the checking for incoming results and the sending of user modifications and to not interrupt steering and post-processing. To keep the data transfer as short and infrequent as possible, only modifications to the setup are forwarded. Therefore, the transmission process is not triggered until after the user has completed any modifications. Since

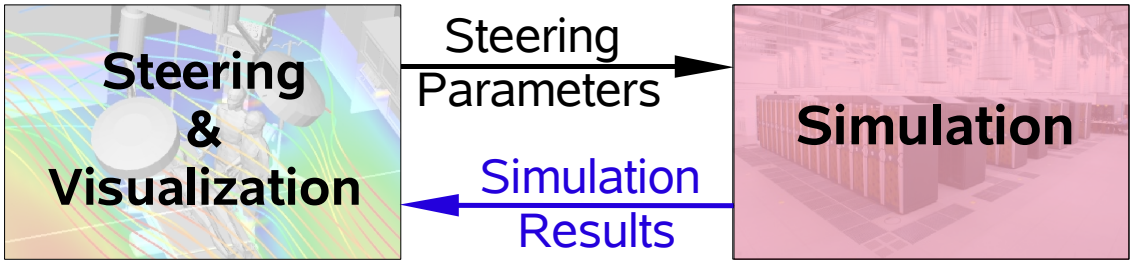

Figure 7.1: *On the visualization and steering workstation the user can visualize the current flow data received from the simulation running concurrently on the supercomputer. On the occasion of user interactions the specific modifications are sent to the supercomputer where they are incorporated into the simulation model immediately.*

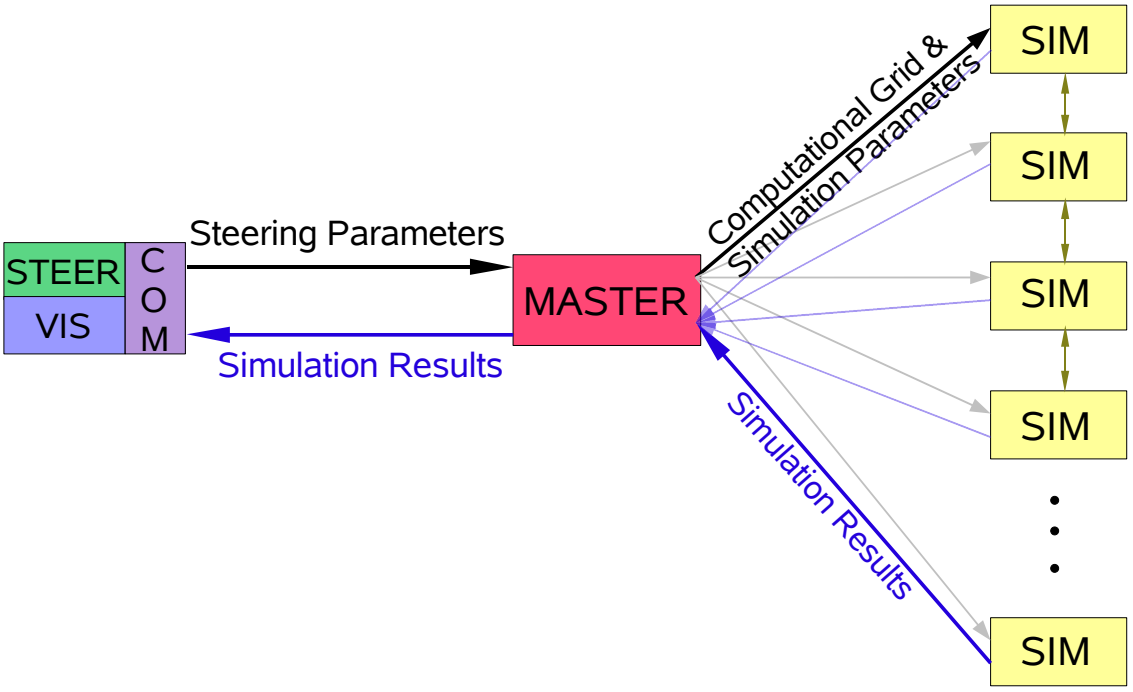

Figure 7.2: *On the left the steering and visualization front-end is shown. It consists of three threads, one for visualization (VIS), one for user interaction (STEER), and the communication (COM) thread. On the right the simulation master (MASTER) and the simulation slaves (SIM) are shown. Between COM and MASTER steering parameters and simulation results are exchanged, often in a distributed and heterogeneous hardware setup. Within the supercomputer simulation results of the slaves' computational domains are sent to the MASTER, which incorporates the steering parameters into the computational model and, accordingly, forwards decomposed grids containing all simulation data to the slaves.*

the results are not necessarily sent at regular intervals either, the receipt of data is, in essence, an event-driven process in both directions. In contrast to this Kühner (2003) used a communication layout with fixed intervals for sending and receiving neglecting any occurring events.

The simulation master (MASTER) can be seen more or less as the core of the computational steering framework connecting the visualization and steering frontend to the simulation. When the master receives modifications due to user interaction, they are incorporated into the computational model, using the powerful grid generator described in section 6.4. Then it performs the domain decomposition and sends the computational grid and all further necessary information to the simulation salves (SIM). The results computed by the slaves are gathered by the master and are prepared for being sent back to the visualization and steering client.

The slave processes (SIM) on the supercomputer mainly perform the Lattice-Boltzmann computation. To achieve good performance, it is essential to use vendor-optimized intra-machine (and possibly inter-machine) MPI for the communication with other processes. As long as no interaction has occurred, the slaves send current results at user-defined, regular intervals and check for updated computational grids. In the event of interaction, a new grid is received and new results can be sent after just a few time-steps to give the user fast initial feedback depending on his manipulations. Consequently, the transmission intervals are not necessarily regular throughout the run.

Since the visualization is usually conducted on an external graphics workstation with a different hardware architecture from that of the supercomputer, an appropriate MPI derivate is needed for the inter-machine communication, between master and visualization and steering front-end, which also enables the slaves to use vendor-optimized MPI. In this respect, the Globus MPICH-G2 (MPICH-G2, 2007) and PACX-MPI (Pacx-MPI, 2007) libraries have been tested within *iFluids*. Unfortunately, all efforts to port MPICH-G2 to the SR8000 with vendor-MPI enabled have not been successful. At the time the measurements were done for the SGI Altix, Globus MPICH-G2 was not yet available on this system. Therefore, all performance data presented in the following section are based on the PACX-MPI library. In this context it is important to note one drawback of PACX-MPI, namely the fact that it starts two extra MPI processes on each system for internal purposes. In the case of the Hitachi, which has only eight interactive nodes, only six nodes could be used for the application, accordingly. In addition, we observed that performance decreased by approximately 5 % due to the overhead caused by PACX-MPI Keller (2005). In the absence of other competitive alternatives one has to simply accept this for the time being.

7.2 Framework Performance

After all involved modules are connected to the computational steering framework, the overall performance is investigated and solutions for problems which could be detected are discussed in the following.

Overlapping Communication and Computation

As shown in Figure 7.2, the master process uncouples the communication between visualization and steering terminal and simulation, to avoid communication dependencies between simulation slaves and steering terminal. The decoupling of computation and communication is shown in Figure 7.3 for a trace collected with the *Intel Trace Analyzer* (intel, 2007).

Another important task for the master is to collect results from the slaves and send them, combined in a single message, to the visualization client to avoid additional latencies. This is especially important when the network connection between supercomputer and visualization client is limited by routers, firewalls, and slow connections — maybe even in 'competition' with other users.

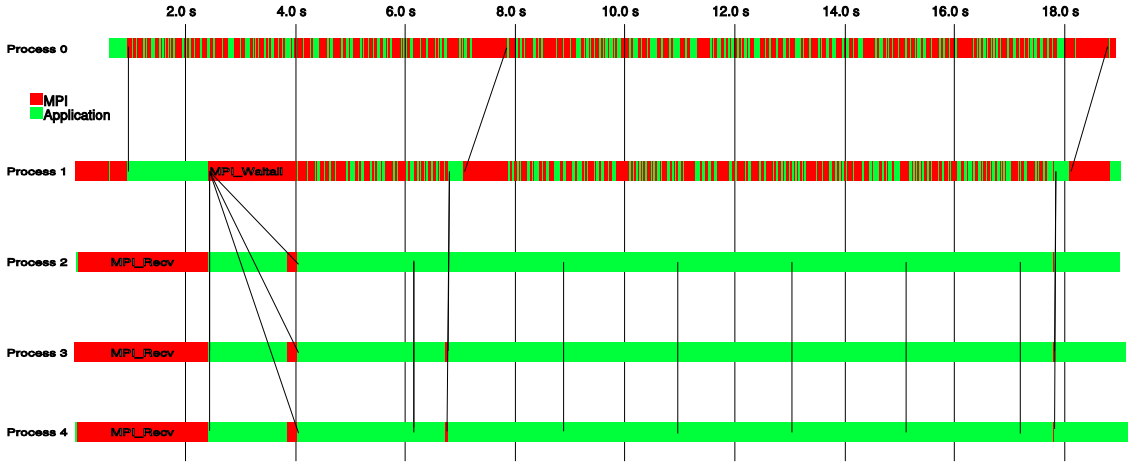

Figure 7.3: *This trace depicts the distribution of computation and communication. In this case, five processes were recorded, namely visualization (process 0), simulation master (process 1), and three slaves (processes 2-4). The timeline is given on the abscissa and covers 11 time-steps starting from the application's initialization. Red marks represent MPI function calls while green shows periods of computation or other application-specific processing. Frequent checks for user interaction and time-consuming communication to an external machine can be taken off the computation processes by introducing a master node.*

Finally, Figure 7.3 also reveals the main advantage of introducing a 'collector' node: During the time-consuming transfer of results (from process 1 to 0), the slave processes (2-4) are able to overlap computation with communication as long as the computation time is longer than the communication time. It is also necessary to point out that using *non-blocking* MPI communication on the SR8000 (and several other MPI implementations, cf White III and Bova (1999)) does not allow this overlap.

The impact of this fact on the performance of the application is also shown in Figure 7.4, which compares the performance of the application with and without this master node. The measurements have been carried out with the computation

running on the Hitachi SR8000 at LRZ and the visualization on an external Dual Opteron Linux PC at the Chair for Bauinformatik.

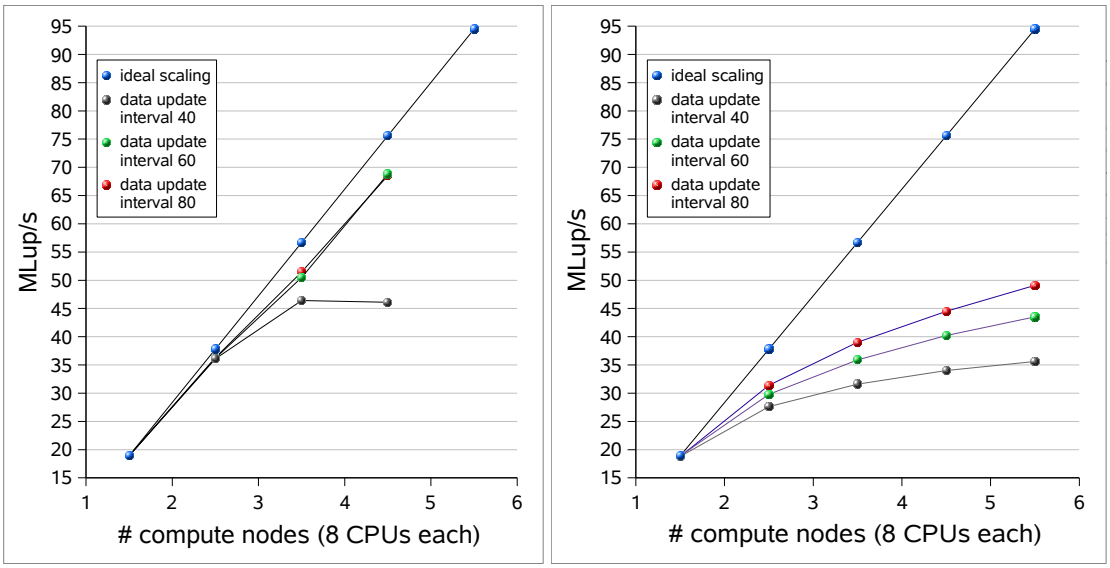

Figure 7.4: *The two graphs show the scaling behavior for various data update intervals in dependence on the number of computation nodes used. Performance is measured in MLup/s (million lattice site updates per second) at the visualization process. The left-hand panel refers to runs with a simulation master. Runs with update intervals of 60 or more time steps show good scaling already, whereas shorter intervals have a negative influence. On the right-hand panel, the same measurements were performed without a simulation master and the application shows poor scaling efficiency (<50%) in all cases.*

Furthermore, the right graph in Figure 7.4 shows the dependency on the data update intervals. True scaling of the application could only be achieved using a master node in combination with long update intervals of the visualized scene. The saturation of performance is caused due to only a small amount of computational work per slave while the communication between master and visualization front-end takes more time than the computation by the slaves. After a few cycles already, the master is still transferring data while the slaves have to wait until being able to send new results to their master. Evidently, this has severe impact on the performance and may be identified as a 'network bottleneck', which will be discussed in more detail in the following section.

Network Bottleneck

When recognizing the impaired scaling behavior during the performance measurements with short update intervals, we looked for possible explanations and found that the network connection of the SR8000 to any other computer was fairly unsatisfactory. It turned out that the maximum throughput on its *single* Gigabit

Ethernet line achieved only 230 MBits/sec maximum. To get an idea of the influence of the network connection, the same measurements have also been performed on Sara's SGI Altix 3700 and were benchmarked there additionally.

To be able to compare these measurements the offline performance (i.e. performance without a connection to a visualization and steering terminal) has been measured on both systems (see Chapter 4) and is shown again in Figure 7.5.

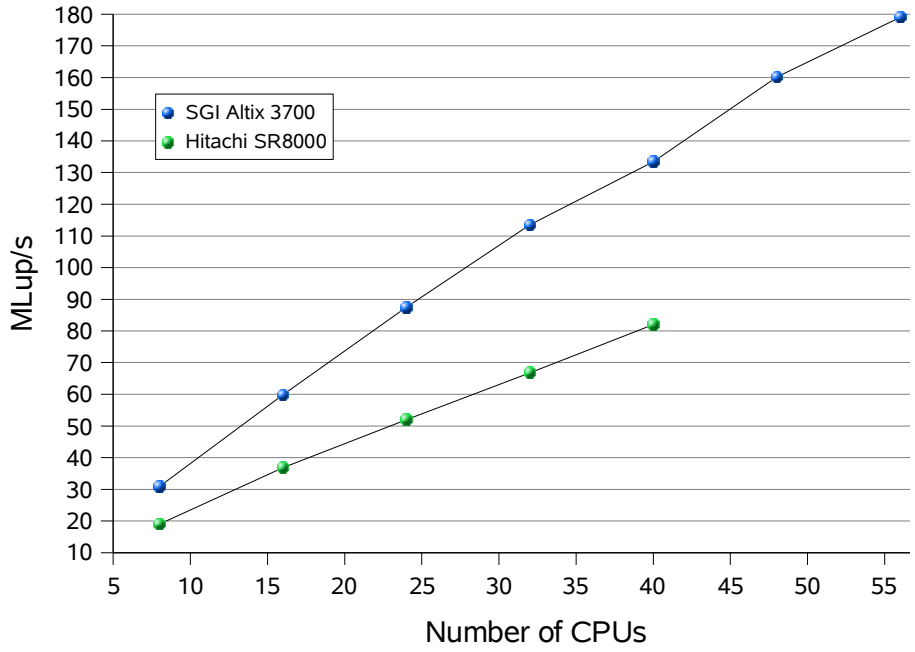

Figure 7.5: *This graph shows the performance of the Lattice-Boltzmann kernel running on Hitachi SR8000 (green) and SGI Altix 3700 (blue). Both codes show good speed-up behavior on the respective machine. The performance gain seen after porting the kernel to the Altix was about 70% when using 40 processors on both machines.*

For the setup at Sara, an external visualization client was connected to the Altix machine also via Gigabit Ethernet. It was synthetically benchmarked to a bandwidth of 720 Mbits/sec. Comparing the saturation performances on the SR8000 and the Altix 3700 showed a performance gain of 210% as compared to 70% (see Figure 7.6). This corresponds to data updates every 13 seconds on Hitachi's SR8000 and only 4 seconds on the SGI Altix for 380x355x122 grid points of the example room with dimensions of 7.6m x 7.1m x 2.24m. The graphs in Figure 7.6 also show that even with a higher performance of the computational kernel data updates every 13 seconds cannot be surpassed with the network connection provided in the setup of SR8000 with external visualization. Only the superior efficiency of the network at Sara is able to further decrease the data update time significantly.

To further improve the application with respect to the network bottleneck even for high update rates and high resolution grids, compressed data transfer

between computation and visualization should be investigated. If possible the amount of data sent to the visualization and steering front-end should be minimized. This is, for example, an option for porous media, where the boundary nodes need not be sent. Data reduction as suggested in Kühner et al. (2001) does not seem to solve the problem per se, since the message size can only be decreased by a small amount inevitably accompanied by (considerable) computational costs.

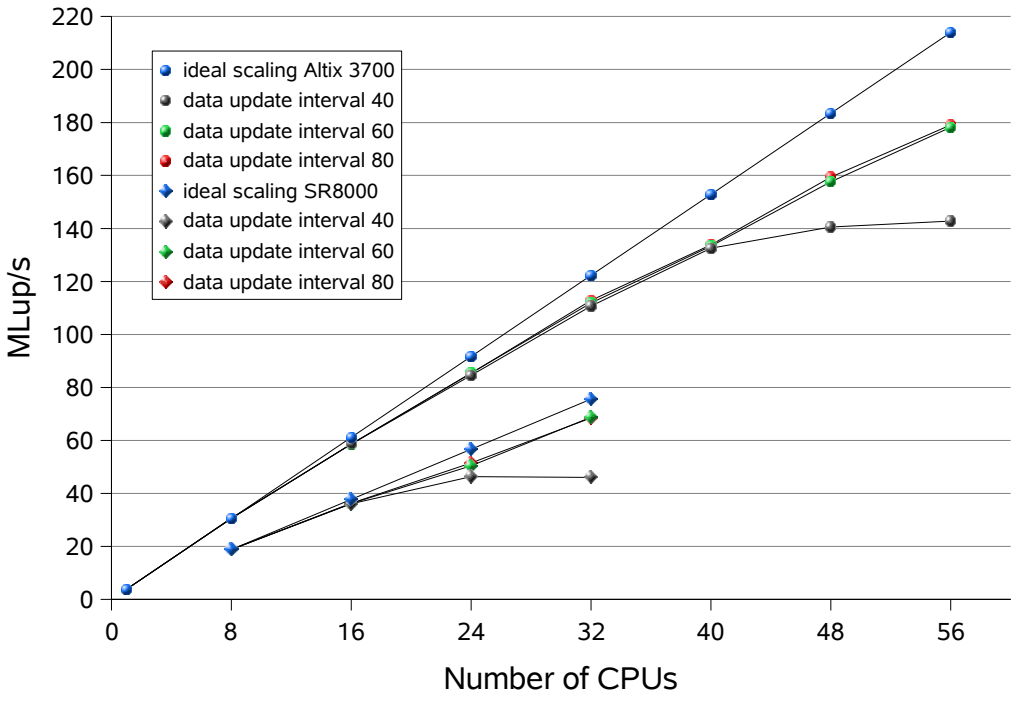

Figure 7.6: *These pairs of graphs show the performance gain in dependence on the number of time steps between result updates to the visualization for runs on the SR8000 and Altix 3700, respectively. Both machines show performance saturation when frequently updating results, because communication time then exceeds computation time. Updating less often than every 40 time steps quickly restores good scaling behavior. It is evident that the SGI can make much better use of its outgoing Gigabit ethernet network than the older Hitachi system.*

Still another way of bypassing the bottleneck would be to move the post-processing and graphics computation onto the supercomputer (maybe even in a parallel version). This, in fact, seems to be the most promising solution for the dilemma, but may not always be applicable. Especially on the Hitachi SR8000 this ansatz was infeasible, as this machine was not well-suited to support post-processing or visualization due to its bad performance in non-vectorizable code. An alternative solution would be a coupled system of a high-performance computer (well-suited for the Lattice-Boltzmann method such as vector computers) and a high-end graphics workstation (such as an SGI Prism or a visualization

cluster) which are connected via specialized and dedicated networks at the same computing center. Then, the visualization data could be transferred as a (compressed) video stream as suggested, e.g., in Heinzlreiter (2005). Although first projects in this direction have already started and products in this direction are already available (remote visualization services such as SGI OpenViz Server, IBM, HP, SUN, ..., VNC), they currently do not yet address all requirements sufficiently. For instance, immersive environments cannot be used with this approach yet.

7.3 Visualization and Steering on Multiple Clients

The modules building the computational steering framework are strictly coupled and implemented with interfaces that allow combining them and interacting with each other. Therefore, it is possible to exchange modules or insert additional modules as has been done in Borrmann (2007). This example implemented an extension to *iFluids* to allow for collaborative engineering.

As it is shown in Figure 7.7 a collaboration server has been inserted between the simulation master and the visualization and steering client. After extending the communication module (COM) on the client side for some additional collaboration parameters, all other modules were reused. The collaboration server is designed in a way that enables the framework to be connected to different simulation servers at a time and to attach or detach an arbitrary number of clients during a collaborative session (see also Borrmann et al. (2006)).

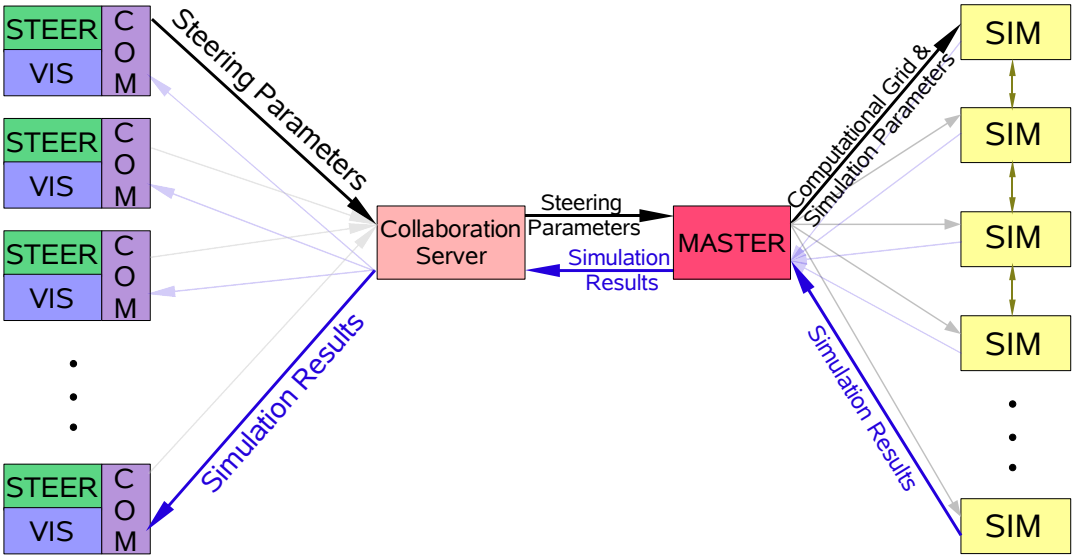

Figure 7.7: *For collaborative engineering a collaboration server has been inserted into the original layout shown in Fig. 7.2. In this way, multi-client computational steering is made possible.*

Chapter 8

General Applicability — A Case Study

This chapter investigates the applicability of a computational steering environment for the study of indoor ventilation problems. To this end, a test case has been simulated as a batch job on a grid with high resolution to serve as a high-accuracy reference. It is then compared with simulations on (increasingly) coarser grids such as the resolutions used in interactive online simulations. This approach has been taken to help determine the minimum grid resolution needed for a preliminary study or estimation of an indoor air flow problem and which is the maximum grid resolution that can be handled interactively. The test case in this study has been the simulation of a ventilation system as it is used in a real setup of an operating room at the Klinikum Rechts der Isar in Munich.

8.1 Ventilation Systems of Operating Rooms

Modern operating rooms set high requirements with regard to aseptic conditions of the working place and, in particular, in the vicinity of the patient and the operation site. Especially for operations lasting several hours the risk of the patient's wound becoming infected by bacteria must be reduced as much as possible. Since the completely bacteria free operating room is not feasible for technical reasons, a ventilation system is used for applying freshly filtered abacterial air onto the patient's wound to push the room's air aside. Therefore, the stream of filtered air should be directed straight onto the wound so as to avoid any contact with human beings or operating facilities as far as possible.

At the Klinikum Rechts der Isar, two operating rooms with different ventilation systems have been inspected and modeled accordingly. The two rooms (OP I and OP II) are of roughly similar size, namely 6.3m x 6.25m x 3.5m and 5.45m x 6.25m x 3.1m. Figure 8.1 and 8.2 show photographs of these rooms, while Figure 8.3 and 8.4 show a close-up of the different ventilation facilities of each room.

Figure 8.1: *OP I: This operating room is equipped with a ventilation system entering from the ceiling. The photograph shows the standard setup just before an operation starts.*

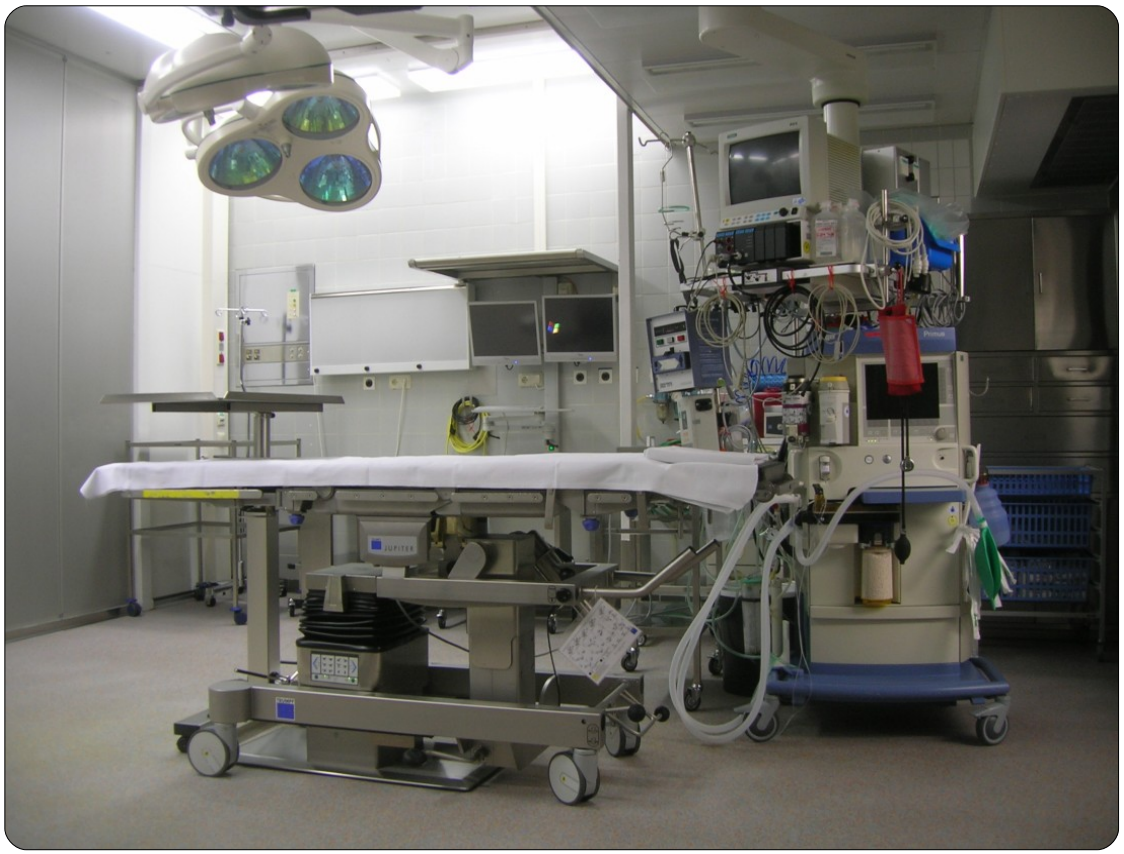

Figure 8.2: *OP II: In contrast to OP I this room has ventilation systems entering from one wall. The photograph also shows the classical arrangement of facilities before an operation.*

Figure 8.3: OP I: Two rectangular ventilation inlets are placed at the ceiling around the area of the operating table. The velocity of the instreaming air is adjusted to 0.24 m/s which corresponds to a ventilated air volume of 2780 m^3/h.

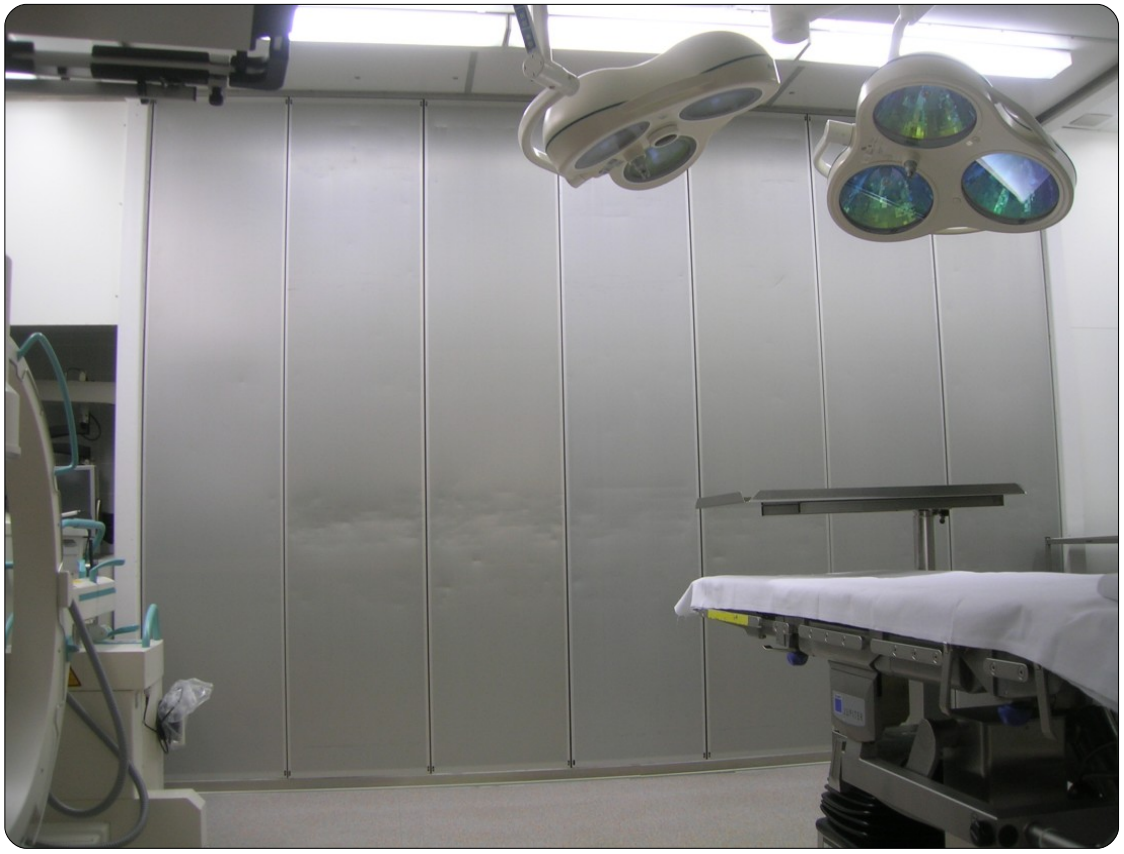

Figure 8.4: OP II: Almost the whole wall is used as the horizontal ventilation inlet. The air is blown with a velocity of 0.28 m/s corresponding to a ventilation volume of 14035 m^3/h.

During an operation three to five people are usually working together within the aseptic zone. Moreover, at least one table with surgical instruments is placed at the end of the patient's table. Because of the many people involved and, additionally, the lighting facilities placed close to the operative situs, the impact of the ventilation and, therefore, the bacteria concentration at the wound is not easily predictable. Typical operation scenes are shown for both rooms in Figures 8.5 and 8.6

Figure 8.5: *A standard scene during an operation in OP I. Here, a mobile ventilation facility is used in addition to the ceiling ventilation.*

Figure 8.6: *A standard scene during an operation in OP II. As opposed to OP I the position of the operation table with the patient can be freely placed for an optimal ventilation.*

8.2 Simulation Studies with Varying Grid Resolutions

The two operating rooms, OP I and OP II, have been modeled with their corresponding boundary conditions[1]. By doing a series of simulations with different grid resolutions, the minimum resolution has been determined which is needed to still allow physically meaningful predictions but, at the same time, is still runnable as an interactive simulation. Figures 8.7 and 8.8 show the models of the operating rooms I and II during an operation, while Figures 8.9 and 8.10 give the simulation results of theses scenes.

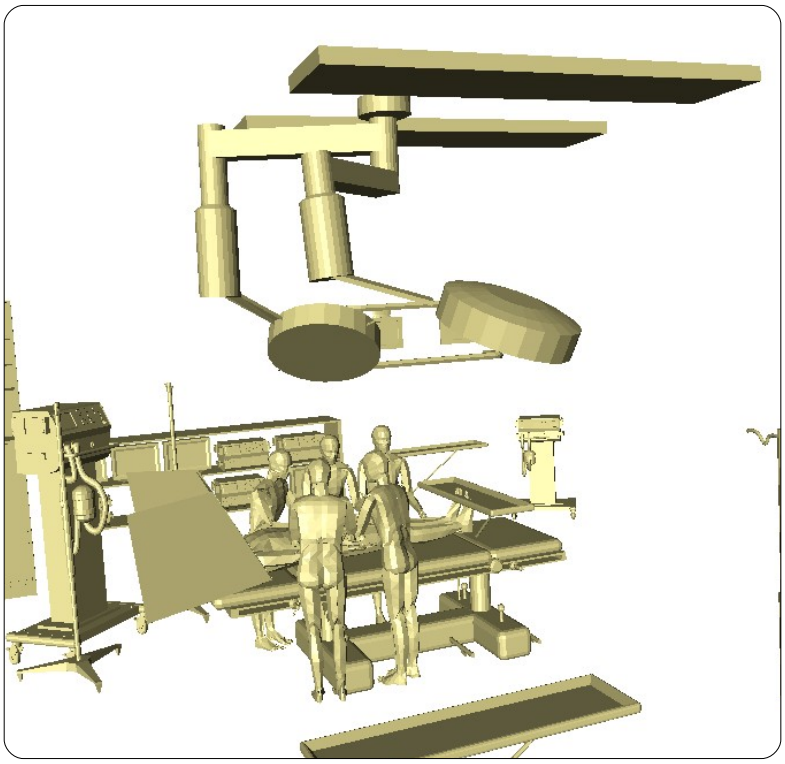

Figure 8.7: *This snapshot shows the simulated surgery scene in OP I.*

After simulating the operation rooms OP I and OP II the configuration of OP II with the additional ventilation device has been run with different grid resolutions. The simulation series shows that already a resolution of 'only' 126x70x125 voxels results in non-negligible deviations. For a resolution of 180x100x179, i.e., one grid point every 3.5 cm, the results agree quite well with the results of the reference solution. After 2500 to 3000 Lattice-Boltzmann timesteps the results oscillate around the steady state. Nevertheless, the qualitative behavior of the flow can already be predicted after about 1500 timesteps.

Running this setup at the minimum resolution of 180x100x179 grid points on the SGI Altix 3700 (connected to a standard laptop for steering over Gigabit Eth-

[1]The geometry of the operating room is based on models taken from the archives www.turbosquid.com, www.3dcafe.com, and www.lightscape.com/VRML/lib/or.wrl

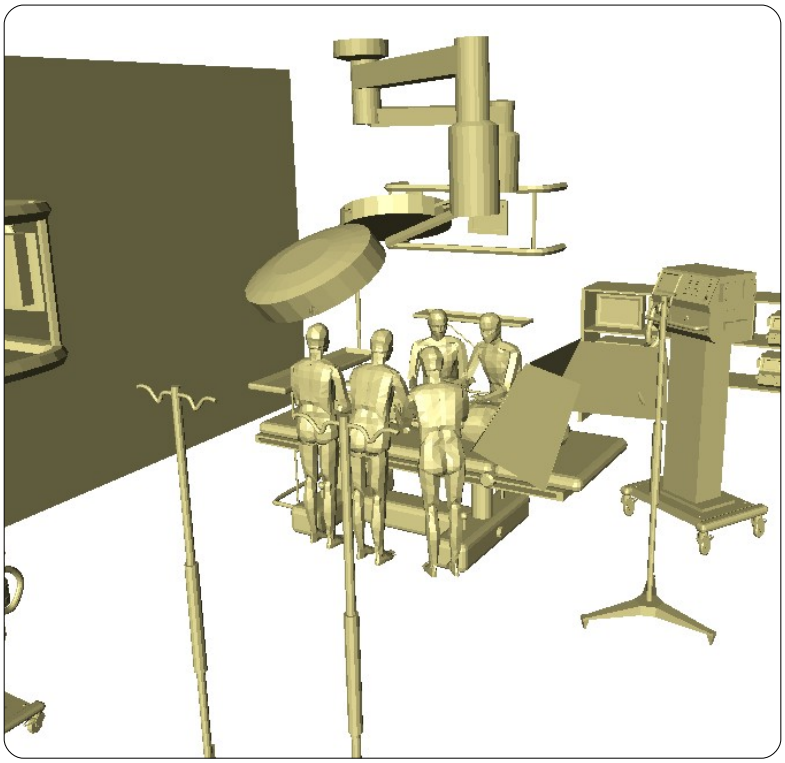

Figure 8.8: *Here, the simulation model for OP II during a surgery is shown.*

ernet as done in the benchmarks in Chapter 7), a data update after 60 Lattice-Boltzmann steps would arrive at the visualization and steering frontend with a rate of 1 frame/s. Therefore, 1500 timesteps would take about 25 seconds to compute and the first reliable predictions can be made. This also marks the point in time, when the first modifications to the setup are starting to make sense. After an additional 25 seconds or 3000 Lattice-Boltzmann steps, the results allow to predict the principle flow.

Taking into account that even more powerful systems are already available (such as LRZ's SGI Altix 4700 together with their new visualization cluster and a dedicated 10 Gigabit Ethernet connection), these results seem quite satisfactory from an engineering standpoint. Accordingly, the bottom line of the investigation in this chapter is that a computational steering application such as the one developed and presented in this work can, in fact, be a helpful supplement in the engineering practice to the current state of the art.

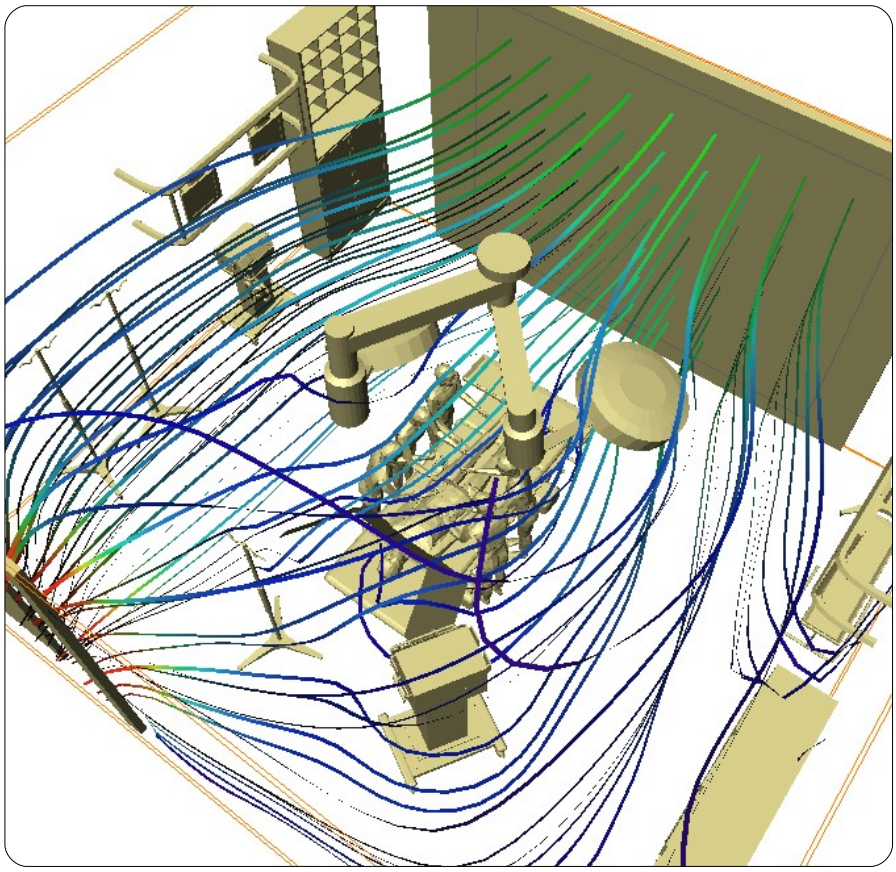

Figure 8.9: *This figure presents the simulation results for OP II for a grid resolution of 545x625x310. The stream lines display the fluid streaming from the ventilation inlet on the back wall to the ajar door in the front. Some of the ventilation arrives at the patient's situs, but the main fluid stream flows around the surgery zone.*

Figure 8.10: *The upper snapshot shows the ventilation in OP I coming from the ceiling. It can be seen in this setup that the filtered air is deflected through the lamps and, accordingly, the patient's wound is not as well ventilated as in OP II. The bottom snapshot represents a scene where an additional ventilation device has been inserted. This device can be freely adjusted as needed and, evidently, is able to compensate the missing air flow onto the operation situs. Both simulations were run with a grid resolution of 630x625x350.*

Chapter 9

Universal Applicability of *iFluids* — Computational Steering Framework

This chapter shows the computational steering framework underlying *iFluids* being applied on a completely different field of application, thus demonstrating its capability and suitability to handle a whole variety of computational engineering problems. In this case, the framework was used for interactive blood flow simulation with respect to vascular reconstruction, which has been done in cooperation with the Institute for Computational Science at the University of Amsterdam.

9.1 Vascular reconstruction

A special field of expertise of the Institute for Computational Science at the University of Amsterdam is the research on computational hemodynamics as needed for vascular reconstruction simulations. Vascular reconstruction includes surgical operations like adding shunts, bypasses and placing stents (in the case of aneurysm) or applying thrombolysis techniques, balloon angioplasty, bypasses, etc. for a stenosis. To find the best treatment is far from trivial and a simulation tool to support the verification of the operation plan may serve as a good supplement to classical approaches. In collaboration with this institute the computational framework was applied to this kind of simulation.

To evaluate the flexibility of the computational steering framework the aim was to allow, in principle, an online simulation of blood flow within a part of an artery. Again, the underlying simulation kernel was based on the Lattice-Boltzmann method but did not incorporate details of hemodynamics as described in Artoli (2003).

9.2 Extensions of *iFluids* for Blood Flow Simulations

In the adaption of the original standard fluid simulation to blood flow simulation, the layout of *iFluids* could be completely reused. Only two central changes had to be made. Firstly, the grid generator needed to be extended to provide a filling

algorithm which also sets voxels in the interior of an object. This was necessary, since in contrast to the indoor simulations the blood flow simulation takes place *within* the objects, .i.e. the arteries and shunts, added to the scene. Secondly, due to simulation efficiency the data layout needed to be adapted, because in a blood flow simulation the grid points most often are only sparsely populated — cells representing blood represent only a fraction in the range between 5% to 15% of the scene's bounding box volume (see 9.1).

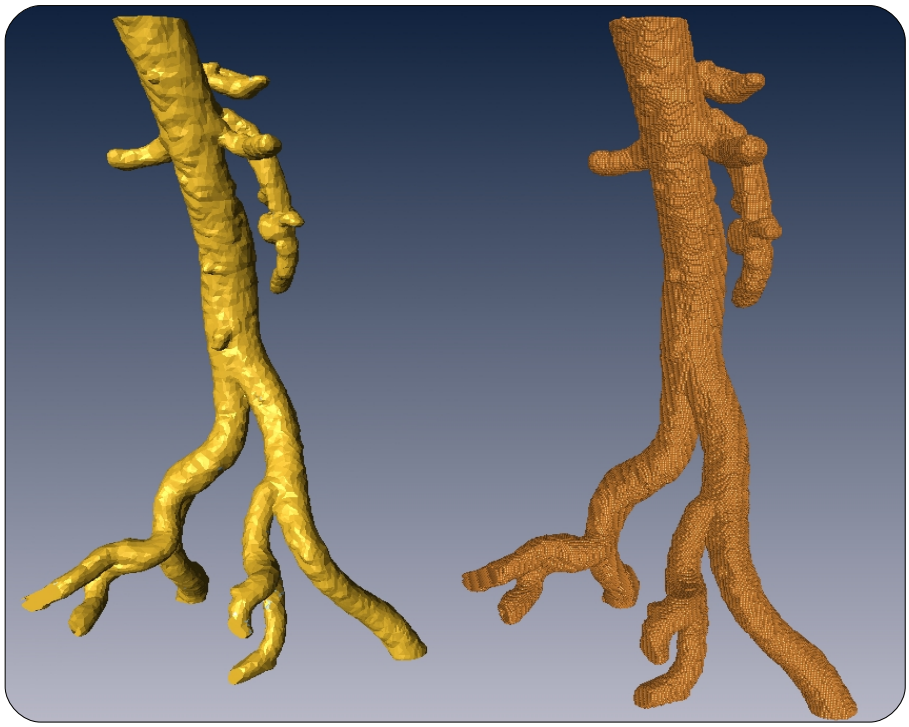

Figure 9.1: *On the left this figure shows the triangulated surface of the aorta. On the right its voxel representation is depicted. The underlying grid resolution was 433x206x126 and 127432 voxels were set as boundary nodes (including boundary conditions like inlet and outlet) and 403734 fluid nodes in the interior. Therefore, the percentage of 'relevant' grid nodes is 4.73%. The voxelization was done in 0.68 sec using a single CPU on an SGI Altix 4700 (1.6 GHz).*

To replicate the obstruction of a blood vessel and its surgical remedy two simple forms of interaction during the blood flow simulation are supported. On the one hand blockers to the interior of the artery for narrowing or completely blocking the flow can be added to the scene. On the other hand artificial bypasses can be added or removed from the artery, as can be seen in Figure 9.2.

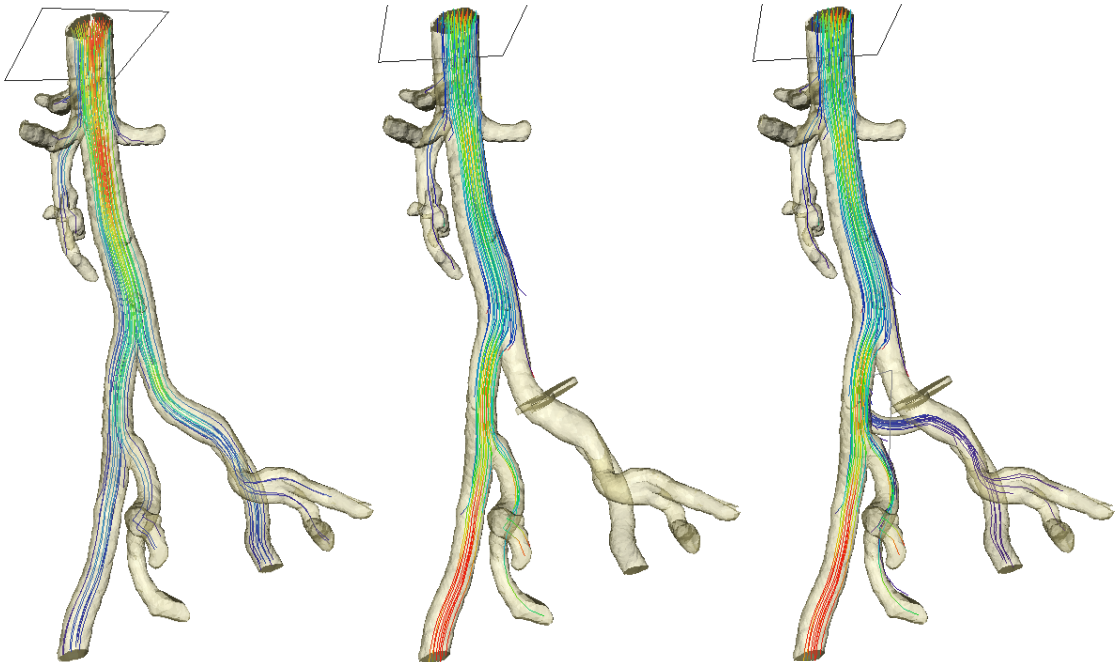

Figure 9.2: *This figure shows three screenshots taken during an interactive simulation session. On the left the unmodified artery with blood flow in its interior is shown. In the middle picture one branch of the aorta is blocked. Finally, on the right a bypass is inserted to supply the blocked part of the artery through this artificial vessel.*

Grid Generation

Objects added to a scene during an indoor simulation are, by definition, interpreted as fluid obstacles, which may have boundary conditions on their surfaces such as, e.g., a personal computer as a heat source or velocity boundaries of a tabletop ventilator. Therfore, the interior of the obstacles can be neglected. However, in case of the blood flow simulation especially the interior of the artery is of interest and the surrounding nodes are neglected. To cope with this situation the grid generator marks all nodes of the outside of the artery as neglectable, the boundary nodes with their corresponding boundary conditions, and the interior as fluid nodes. An important extension in this respect is the capability of intersections of objects with each other. This, for instance, is needed when modeling the adding of a bypass such that the boundary nodes of the bypass which extend into the aorta are removed on both sides and the aorta is opened where the bypass enters it. When adding a blocker these boundary nodes are attributed a higher priority and are therefore not replaced by fluid nodes. This special form of intersection functionality is demonstrated in Figure 9.3 for a smaller part of the aorta shown in Figure 9.1, which has been extracted and scaled up for this purpose (see Figure 9.4).

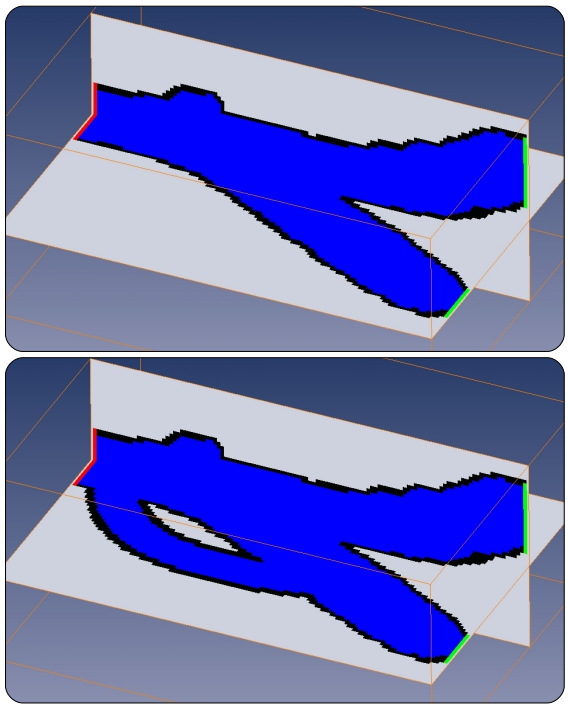

Figure 9.3: *These two figures show the boundary conditions for a small part of the aorta. In the upper part the original aorta can be seen, while in the lower part a bypass has been added. The red color refers to the inlet condition and the green color to the outlet. Fluid nodes are colored blue, the wall nodes black, and the neglectable nodes outside the artery are depicted in grey. The triangulated aorta surface is shown in Figure 9.4*

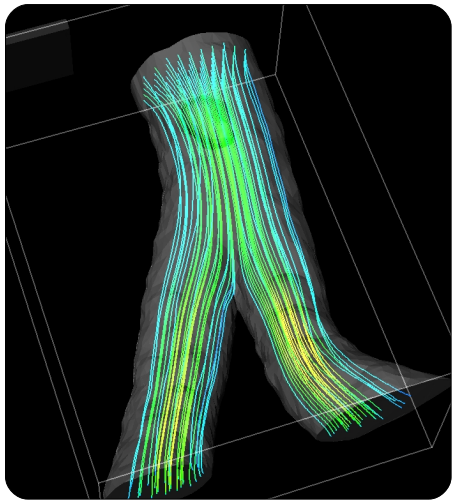

Figure 9.4: *This figure shows a small part of the aorta with streamlines visualizing the interior blood flow.*

Data Layout

Since an artery usually is a rangy structure with multiple and increasingly delicate branches, the fraction of fluid nodes within the bounding box ranges approximately between 5% to 15%. This fact allows a special form of optimization of the simulation kernel and has been implemented as an additional option, which skips all non-fluid nodes and performs the Lattice-Boltzmann update on a list structure of fluid and boundary nodes only.

Additionally, the network traffic between the supercomputer and an external visualization and steering terminal is reduced by only transferring these relevant nodes and their location within the simulation field. As stated in (Artoli, 2003), a Reynolds' number of approximately 500 is sufficient for blood flow simulation within an abdominal aorta. To resolve this kind of flow situation for the exemplary artery shown in Figure 9.1, a moderately fine grid is sufficient. In this case the transfered data volume causes no bottleneck. The described aorta simulation could be run on an SGI Altix 3700, while the steering and visualization was performed on a laptop (Intel 2.13 GHz processor with ATI Mobility FireGL V5000). Interaction with the simulation and its response was, by subjective impression, swift and fluent.

Chapter 10

Summary

In this thesis the central aspects, problems, and principle approach to realizing a computational steering framework have been elaborated on. In particular, the utilization of supercomputers for fluid simulations and high-end visualization techniques like immersive Virtual-Reality Environments for a remote visualization and steering frontend have been covered in detail. It has been shown that within the *iFluids* computational steering framework a user is enabled to interact with the simulation during its execution without the need of interrupting or restarting it. The main feature distinguishing it from all currently known computational steering approaches is its powerful interaction possibility with regard to the geometrical layout. It is possible to load arbitrary geometries exported from CAD systems or similar software from the filesystem without any special preparations. Other approaches, so far, support only predefined parameterized objects — if geometry can be changed significantly at all. Besides interacting with the geometry, the user can modify flow parameters, define new or change existing boundary conditions during runtime. For advanced users or benchmarking purposes even optimization options can be changed during runtime.

By using an integrated front-end for simultaneous visualization and steering augmented through a Virtual-Reality Environment the usage of this kind of interactive simulation tool becomes intuitive and allows to quickly gain insights even when studying complex flow phenomena.

The simulation kernel underlying the framework is based on the Lattice-Boltzmann method and has been optimized for two types of supercomputers that have been available at the Leibniz Computing Center during the time of this thesis, namely the Hitachi SR8000 and the SGI Altix 3700[1]. These machines represent the oppositional architectures of a pseudo-vector and a shared-memory system, respectively.

When running the simulation on the SGI Altix and the visualization on an external graphics workstation, both connected via Gigabit Ethernet, an updated data set typically could be received and visualized every 4 seconds. In comparison, the performance achievable with the SR8000 connected with a remote vi-

[1]The code has also been tuned for performance on the LRZ Linux Cluster, where the application framework has been mainly developed on.

sualization was only about 25% of this value, even though the performance of the kernel running 'offline' lay at about 60%. It turned out that the reason for this finding was due to a network bottleneck between visualization and computation in so far as Hitachi's outgoing network connection could only provide 230 MBits/sec maximum bandwith.

The new system setup at the LRZ consisting of an SGI Altix 4700 and a visualization system based on a SUN x4600 Multi-Core Opteron system connected through a dedicated 10 Gigabit Ethernet interface promises an enormous performance gain for our computational steering application. Just the same, data updates every 4 seconds seems already quite satisfactory as compared to the usual waiting times of (several) minutes or even hours for general purpose commercial codes.

The applicability of the presented computational steering application has been tested by simulating real operating rooms modeled after two rooms at the Klinikum Rechts der Isar in Munich. These have been simulated with a very fine grid resolution to obtain a "reference solution" for comparison with interactive simulations on varying discretization grid sizes. It was found that already for moderately fine grids (e.g., a room of 6mx6mx3.5m, discretized to one grid point every 3.5cm) a good qualitative estimation of the flow can be made after 50 seconds (using a visualization laptop connected to the SGI Altix 3700 via Gigabit Ethernet). This enables an engineer to quickly test several setups and experiment with them interactively during the simulation within only a short amount of time. After examining the principle fluid behavior, a few carefully selected test cases can be run additionally in a more detailed offline simulation for quantitative analysis.

To evaluate the flexibility of the *iFluids* computational steering framework it has been applied exemplarily to a simplified artery simulation, where the user is able to (partially) block an artery or add bypasses during simulation runtime. After only a few adaptations interactive simulations could be performed for adequate grid resolutions using the SGI Altix 3700 and a laptop (Intel 2.13 GHz processor with ATI Mobility FireGL V5000) for visualization.

Hopefully, this thesis could show that a computational steering application as developed within this work can be a valuable enrichment for engineers during the construction design phase. However, there are still some open requirements with respect to computational steering. To be able to use a computational steering application like *iFluids* in real life on the supercomputer or cluster, the corresponding necessary resources on the machine must be available for exclusive interactive access during an engineer's working time. Especially in the case of a concurrent collaborative session, it is indispensable that the required resources are available at the time of appointment. This could be (and partially is already) realized through an enhanced scheduling system offering the possibility of reserving resources for a certain day and time of an appointment. This feature is referred to as "advanced reservation" which, understandably, is only hesitantly used in computing centers and not made publicly available or accessible to the general user.

Furthermore, when working with large computational grids and correspond-

ingly large visualization data sets, the communication between computation, visualization and steering front-end can become a bottleneck, especially between remote sites. To counter this network bottleneck, some sort of middleware would be needed, which supports (remote) visualization on specialized hardware that is connected to the supercomputer powerfully enough and transfers precomputed visualization data to the client — perhaps as a simple video stream. Two final requirements in this respect would be that also Virtual-Reality Environments with their special types of input device and collaborative engineering with independent views onto the simulated scene are supported. First products or projects in this direction (SGI OpenViz Server, IBM, HP, SUN, ..., VNC) have started and are becoming available already. However, they currently do not yet address these requirements sufficiently.

Future developments of *iFluids* will investigate these current techniques of remote visualization in more detail. It will be interesting to find out, how they could help to further improve the computational steering experience or at least partially solve the issues mentioned.

To extend the field of application for indoor simulations, the computational kernel of *iFluids* will be improved by using a more detailed physical model. In particular, this effort will be continued within the research project *ComfSim* (SIEMENS AG, 2006), which aims at developing an interactive CFD environment allowing an analysis of local thermal comfort by utilizing high-performance supercomputing facilities and Virtual-Reality techniques.

Bibliography

Abrams, M., Allison, D., Kafura, D., Ribbens, C., Rosson, M. B., Shaffer, C., and Watson, L. `http://research.cs.vt.edu/pse/intro.html` (2007).

Akenine-Moeller, T. Fast 3D triangle-box overlap testing. *Journal of Graphics Tools*, Vol. 6(1):pp. 29–33 (2001).

Akenine-Moeller, T. `http://www.cs.lth.se/home/Tomas_Akenine_Moller/` (2007).

Allard, J. and Raffin, B. Distributed physical based simulations for large VR applications. In *VR '06: Proceedings of the IEEE Virtual Reality Conference (VR 2006)*, p. 12. IEEE Computer Society, Washington, DC, USA (2006). ISBN 1-4244-0224-7. doi:http://dx.doi.org/10.1109/VR.2006.53.

answers.com. `http://www.answers.com/topic/risc` (2007).

Artoli, A. M. *Mesoscopic Computational Haemodynamics*. PhD thesis, Universiteit van Amsterdam (2003).

AVS Inc. `http://www.avs.com` (2007).

Bella, G., Filippone, S., Rossi, N., and Ubertini, S. Using OpenMP on a hydrodynamic Lattice-Boltzmann code. In *Proceedings of EWOMP 2002* (2002).

Bellemann, R. *Interactive Exploration in Virtual Environments*. PhD thesis, University of Amsterdam (2003).

Benzi, R., Succi, S., and Vergassola, M. The lattice Boltzmann equation: theory and applications. *Physics Reports*, Vol. 222:pp. 145–197 (1992).

Bernsdorf, J., Harrison, S. E., Smith, S. M., Lawford, P. V., and Hose, D. R. Numerical simulation of clotting processes: a lattice Boltzmann application in medical physics. *Math. Comput. Simul.*, Vol. 72(2-6):pp. 89–92 (2006). ISSN 0378-4754.

Bhatnagar, P., Gross, E., and Krook, M. A model for collision processes in gases. *Physical Review*, Vol. 94(3):pp. 511–525 (1954).

Biermann, G. and Kalze, F.-J. Helios - computer aided lighting, the path from simulation to prototype. In *29th ISATA Conference*. Florenz, Italy (1996). ISBN 0-7803-8431-8.

Borrmann, A. *Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken*. PhD thesis, Lehrstuhl für Bauinformatik, TU München (2007).

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

Brodlie, K., Wood, J., Duce, D., and Sagar, M. gViz: Visualization and computational steering on the grid. In *UK e-Science All Hands Meeting*, pp. 54 – 60 (2004). ISBN 1-904425-21-6.

Brooke, J. E., Coveny, P. V., Harting, J., Jha, S., Pickles, S. M., Pinning, R. L., and Porter, A. R. Computational steering in RealityGrid. In *UK e-Science All Hands Meeting* (2003).

Bryson, S. and Levit, C. The virtual wind tunnel. *Computer Graphics and Applications*, Vol. 12(4):pp. 25–34 (1992). ISSN 0272-1716.

Chen, H., Chen, S., and Matthaeus, W. H. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method. *Physical Review A*, Vol. 45:pp. 5339–42 (1992).

Chen, S. and Doolen, G. D. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, Vol. 30:pp. 329–364 (1998).

ConceptCar. `http://features.conceptcar.co.uk/psa-design-centre/design-centre-2.php` (2007).

Covise. `http://www.hlrs.de/organization/vis/covise` (2007).

Crouse, B. *Lattice-Boltzmann Strömungssimulationen auf Baumdatenstrukturen*. PhD thesis, Lehrstuhl für Bauinformatik, TU München (2003).

Crouse, B., Krafczyk, M., Tölke, J., and Rank, E. A LB-based approach for adaptive flow simulations. *International Journal of Modern Physics B*, Vol. 17(1-2):pp. 109–112 (2003).

Diederichs, C. J. *Kostensicherheit im Hochbau*. Deutscher Consulting Verlag, Essen (1984).

Domain Decompostion. `http://www.ddm.org` (2007).

Donath, S. On Optimized Implementations of the Lattice Boltzmann Method on Contemporary Architectures. Bachelor's Thesis (2004). Betr. Wellein, Hager, Zeiser, F. Deserno.

Dowd, K. and Severance, C. *High Performance Computing*. O'Reilly & Associates, Inc., Sebastopol, CA, USA (1998). ISBN 156592312X.

Eberly, D. *3D game engine design: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000). ISBN 1-55860-593-2.

Exa Corporation. `http://www.exa.com` (2007).

Freudiger, S., Hegewald, J., and Krafczyk, M. A parallelization concept for a multi-physics lattice Boltzmann prototype based on hierarchical grids (submitted 2007).

Frisch, U., Hasslacher, B., and Pomeau, Y. Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters*, Vol. 56:pp. 1505–1508 (1986).

Georgii, J. and Westermann, R. Interactive simulation and rendering of heterogeneous deformable bodies. In *Vision, Modeling and Visualization 2005* (2005).

Flensburger Schiffbau Gesellschaft. `http:www.fsg-ship.de` (2007).

Gibson, S. F. Beyond volume rendering: Visualization, haptic exploration, and physical modeling of voxel-based objects. In *Visualization in Scientific Computing'95*, pp. 10–24. Springer-Verlag, New York (1995).

Ginzburg, I. and Steiner, K. Lattice Boltzmann model for free-surface flow and its application to filling process in casting. *J. Comput. Phys.*, Vol. 185(1):pp. 61–99 (2003). ISSN 0021-9991.

Gottschalk, S., Lin, M. C., and Manocha, D. OBBTree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180. ACM Press, New York, NY, USA (1996). ISBN 0-89791-746-4.

Götz, J. *Numerical Simulation of Bloodflow in Aneurysms using the Lattice Boltzmann Method*. Master's thesis, Lehrstuhl für Informatik 10 (Systemsimulation), Friedrich-Alexander-Universität Erlangen-Nürnberg (2006).

Gropp, W. and Thakur, R. An evaluation of implementation options for MPI one-sided communication. In *Lecture Notes in Computer Science: Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Vol. 3666, pp. 415–424. Springer (2005).

Hager, G., Deserno, F., and Wellein, G. Pseudo-vectorization and RISC optimization techniques for the hitachi SR8000 architecture. In *High-Performance Scientific and Engineering Computing, Munich 2003* (2003). ISBN 3-540-00474-2.

Haines, E. and Wallace, J. Shaft culling for efficient ray-traced radiosity. In *Eurographics Workshop on Rendering*. Springer-Verlang, Berlin, Germany (1994).

Hartmann, H. *Detailed Simulations of Liquid and Solid-Liquid Mixing: Turbulent agitated flow and mass transfer*. PhD thesis, Technische Universiteit Delft (2005).

Haumont, D. and Warzee, N. Complete polygonal scene voxelization. *Journal of Graphics Tools*, Vol. 7(3):pp. 27–41 (2002).

Haydock, D. and Yeomans, J. M. Lattice Boltzmann simulations of attenuation-driven acoustic streaming. *J. Phys. A: Math. Gen.*, Vol. 36:pp. 5683–5694 (2003).

He, X., Chen, S., and Zhang, R. A lattice Boltzmann scheme for incompressible multiphase flow and its application in simulation of Rayleigh-Taylor instability. *J. Comput. Phys.*, Vol. 152(2):pp. 642–663 (1999). ISSN 0021-9991.

He, X., Zou, Q., Luo, L.-S., and Dembo, M. Analytic solutions of simple flows and analysis of nonslip boundary conditions for the lattice Boltzmann BGK model. *Journal of Statistical Physics*, Vol. 87(1-2):pp. 115–136 (1997). ISSN 0022-4715 (Print) 1572-9613 (Online).

Heinzlreiter, P. Interactive result visualization on the grid. In *Grid Computing for Complex Problems*, pp. 20–21. VEDA, VEDA (2005). ISBN 80-969202-1-9.

Hella KG. `http://www.hella.com` (2007).

Hirabayashi, M., Ohta, M., Rüfenacht, D. A., and Chopard, B. Characterization of flow reduction properties in an aneurysm due to a stent. *Phys. Rev. E*, Vol. 68(2):p. 021918 (2003).

intel. `http://www.intel.com/software/products/cluster/tcollector/index.htm` (2007).

Johnson, C. R. and Parker, S. G. A computational steering model for problems in medicine. In *Supercomputing '94*, pp. 540–549. IEEE Press (1994).

Johnson, C. R., Parker, S. G., Hansen, C. D., Kindlmann, G., and Livnat, Y. Interactive simulation and visualization. *IEEE Computer*, Vol. 32(12):pp. 59–65 (1999).

Jones, M. W. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, Vol. 15(5):pp. 311–318 (1996).

Kafczyk, M. *Die Gitter-Boltzmann-Methode: Von der Theorie zur Awendung*. Habiltionsschrift, Lehrstuhl für Bauinformatik, TU München (2001).

Keller, R. Personal communication (2005).

Kipfer, P. and Westermann, R. Realistic and interactive simulation of rivers. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, pp. 41–48. Canadian Information Processing Society, Toronto, Ont., Canada, Canada (2006). ISBN 1-56881-308-2.

Kolb, A. and John, L. Volumetric model repair for virtual reality applications. pp. 249–256. EUROGRAPHICS (2001).

Kollinger, M. *Definition strömungsmechanischer Randbedingungen für interaktive CFD Simulationen*. Diplomarbeit, Lehrstuhl für Bauinformatik, TU München (2007).

Krumhauer, P., Tsygankov, M., Reich, C., and Evgrafov, A. Efficient volume rendering using octree space subdivision. In *Visual Data Exploration and Analysis VI*, Vol. 3643, pp. 211–219. The International Society for Optical Engineering (1999).

Kühner, S. *Virtual Reality-basierte Analyse und interaktive Steuerung von Strömungssimulationen im Bauwesen*. PhD thesis, Lehrstuhl für Bauinformatik, TU München (2003).

Kühner, S., Rank, E., and Krafczyk, M. Efficient reduction of 3D simulation results based on spacetree data structures for data analysis in Virtual Reality environments. In *Applied Virtual Reality in Engineering and Construction*. Goteborg, Sweden (2001).

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

Lanfear, T. SR8000 concept. `http://research.ac.upc.edu/HPCseminar/SEM9900/SR8000_concept.ppt` (2000).

Leibniz Rechenzentrum München. `http://www.lrz-muenchen.de` (2007).

van Liere, R., Mulder, J. D., and van Wijk, J. J. In , pp. 696–702 (1996).

Luecke, G. and Wang, Y. Sending non-contiguous data in MPI programs. Technical Report, Iowa State University (2005).

Marcheix, L. *A 3D User Interface for a Virtual Environment*. Diplomarbeit, Lehrstuhl für Bauinformatik, TU München (2004).

McCormick, B. H., DeFanti, T. A., and Brown, M. D. Spessial issue on visualization in scientific computing. *Computer Graphics*, Vol. 21(6) (1987).

Mercury Computer Systems, Inc. `http://www.amiravis.com` (2007a).

Mercury Computer Systems, Inc. `http://www.tgs.com` (2007b).

METIS. `http://glaros.dtc.umn.edu/gkhome/views/metis` (2007).

Mezrhab, A., Bouzidi, M., and Lallemand, P. Hybrid lattice-Boltzmann finite-difference simulation of convective flows. *Computers and Fluids*, Vol. 33:pp. 623–641 (2004).

Möller, T. and Haines, E. *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA (1999). ISBN 1-56881-101-2.

MPI-Forum. `http://www.mpi-forum.org/docs/docs.html` (2007).

MPICH-G2. `http://www3.niu.edu/mpi` (2007).

Mulder, J. D., van Wijk, J. J., and van Liere, R. A survey of computational steering environments. *Future Gener. Comput. Syst.*, Vol. 15(1):pp. 119–129 (1999). ISSN 0167-739X. doi:http://dx.doi.org/10.1016/S0167-739X(98)00047-8.

Mundani, R.-P. *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*. PhD thesis, Technische Universität München (2006).

Neuhierl, B. Mehrfeldsimulation von Strömungsvorgängen, strömungsakustischen Phänomenen und Schallwellenausbreitung mit der Lattice-Boltzmann-Methode (2006). Eingeladener Vortrag, Lehrstuhlseminar, Lehrstuhl für Bauinformatik, TU München.

Noll, B. *Numerische Strömungsmechanik: Grundlagen*. Springer-Verlag (1993). ISBN 3-540-56712-7.

Norman, D. A. *The Psychology of Everyday Things*. Basic Books, New York (1988). ISBN 0-465-06709-3.

Pacheco, P. S. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996). ISBN 1-55860-339-5.

Pacx-MPI. `http://www.hlrs.de/organization/pds/projects/pacx-mpi` (2007).

Parker, S. G., Miller, M., Hansen, C. D., and Johnson, C. R. Computational steering and the SCIRun integrated problem solving environment. In *Dagstuhl '97, Scientific Visualization*, pp. 257–266. IEEE Computer Society, Washington, DC, USA (1999). ISBN 0-7695-0505-8.

Pickles, S. M., Haines, R., Pinning, R. L., and Porter, A. R. Computational steering in RealityGrid. In *UK e-Science All Hands Meeting* (2004). ISBN 1-904425-21-6.

Pohl, T., Thürey, N., Deserno, F., Rüde, U., Lammers, P., Wellein, G., and Zeiser, T. Performance evaluation of parallel large-scale Lattice Boltzmann applications on three supercomputing architectures. In *Proceedings of the IEEE/ACM SC2004 Conference (Supercomputing Conference '04, Pittsburgh, 06. - 12.11.2004)*, pp. 1–13 (2004). ISBN 0-7695-2153-3.

Qian, Y.-H., D'Humiers, D., and Lallemand, P. Lattice BGK model for Navier-Stokes equation. *Europhysics letters*, Vol. 17:pp. 479–484 (1992).

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

Sanders, M. S. and McCormick, E. J. *Human Factors in Engineering and Design*. McGraw-Hill (1993). ISBN 0-07-054901-X.

Satofuka, N. and Nishioka, T. Parallelization of lattice Boltzmann method for incompressible flow computations. *Computational Mechanics*, Vol. 23:pp. 164–171 (1999).

Schönung, B. E. *Numerische Strömungsmechanik: Inkompressible Strömungen mit komplexen Berandungen*. Springer-Verlag (1990). ISBN 3-540-53137-8.

Schulz, M., Krafczyk, M., Tölke, J., and Rank, E. Parallelization strategies and efficiency of CFD computations in complex geometries using Lattice-Blotzmann methods on high-performance computers. In *High-Performance Scientific and Engineering Computing,Proceedings of the 3rd International FORTWIHR Conference on HPSEC*, pp. 115–122 (2002).

University of Amsterdam: Section Computational Science. `http://www.science.uva.nl/research/scs/index.html` (2007).

Seidenschwarz, W. *Nie wieder zu teuer!: 10 Schritte zum Marktorientierten Kostenmanagement*. Schäffer-Poeschel Verlag, Stuttgart (1997). ISBN 3-7910-1019-0.

Shan, X. and Chen, H. Lattice Boltzmann model for simulating flows with multiple phases and components. *Phys. Rev. E*, Vol. 47(3):pp. 1815–1819 (1993).

SIEMENS AG. Interaktive, lokale Komfortsimulation in einer VR-Umgebung. Pictures of the Future, issue spring 2006 (2006).

Sloot, P. M. A., Tirado-Ramos, A., Hoekstra, A. G., and Bubak, M. An interactive grid environment for non-invasive vascular reconstruction. In *2nd International Workshop on Biomedical Computations on the Grid (BioGrid'04) in conjunction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*. Chicago, Illinois, USA (2004). ISBN 0-7803-8431-8.

Standish, R. Introduction to high performance computing. `http://www.ac3.edu.au/edu/hpc-intro/` (2006).

Stolte, N. and Kaufman, A. Novel techniques for robust voxelization and visualization of implicit surfaces. *Graph. Models*, Vol. 63(6):pp. 387–412 (2001). ISSN 1524-0703.

Succi, S. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press (2001). ISBN 0-19-850398-9.

Succi, S., Amati, G., and Benzi, R. Challenges in lattice Boltzmann computing. *Journal of Statistical Physics*, Vol. 81:pp. 5–16 (1995).

Sudhir, A. and Kesavadas, T. Computational steering of manufacturing steering using virtual reality. In *ICRA '00: International Conference on Robotics and Automation*, Vol. 3, pp. 2654–2658. San Francisco, CA, USA (2000). ISBN 0-7803-5886-4.

Tamaki, Y., Sukegawa, N., Ito, M., Tanaka, Y., Fukagawa, M., T.Sumimoto, and N.Ioki. Node architecture and performance evaluation of the hitachi super technical server SR8000. In *12th International Conference on Parallel and Distributed Computing Systems*, pp. 487–493 (1999). ISBN 1-880843-9-3.

The Stanford 3D Scanning Repository. `http://graphics.stanford.edu/data/3Dscanrep` (2007).

Thürey, N. *Physically based Animation of Free Surface Flows with the Lattice Boltzmann Method*. PhD thesis, University of Erlangen-Nuremberg (2007).

Thürey, N. and Rüde, U. Free surface Lattice-Boltzmann fluid simulations with and without level sets. In *MPI for Computer Science: VMV 04 Proceeding*, pp. 199–208. Max Planck Center for Visual Computing and Communication (2004).

Thürey, N. and Rüde, U. Technical report on turbulent free surface flows with the Lattice Boltzmann method on adaptively coarsened grids. Technical Report, Lehrstuhl für Informatik 10 (Systemsimulation), Friedrich-Alexander-Universität Erlangen-Nürnberg (2005).

Thürey, N., Rüde, U., and Körner, C. Interactive free surface fluids with the Lattice Boltzmann method. Technical Report, Lehrstuhl für Informatik 10 (Systemsimulation), Friedrich-Alexander-Universität Erlangen-Nürnberg (2005).

Tölke, J. *Gitter-Boltzmann-Verfahren zur Simulation von Zweiphasenströmungen*. PhD thesis, Lehrstuhl für Bauinformatik, TU München (2001).

Tölke, J. Modellvergleich Surfwelle: Lattice-Boltzmann Methoden. In *Berichte des Lehrstuhls und der Versuchsanstalt für Wasserbau und Wasserwirtschaft*, 104, p. 260 (2006).

Tölke, J., Freudiger, S., and Krafczyk, M. An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations. *Computers and Fluids*, Vol. 35(8-9):pp. 820–830 (2006).

TOP500. `http://www.top500.org` (2007).

van Treeck, C. *Gebäudemodell-basierte Simulation von Raumluftströmungen*. PhD thesis, Lehrstuhl für Bauinformatik, Technische Universität München (2004).

van Treeck, C., Rank, E., Krafczyk, M., Toelke, J., and Nachtwey, B. Extension of a hybrid thermal lbe scheme for Large-Eddy simulations of turbulent convective flows. *Computers and Fluids*, Vol. 35:8-9:pp. 863–871 (2006).

van Treeck, C., Wenisch, P., Borrmann, A., Pfaffiger, M., Wenisch, O., and Rank, E. ComfSim - Interaktive Simulation des thermischen Komforts in Innenräumen auf Höchstleistungsrechnern. *Bauphysik*, Vol. 29(1):pp. 2–7 (2007).

Wellein, G., Lammers, P., Hager, G., Donath, S., and Zeiser, T. Towards optimal performance for lattice boltzmann applications on terascale computers. In *Parallel Computational Fluid Dynamics: Theory amd Applications, Proceedings of the 2005 International Conference on Parallel Computational Fluid Dynamics*, pp. 31–40 (2006).

White III, J. B. and Bova, S. W. Where's the overlap? An analysis of popular MPI implementations. In *MPI Developers Conference* (1999).

wikipedia. `http://en.wikipedia.org/wiki/Computational_fluid_dynamics` (2007a).

wikipedia. `http://en.wikipedia.org/wiki/Lattice_Boltzmann` (2007b).

wikipedia. `http://en.wikipedia.org/wiki/STL_(file_format)` (2007c).

wikipedia. `http://en.wikipedia.org/wiki/Symmetric_multiprocessing` (2007d).

Wilke, J., Pohl, T., Kowarschik, M., and Rüde, U. Cache Performance Optimizations for Parallel Lattice Boltzmann Codes. Lecture Notes in Computer Science (LNCS) Vol. 2790 *In* Proc. of the EuroPar-03 Conf., pp. 441–450. Springer (2003).

Wolf-Gladrow, D. A. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*. Springer-Verlag (2000). ISBN 3-540-66973-6.

Wössner, U., Becker, M., and Lang, U. Tangible interfaces for interactive flow simulation. In *The 2nd Russian-German Advanced Research Workshop on Computational Science and High Performance Computing* (2005).

Zou, Q. and He, X. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, Vol. 9(6):pp. 1591–1598 (1997).

Acknowledgments

This thesis 'emerged' during my work as a scientific assistant at the chair for Bauinformatik at the TU München and in cooperation with the Leibniz Computing Center, Munich. My research project was funded by the Competence Network for Technical, Scientific High-Performance Computing in Bavaria (KONWIHR). In addition, the HPC Europe program supported a research visit of six weeks to the University of Amsterdam, Netherlands.

First of all, I would like to thank my promotor Prof. Ernst Rank who gave me the opportunity to work on the topic of computational steering and writing my thesis at his institute. He has always been a challenging and, therefore, constructive discussion partner and supported me by giving me the freedom to intensify my interests in various areas. I also want to express my gratitude with regard to having been able to visit many international conferences, to meet numerous interesting people, and to keep myself updated to recent devolpments in my research field. Besides research, Prof. Rank gave me the opportunity to further develop myself with respect to my teaching abilites, supervision of students as well as organizing an Elite Masters program and several other duties to smooth the way to a successful and efficient Postdoc period.

I owe many thanks to Prof. Ulrich Rüde, my thesis' second examiner. He inspired me — especially in the last one to two years of my work — in which direction I would like to develop myself. He also gave me interesting insights into computer science research, which has become a main aspect of my thesis. In particular, I would like to thank him for arranging the contact to Prof. Peter Sloot's group in Amsterdam.

I also want to express my gratitude to Prof. Peter Sloot and Alfons Hoekstra for hosting me during my research visit in Amsterdam at Prof. Sloot's Institute for Computational Science and for giving me the opportunity to learn so many things from a number of interesting people in his group.

I would like to thank all the people who supported me during the time of learning Lattice-Boltzmann theory. Foremost, I need to refer to Alfons Hoekstra, who explained to me *the whole story of the LBM theory* — and everything that was left was *just algebra*. Jos Derksen, Jonas T"olke, and Thomas Zeiser have also been very patient and helpful in answering my questions.

For their very kind support of my work, which caused extra effort especially due to the interactive way of using supercomputers, I would like to thank Oliver Wenisch, Reinhold Bader, Helmut Heller, Irene Geiseler, Leonhard Scheck and Matthias Brehm at the Leibniz Computing Center in Munich, as well as Willem Vermin, Laura Leistikov and Huub Stoffers at the Sara Computing Center in Amsterdam and Thomas Zeiser, Georg Hager, and Gerhard Wellein at the Computing Center in Erlangen. I also want to thank Timothy Lanfear for the insight and discussions on Hitachi's supercomputer features and many clues on how to program efficent code on this machine.

I have always enjoyed working with my students: Daniel Alfreider, Miha Gantar, Laurent Marcheix Nikola Cenic, Christian Liefhold, and Michael Kollinger,

and I am thankful for their support in teaching and implementing.

Thanks to all my collegues who were responsible for the friendly working atmosphere at our institute. I especially want to emphazise the true friendship I enjoyed (and still enjoy) with my roommate Uli Heisserer. Special thanks I would like to express to Christoph van Treeck, who has supported me as my group leader and has shared his experience in running LBM simulations.

To enable the modeling and running of the simulations of the operating rooms at the Klinikum rechts der Isar, Rainer Burgkart lead me through the rooms and demonstrated the inportance of the installed ventilation systems. I would like to thank him for his enthusiasm, great support, and his interest in my work.

Although there have been many people who supported me, the most important backing has been my husband Oliver. He was my main cooperation partner at the Leibniz Computing Center, always on the spot to help me with hardware problems and to take care of the newest software updates. He also supported me during implementing, was an excellent discussion partner and my most discerning and constructive lector.

List of Publications

1. Wright, H., Crompton, R. H., Kharche, S., and Wenisch, P.: *Steering and Visualization: Enabling Technologies for Computational Science*. Future Generation Computer System (submitted).

2. Wenisch, P., van Treeck, C., Scheck, L., and Rank, E.: *Computational Steering: Interactive Flow Simulation in Civil Engineering*. Inside, Vol. 5(2) (2007).

3. van Treeck, C., Wenisch, P., Borrmann, A., Pfaffinger, M., Egger, M., and Rank, E.: *Computational Steering of Thermal Comfort Perception*. In 2nd GACM Colloquium on Computational Mechanics. TUM, Munich, Germany (2007).

4. van Treeck, C., Wenisch, P., Borrmann, A., Pfaffinger, M., Egger, M., and Rank, E.: *Utilizing High Performance Supercomputing Facilities for Interactive Thermal Comfort Assessment*. In Proc. 10th Int. IBPSA Conference Building Simulation. Bejing, China (2007).

5. Wenisch, P., van Treeck, C., Borrmann, A., Rank, E., and Wenisch, O.: *Computational Steering on Distributed Systems: Indoor Comfort Simulations as a Case Study of Interactive CFD on Supercomputers*. International Journal of Parallel, Emergent and Distributed Systems, Vol. 22(4):pp. 275–291 (2007).

6. Wenisch, P.: *Interactive Fluid Simulations: Computational Steering on Supercomputers*. In Science and Supercomputing in Europe - report 2006: pp. 453-460 (2007). ISBN 978-88-86037-19-8

7. van Treeck, C., Wenisch, P., Borrmann, A., Pfaffinger, M., Wenisch, O., and Rank, E.: *Comfsim interaktive Simulation des thermischen Komforts in Innenräumen auf Höchstleistungsrechnern*. Bauphysik, Vol. 29(1):pp. 2–7 (2007).

8. van Treeck, C., Wenisch, P., Borrmann, A., Wenisch, O., Kuehner, S., Toelke, J., Krafczyk, M., and Rank, E.: *Computational Steering of Lattice-Boltzmann based CFD Simulations in Virtual Reality (hlrb i)*. In Research projects HLRB I (Hitachi SR8000) (2006).

9. van Treeck, C., Wenisch, P., Borrmann, A., Pfaffinger, M., Egger, M., Wenisch, O., and Rank, E.: *Towards Interactive Indoor Thermal Comfort Simulation* In Proceedings of ECCOMAS CFD 06, European Conf. on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands (2006).

10. van Treeck, C., Wenisch, P., Borrmann, A., Pfaffinger, M., Wenisch, O., and Rank, E.: *Comfsim interaktive Simulation des thermischen Komforts in Innenraeumen auf Hoechstleistungsrechnern*. In Tagungsband BauSIM 2006, pp. 205–207. IBPSA Germany, München, Germany (2006). ISBN 978-3-00-019823-6.

11. Borrmann, A., Wenisch, P., van Treeck, C., and Rank, E.: *Collaborative Computational Steering: Principles and Application in HVAC Layout*. Integrated Computer-Aided Engineering (ICAE), Vol. 13(4):pp. 361–376 (2006).

12. Wenisch, P., Wenisch, O., and Rank, E.: *Harnessing High-Performance Computers for Computational Steering*. In Lecture Notes in Computer Science: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Vol. 3666, pp. 536–543. Springer (2005).

13. Borrmann, A., Wenisch, P., van Treeck, C., and Rank, E.: *Collaborative HVAC Design using Interactive Fluid Simulations: A geometry-focused Collaboration Platform*. In Proceedings of the 12th International Conference on Concurrent Engineering. Fort Worth, Texas, USA (2005).

14. Wenisch, P., Borrmann, A., Rank, E., van Treeck, C., and Wenisch, O.: *Collaborative and Interactive CFD Simulation using High Performance Computers*. In 18th Symposium AG Simulation (ASIM) and EuroSim, pp. 145–151. SCS Publishing- House e.V. Erlangen, Erlangen, Germany (2005). ISBN 3-936150-41-9.

15. Rank, E., Borrmann, A., Duester, A., Niggl, A., Nuebel, V., Romberg, R., Scholz, D., van Treeck, C., and Wenisch, P.: *From Adaptivity to Computational Steering: The long Way of Integrating Numerical Simulation into Engineering Design Processes*. In ADMOS 2005. CIMNE, Barcelona, Spain (2005).

16. Wenisch, P., Wenisch, O., and Rank, E.: *Optimizing an Interactive CFD Simulation on a Supercomputer for Computational Steering in a Virtual Reality Environment*. In High Performance Computing in Science and Engineering, pp. 83–93. Springer (2005).

17. Borrmann, A., Wenisch, P., van Treeck C., and Wenisch, O.: *Eine verteilte Architektur fuer synchrones kooperatives Arbeiten mit einer interaktiven Stroemungssimulation*. In 16. Forum Bauinformatik. Shaker Verlag, Aachen, Germany (2004). ISBN 3-8322-3233-8.

18. Wenisch, P., van Treeck, C., and Rank, E.: *Interactive Indoor Air Flow Analysis using High Performance Computing and Virtual Reality Techniques*. In 9th International Conference on Air Distribution in Rooms (RoomVent2004). Coimbra, Portugal (2004).

19. Wenisch, P. and Wenisch, O.: *Fast Octree-based Voxelisation of 3D Boundary Representation-Objects*. Technical Report, Lehrstuhl fuer Bauinformatik, Technische Universität München (2004).

20. Wenisch, P.: *Kopplung von Hochleistungsrechner und Virtueller Realitaet*. KONWIHR Quartl, Vol. 37 (2003).

21. Hardt, P., Kuehner, S., Rank, E., and Wenisch, O.: *Interactive CFD Simulations by Coupling Supercomputers with Virtual Reality*. Inside, Vol. 1(2):pp. 12–13 (2003).

22. Hardt, P., Kuehner, S., Wenisch, O., and Rank, E.: *Interactive CFD Simulation by Coupling Supercomputers with Virtual Reality*. In High Performance Computing in Science and Engineering. Springer (2004). ISBN 3-540-44326-6.

23. Kuehner, S., Hardt, P., Krafczyk, M., and Rank, E.: *Computational Steering of a Lattice-Boltzmann based CFD-solver in Virtual Reality*. In Conference on Construction Applications of Virtual Reality. Virginia, USA (2003).

24. Hardt, P. and Crouse, B.: *Präprozessor für einen computergestützten Windkanal*. Fortschritt-Berichte Vol. 4 In 14. Forum Bauinformatik, pp. 165–172. VDI Verlag, Bochum, Germany (2002). ISBN 3-18-318104-5.

25. Hardt, P.: *Entwicklung eines Moduls zur Definition von stroemungsmechanischen Randbedingungen als Attribute von 3D CAD-Geometrien*. Diploma thesis, Lehrstuhl fuer Bauinformatik, TU Muenchen (2001).