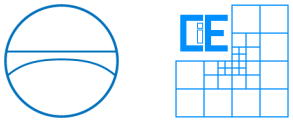


Faculty of Civil, Geo and Environmental Engineering
Chair for Computation in Engineering
Prof. Dr. rer. nat. Ernst Rank

Interactive Exploration and Computational Steering in CAVE-like Environments for High-Performance Fluid Simulations

Uğurcan Sarı

Master's thesis
for the Master of Science program Computational Mechanics

Author:	Uğurcan Sarı
Matriculation Number:	03693955
Supervisor:	PD Dr. rer. nat. habil. Ralf-Peter Mundani
	Christoph Ertl, M. Sc
Date of issue:	01. May 2019
Date of submission:	01. November 2019

Abstract

Numerical simulations take an important role in research and applications today. After each simulation has completed in order to change some simulation parameters, the simulation steps are repeated from the beginning which costs time, money, and effort. Another challenge in numerical simulations is exploring the outcome data. Today there are numerous tools to process and visualize the data computed in simulations. However, the best approach to present the simulation results to the end user is an ongoing research topic.

This thesis consists of an implementation of extending a high performance computing computational fluid dynamics code from the traditional simulation pipeline of mesh-compute-visualize to a computational steering state. Furthermore, it includes a porting of the visualization to a virtual reality environment with the details of implementation.

Computational steering is an approach to affect the simulation parameters while the simulation is still running. This provides the researchers the possibility of altering the simulation parameters without starting the simulation all over, therefore, increasing the time efficiency. Through this work, the global steering commands such as start, dump checkpoint, pause and continue as well as parameters that effect the result and change the initial configurations such as add-modify-delete geometry or boundary conditions are implemented.

In addition to this, a visualization front end is developed for the purpose of working in an immersive virtual reality environment. Therefore, the user is able to visualize the outcome during runtime and in stereo view inside this virtual reality environment. The user of the front end in the virtual reality environment also holds the possibility of changing the domain of interest which results in changing the data size requested from the fluid dynamics code. The visualization in virtual reality is adjusted to take the user inside the domain of interest to give the feeling of living in the simulation.

Preface

While finishing my journey to achieve completing the Master of Science degree at Technical University of Munich, it is now a perfect chance to acknowledge and to thank those who helped me, supported me, and trusted me. Without them this degree would never be possible. Therefore, I would like to express my thoughts and feelings to them.

Firstly, I would like to thank to PD Dr.rer.nat. habil. Ralf-Peter Mundani. His excellent lectures, which were never boring, not just kept me interested in engineering but also kindled a strong interest in computational sciences. This thesis that he provided, something I believe I will call a corner stone in my following years. He was not only great in supervising the thesis, but his door was always open for any problem. He was always there to discuss anything from future plans to career decisions. I was so lucky that to be supervised by him. Thank you Prof. Mundani and good luck in your new position as a full time professor.

I would like to thank Christoph Ertl, M.Sc, for supervising my thesis. He was never bored of my not-so-clear questions and tried to give every possible answer. He was always so quick to provide all the necessary information as well as the support for the separate parts and the general organization of the thesis.

Additionally, I would like to thank to Jutta Dreer and Dr. Thomas Odaker from Virtual Reality and Visualization Center of LRZ. They were always ready to help for any problems related to CAVE. They have never shown any negative impression to any question I asked.

A huge thanks goes the whole Open Source community, who extended my horizon. With this I would like to thank specially to Vladimír Vondruš(mosra) for his huge patience and his valuable library, Magnum. As a person whose background is not computer graphics, without Vladamír my learning curve might have been much steeper.

I would like to thank to my company Spanflug Technologies for putting no pressure on my shoulders, providing a warm and comfortable workspace that is helping me to keep myself relaxed for my thesis, and helping me getting integrated for the German life and work culture.

The beginning of this thesis was a huge step out from my comfort zone. The Computational Mechanics program did not just provide me a great self-development opportunity, but also great friends who I would like to thank here.

My colleague Oğuz Öztoprak, who is probably at this very moment proof reading my thesis, deserves a big gratitude. He almost weekly asked about my progress. In his "tea time"

invitations we discussed some methods as well as some bottleneck problems of this work.

I would like to thank to Oğuzhan Karakaya and Berkay Alp Çakal. In the beginning of the thesis, when I was not even sure of the platform I would use, their help motivated me a lot. They were also my first visitor in CAVE to judge my work. Their opinions improved this work so much.

Three of the great people TU Munich gave to me, Basser Saridar, Olivér Schwahofer and Mahmoud Ammar, are three "battery rechargers" of this work. Their supports and energies kept my mood and motivation always at the top level.

There are many more friends, who I did a lot of brainstorming with, who helped with proof readings, who were there to stress out, and who were there to debug with. Mert Göldeli, Berkay Tutal, Saaransh Saxena, Alexander Broad, Nicolau Reis, Guillermo Barraza, Mehmet Akif Ortak, Kunter Özbay, Gizem Sacıhan, Selvin Selvi, and Pınar Ece Mutlu are just some of those who should not be forgotten. Thank you my friends.

I would like to express my warm feelings to my partner, Charlotte, who was there to support me in my most stressful moments. She was always so understanding about the great amount of time this thesis required.

Last but not least, there is no word to describe my thoughts about my family. They have been under no condition supporting every act of mine. Without their continuous support and love, any of my achievements would be impossible. Thank you.

1 November 2019, München
Uğurcan Sarı
ugurcan.sari@tum.de

Contents

Chapter 1

Introduction

1.1 Motivation

For centuries, scientists have been conducting experiments in order to understand nature. However, the improvements in mathematics with the combination of necessary computing power created another opportunity for these wandering minds: Numerical simulations. Compared to an experimental methodology, a numerical methodology has several advantages. It is probable that conducting an experiment may be more expensive than numerically solving the problem. In some research topics, conducting experiments could be ethically wrong (for example human based experiments). It is also possible that experiments may be very dangerous (e.g. explosions).[Kne13]

Today, using numerical simulations to predict the behaviors of solids or fluids is quite common. In many academic and industrial application simulations, the three basic steps exist, which are preprocessing, computation, and post-processing. Preprocessing is defined as the phase of preparing the model for solving, which involves domain decomposition and meshing. [ATW+19]. Post-processing can be defined as extracting the information from the data. After the exploration of the results of the simulation, if it is decided to modify the simulation, then the simulation steps are repeated from the beginning.

Worth noting about numerical simulations is that they can be very expensive, time-consuming, and resource-dependent. There can also be parameters now included, that were previously excluded for the purpose of simplicity, due to limitations in computational power, thus, no improvement in computing speed may be observed. If a simulation is running with wrong parameters, it is not always easy to detect while the simulation is running until the last step which is post-processing. At that point, the person conducting the research has to do everything almost from scratch. First checking the geometry to see if geometrical primitives are correctly represented, then if a mesh exist, the mesh. Simulation parameters, as

well as initial conditions, should be checked once more. With clicking the button of 'start simulation', the researcher again has no choice but to wait until the post-processing. The problem with the time time taken to run the simulation is a topic of many researchers today. People use high-performance computing (HPC) methods in order to decrease the amount of time spent on simulations as well as increasing the memory-bound that simulations create.

Another emerging technology today is virtual reality (VR) technology. With this technology, people experience a world, that does not exist but created by some other person using computer resources.

This work combines the topics and technology discussed above. An HPC code is extended to enable computational steering. Furthermore, a steering program is developed to provide an interface for the user to process inputs such as configurations and geometrical entities as well as to visualize the results to explore the simulation in detail by using a novel technique, so-called sliding window. Finally, the view is ported to the immersive virtual reality environment located at Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ).

1.2 Outline

The current work is structured into two main parts:

The first part details the fundamental information about previous work that was used as the basis of this work. The highlights that are necessary for understanding such as data structures are given. Then the different parts of the code such as preprocessing, computation, observing, and communication are investigated. Later on, the state of the art of computational steering is presented. The different types of steering are explained in detail. The use cases of computational steering in different branches of science are provided. Lastly, the current status of virtual reality is reviewed. The problems of the virtual world are discussed. The CAVE Automatic Virtual Reality Environment is introduced. The auxiliary hardware that supports virtual reality is explained. Lastly, the hardware in LRZ is briefly discussed.

In the second part, the implementation is explained in detail. Firstly, the changes in the main scripts of the code are explained. It is followed by the implementation of the computational steering, and how the data flow is handled. The scientific visualization pipeline is described with the example of a journey from raw data to rendered streamline. The separate parts of the visualization code are explained. The implemented visualization methods are shown. The extension of the code from 2D to 3D in the VR environment is briefly described. Lastly, the manipulation of the output scene according to the sliding window is shown.

High-performance computing, visualization, virtual reality, software development, computational fluid dynamics (CFD) are all separately very large topics. While this thesis unites all of them, there are some new future possible implementations that have arisen. In the final part, the outlook, these points are given to the reader, or the successor so as to provide some ideas for possible avenues of further research in this field.

Chapter 2

Fundamental concepts

Currently, the in-house code that is used in this work contains a passive visualization which means the data can be visualized during run-time however, no interaction and parameter tuning of the simulation can be performed while the simulation runs. The purpose of this work is to enhance the current in-house code from passive visualization to computational steering and transfer the visualization into the CAVE. To do so, numerous libraries are used in the current HPC code. Herein, computational steering, the main HPC fluid code, and CAVE are explained.

2.1 Computational Steering

Computational Steering is defined as visualizing data while the simulation runs, and affecting the simulation so that it will be altered in its future [LM00]. According to many, it is harder to implement, yet providing scientists the opportunity of exploring the simulations with the altered parameters immediately without going through all the steps of the simulation pipeline.

2.1.1 Classifications of Computational Steering Systems

Many different computational steering systems have been proposed over the years. While many of these systems vary heavily from each other, these systems can be differentiated under 7 different titles, according to Knežević[Kne13]: user interfaces, the aim of steering, the type of steering, level of interactivity, support for distributed computing, support for multi-component and collaborative steering and utilization method.

User interfaces

User interfaces may vary in four different ways: Command-line interface(CLI), visual programming interfaces, graphical user interfaces(GUI), and immersive interfaces.

The command-line interface is the most primitive one, which lets the user interact through the command line or from written text files. The visual programming languages let the user interact through adding graphical primitives such as boxes and lines, and connecting them together on a graphical layout.

Another type of interface is GUIs. In GUIs the interaction is provided through sliders, text boxes, tables and more. It is also possible to have a command line in a GUI for direct interaction with the lower-level parameters.

The last type of interaction is interacting through an other hardware such as head tracker. The sensors are placed on the user. When the user moves, the sensor displacements are tracked using cameras. The direct reaction is shown in the program. The CAVE is a great example of this kind of interaction.

Aim of Steering

There are two types of steering in this subtopic: human-interactive steering and performance steering.

The user steers the algorithm according to the exploration of the interest in human-interactive steering. Performance steering is to improve the performance of the simulation. [VS97] An example of performance steering is load balancing, which is to distribute the load as equal as possible to computing components.

Type of Steering

There are two types of steering: Global steering, and application-level steering. Global steering refers to interaction with the whole program by starting, stopping, pausing, restarting, etc.

The application steering is more of what the user is interested in. Modifying some parameters, adding new entities, modifying entities, or deleting entities, altering the domain boundaries are some of what the user can apply to the simulation.

Level of Interactivity

The interactivity level can be categorized in two: *stop-and-go* and *on-the-fly*.

In the so-called *stop-and-go* type simulations, before interacting, the simulation is stopped or paused. The new configurations are sent in when the simulation stopped or paused. Then the simulation is run again.

On the contrast, *on-the-fly* offers instant changes in the simulations without any stops. The simulation is always ready for some new orders to do changes in simulation's configurations.

Support for Distributed Computing

The computational steering application may be written for working in a local computer or permits the user to work on cloud systems or distributed grid systems.

Support for Multi-Component and Collaborative Steering

Running and steering multiple simulations at once is also a point of interest. These kinds of applications may also require more than one agent of visualization. These kind of applications are really important in multidisciplinary research while they may come up with challenges of synchronization and data sharing policy.

Utilization Method

The aim of the simulations is to find solutions for real-life problems. Therefore, the center of the simulation should be the experts who are analyzing the results rather than the simulation steps. The parts of meshing, solving, and visualizing, and especially, the steps involving the development of the methods, might be hidden from them.

Problem Solving Environments(PSEs) are such tools that allow the experts to stand at the center without knowing the algorithmic structures. [MvWvL99] PSEs are mostly a whole program for both development and application, however, out of interest for the user, some lower level functionalities can be kept hidden. [KMR+12]

Besides these tools, there are many light-weight tools such as frameworks, wrappers, middle-layers and libraries out in the world supporting computational steering.

2.1.2 Applications of Computational Steering

The biggest advantage of using simulations can be producing results faster, while always having the chance of altering the parameters. The opportunity of altering parameters instantly in runtime makes computational steering appealing due to the fact that there is no time needed for starting everything all over. Many areas of science and engineering are trying to do research and development in computational steering environments.

Biomedical Applications

Biomedical applications of simulations are very important due to the fact that in most cases the patient is alive, therefore, there is no chance of experimentation. While the required time of the simulation may get longer and longer depending on the problem's complexity, the possibility of different simulations is numerous, resulting in covering the most abnormal and rare cases.

In [JP95], Johnson introduces the code, SCIRun, and its applications in computational medicine. In [JB18], Janson and his colleagues develop an interactive, and specific to the patient series of calculations in order to simulate the results sourcing from the deep brain stimulation in almost real-time. The simulation also provides feedback to give information about how the devices influence the brain's neural activity.

Physics Based Simulations

Computational steering has been a good research topic for physics-based simulations such as computational fluid dynamics, nuclear engineering, and manufacturing.

In CFD the opportunity of changing the domain as well as altering some parameters such as adding/deleting new objects to/from the computational domain while keeping the developed flow field almost the same(grid, boundary condition, and geometry alterations may create disturbance) allures many researchers to computational steering. The applications of CFD are widespread from atmospheric modeling to complex flows. In [WvTB+07], Wenisch explains indoor comfort simulations on supercomputers with the application of computational steering. The code works on the distributed memory system, and is able to steer the simulation in an on-the-fly fashion.

In order to define efficient evacuations ways, the evacuation planners may use pollution modeling to simulate danger. In [MvLvW98], using parameters such as emission fields, meteorological and geographical parameters, the smog prediction is done over Europe. The calculations are conducted through 4 layers: the surface layer, the mixing layer, the reservoir layer, and the upper layer. The steering is done by examining different atmospheric layers and emission sources.

Computational steering is also employed in the area of nuclear research. At the University of Illinois at Urbana-Champaign (UIUC), a code was developed in order to steer the application of nuclear reactor simulation. [KU07] The code is a web-based code and to create the front end and for web-casting, a commercial code was employed. The code provides the option of steering the simulation parameters such as temperature through the website.

Civil engineering and Architecture

Much research in the areas of civil engineering and architecture is conducted using computational steering approach([WvTR04],[PvTB+07],[BWvTR06],[BWFR05]). The indoor air quality assessments are quite valuable in this area. Heating, ventilation, and air conditioning(HVAC) simulations used to determine room air qualities are a type of wide application. Researches, while trying to find the cheapest solution for the best air quality; apply, modify, and delete the objects and boundary conditions interactively after visual and numeric investigation. These kinds of simulations can get so detail-oriented that different metabolisms, as well as clothes, may be taken into account.

2.2 High-Performance Computing

High-Performance Computing(HPC) can be defined as using computing systems that have much more computing resources as well as memory resources compared to personal devices in order to solve problems. In June 2019, the most powerful supercomputer according to TOP500[1] is in DOE/SC/Oak Ridge National Laboratory, Oak Ridge, the United States with 2,414,592 cores and 148,600 Tera floating-point operations per seconds(TeraFLOPS). The supercomputer is a computing system that refers to the best computer available. The application areas of the high-performance computing in computational steering are numerous from mathematics to finance.

Many application codes that run on supercomputers use algorithms that can work in a parallel architecture. Parallel computing is executing of same or different instructions simultaneously. Depending on the resources, it increases the computing speed dramatically and is a great tool for handling complex problems.

2.3 Massive Parallel Fluid Code

The basis code behind this work is the *Massive Parallel Fluid Code*(which will be referred to as mpFluid through this work) which was developed by Jérome Frisch in his Ph.D. work titled "Towards Massive Parallel Fluid Flow Simulations in Computational Engineering".[Fri14] The code is based on an adaptive, hierarchic, block-structured, orthogonal, Cartesian grid structure. This grid structure is the key point of traveling through the different scale length for both the multigrid-like method(which is beyond the scope of this work) that the code employs and the sliding window approach for visualization. In the code, an explicit time discretization scheme with the second-order Adam Bashforth method was implemented. For thermal coupling, Boussinesq approximation was employed. The current code behind this thesis' solves laminar Navier-Stokes equations.

2.3.1 General Overview of the mpFluid Parts

The mpFluid code can be divided into 3 main parts; computing process(es), neighborhood server(s), and sliding window server. The parts communicate with their necessary partners through the MPI library. The parts are explained assuming the code runs for n number of processes.

Computing processes: The ranks of the computing processes alter from [0, m-1]. Their main job is to solve the Navier-Stokes equations. The master process(rank 0) is responsible

[1] www.top500.org

for also creating the computational domain including reading geometry files and processing them. After the domain is generated the boundary conditions are applied to the corresponding cells. Then it broadcasts all domain data. The domain is distributed within the computing processes. Every computing process registers its grids to its corresponding neighborhood server. Each of the processes checks if there is any query from its neighborhood server, such as grid migration. The master process has also the duties of cleaning the task queue of the neighborhood server and reporting the user proper log messages.

Neighbourhood servers: The neighborhood servers are implemented to have a full overview of the computing processes. Their ranks are [m, n-2]. Computing processes require data exchange and load balancing, the neighborhood server is there to do this. The neighborhood server(s) is also there for the outer communication which is used for sliding window collector. The orders are sent through or from the neighborhood server to the computing processes. The neighborhood server also keeps the information about locations of the cells, as well as the topological connection between parent-child grids.

Sliding window collector process: The sliding window collector process(SWCP) is a socket for outer communication. It always occupies the last node, n. The main duty of this node is to get requests from a client with a certain size of a bounding box and returning back the cell data in an unstructured grid. It is able to communicate with the neighborhood server using MPI protocol. The sliding window also has a Transmission Control Protocol (TCP) output in order to communicate with third party codes or programs. In the paper [MFVR18] the client-side was chosen as Paraview.

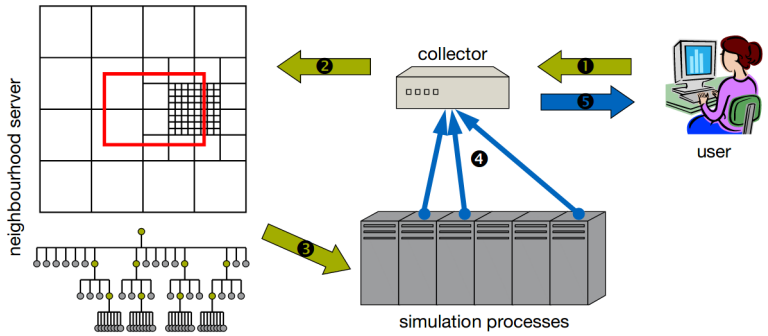


Figure 2.1: The sliding windows working process of the mpFluid.[FMR13]

The overall communication scheme for receiving data using the SWCP is shown in the figure 2.1. The user defines in the third party program the bounding box of interest while the simulation is running. The desired bounding box definition is transferred to the sliding window collector process through TCP. The sliding window collector sends it to the neighborhood server using MPI communication. The neighborhood server converts the inquiry into a neighborhood task and sends it to the computing processes. Computing processes prepare the data according to the size of the bounding box, pack the data and send it directly to the

sliding window collector process using MPI. And finally, the sliding window collector process sends the data to the client which applies some post-processing methods such as streamline generation to have an idea of how the simulation runs at the back end.

2.3.2 Data Structure

The data structure was designed especially for the parallel running environment. There exist two types of grids: logical grids(l-grids) and data grids(d-grids). Each l-grid is connected to a d-grid. ld-grid is used as a term in order to refer one l- and of d-grid pair.

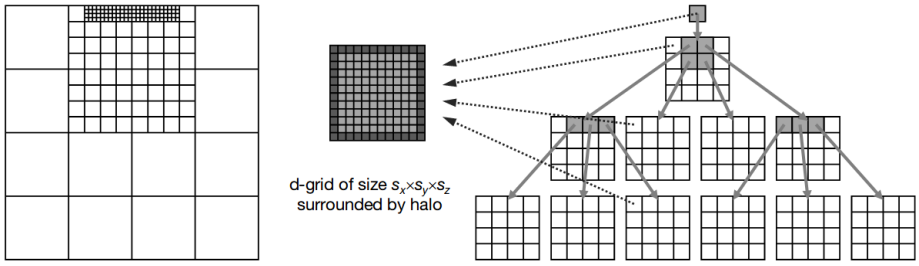


Figure 2.2: Hierarchical Data Structure: l-grid structure without d-grids(right), l-grid construction scheme(left), d-grids that are in all l-grids(middle)[FMR13]

In figure 2.2 the data structure can be seen. On the right figure, one may see the l-grid structure. The l-grids are are refined with respect to the directions desired. The child l-grids are refined until it reaches to the desired depth. Each l-grid can have just one parent grid because of the fact that each l-grid are a production of its parent. In the left figure, one can see the fine refined and coarse refined sections clearly for l-grid.

d-grids store the flow attributes of cells such as pressure, density, or pressure. d-grids are surrounded by ghost cells which can be seen in the figure in the middle.

2.4 Virtual Reality

"*A set of images and sounds, produced by a computer, that seem to represent a place or a situation that a person can take part in*" is how virtual reality(VR) is defined in the Cambridge Dictionary [Pre08]. The aim of VR is to immerse the user in the virtual world and to make the user interact with it. The goal is to increase this immersion as much as possible. In order to do so, a proper VR hardware with a responsive 3D environment is used.

According to [KOU03] there are four crucial technologies for VR:

[2]http://www.paraview.com

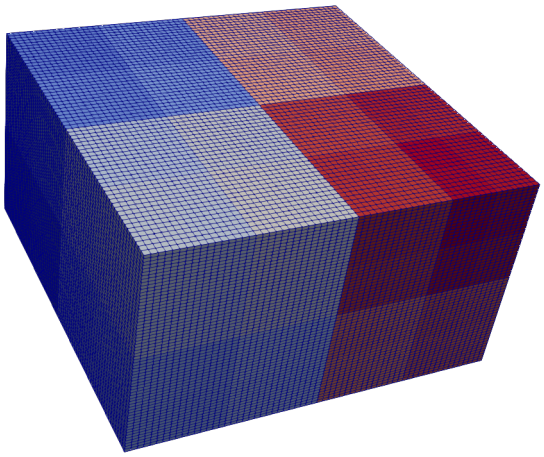


Figure 2.3: Capture of the discritisized domain from Paraview[2]. Smallest same colored grids are ld-grids. The separate colour blocks are parent l-grids..

- The visual output that takes the user into the VR world, and giving feed back to the user according to users and environments reactions.
- A rendering pipeline for stereo images with 20-30 frames per second (fps) rate for each eye.
- A proper tracking system.
- A database for the back end system to maintain the 3D world model.

In order to have the feeling of 3D there have to be some physical and psychological hints in order to change the perception. Some of these hints are:

- Two parallel lines converging to a point in infinity,
- Objects under the shade are further than the objects shading them,
- The motion of the object that is closer to the user is faster than the one that is further.

A method used to reach the 3D feeling is to generate stereo images. The purpose of this to generate separate images of the same object for the left and the right eyes individually. The human brain reconstructs the combination of these two images as if there is a depth in it.

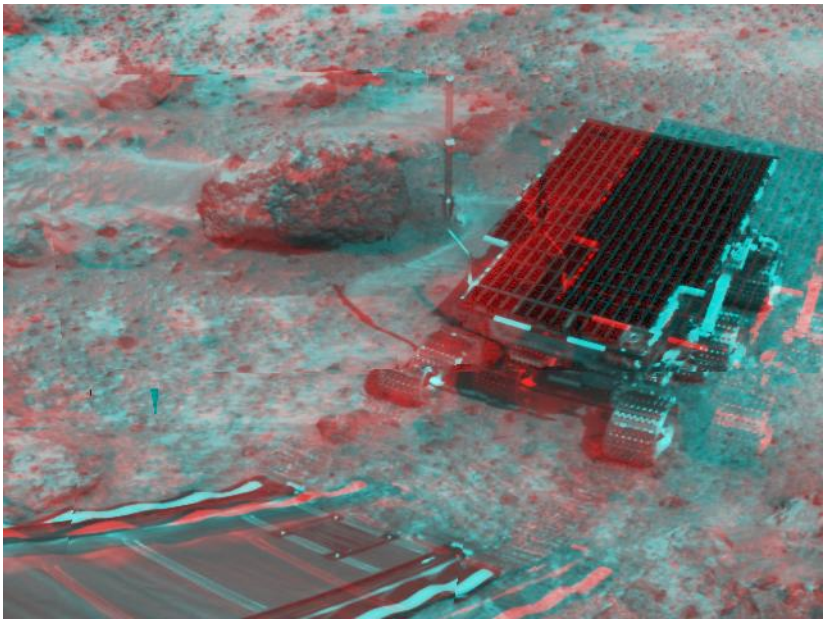


Figure 2.4: Stereo Image taken of the Mars Rover [NAS97]

2.4.1 Cave Automatic Virtual Environment(CAVE)

Cave Automatic Virtual Environment (CAVE) is a fully closed environment where the walls are the screen with projectors behind them. The first CAVE was built at the University of Illinois in 1992.[CNSD+92] The user uses 3D glasses inside the CAVE in order to see the generated 3D graphics. With this technology in the CAVE, the view is perceived as 3D. It is possible to have infrared cameras or electromagnetic sensors which track the user who has appropriate items that allow the detection. In such schemes the view is oriented, translated or simply adapted according to user's movements.

The main goal of the CAVE is to give the user the full experience of the simulation by taking the user inside the simulation environment. However, the challenge is still ongoing.

The main immersion problems are the followings [CNSD+92]:

- **Field of View**: Field of view is the visual angle of the viewer without head rotation. At CAVE this is not the biggest problem because the viewer can see by turning their body.. The issue may happen because of the the 3D glasses as there are always blind spots that leak the 2D stereo view.

- **Panorama**: In order to convey a real experience to a CAVE user, the user should be able to look around without immediately finding the limits of visualization. The solution in the design of a CAVE is to enlarge the surface of the projected scenery and to enclose the user as far as possible.

- **Viewer-Centered Perspective**: This is giving the proper screenshots as fast as possible according to the viewers' position. It is highly dependent on hardware. Viewer-centered perspective may be hard to catch in CAVE due to the delays sourcing from the position trackers and the difficulty of synchronization of rendering nodes.

- **Body and Physical Representations**: As the body is there in the system, the representation of the body is also there, therefore this is not a problem for CAVE.

- **Intrusion**: Intrusion is the blockage of the tool, not letting the person get into the environment. This is not a big problem in the CAVE because the person can move around as if it is real.

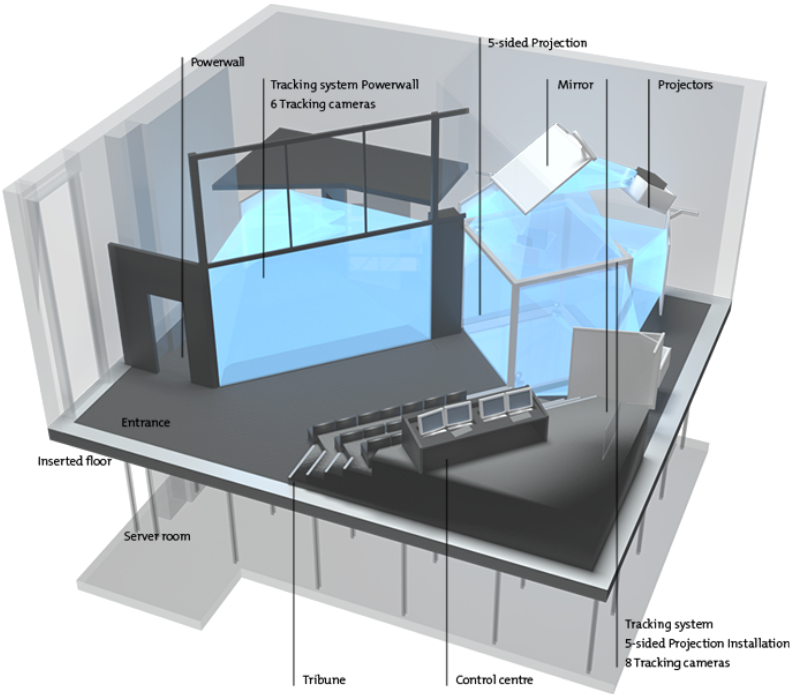


Figure 2.5: Schematic view of the CAVE-room from LRZ. Powerwall at the left side and CAVE-Like installation at the right side.

As well as immersion there are also visualization issues which are:

- **Visual Acuity**: Visual Acuity is defined as the proportion of the number of the pixel taken from the center of the view in 1 minute. According to Snellen fraction, which is a numerical representation of visual acuity [oOS09], the visual acuity can be determined with the function 20/X where X is the number of feet in eyesight. While the average is 20/20, the legally blind is accepted as 20/200. According to the [CN92] the CAVE has the Snellen fraction for 20/110 which is quite poor.

- **Linearity**: Linearity is the linear adeptness of the field of view and resolution of a frame when it is displayed in larger hardware. This might cause some lines to alter to curves. However, there is no problem with it at the CAVE.

- **Look Around**: Look around is having the same object from different views without any anomaly. CAVE successfully fulfills this property.

- **Progressive Refinement**: Progressive refinement is to refine the rendering dynamically as the viewer stops interacting to view. The rendering output first coarsely represented. As long as the view does not need to change, it is refined to a finer model. In CAVE, since it is larger than the viewpoint of humans, there is always some parts of the view to refine as well as some parts to keep coarse.

- **Collaboration** : Collaboration is letting more than one user into the environment. For CAVE, it is easy and possible to get into the environment with more than one person, however, providing separate views for each individual is complicated.

2.4.2 Auxiliary Hardware

In order to achieve the full 3D experience, unfortunately, a X-sided CAVE environment is not enough. For full virtual reality experience there are other auxiliary hardware. The most important ones are shuttered glasses.

An active shutter 3D system is a method to display stereoscopic 3D images. Each shutter glass contains liquid crystals which becomes opaque when a voltage is applied, and transparent when no voltage is applied. The procedure is as follows: While displaying the corresponding image for the left eye, it blocks the right one and vice versa. This process creates depth perception, which the brain reads as 3D.

In order to locate the position, there are infrared cameras or electromagnetic sensors. These cameras follow the reflection or signal coming from any kind of tracker. The trackers may vary, such as head trackers, flystick (which can be seen in Figure 2.6), and finger trackers.

2.4.3 CAVE-like installation at The Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaf-ten (LRZ)

The Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ) is a public IT service provider which was founded in 1962.[LRZc] LRZ is located in Garching and provides numerous research opportunities for organizations throughout Bavaria as well as working actively at the national and international levels.

[3]https://ar-tracking.com

Figure 2.6: Flystick from Advanced Realtime Tracking[3]

The Virtual Reality and Visualisation Centre(V2C) of LRZ is a department which researches visualization and virtual reality.[LRZa] The V2C has a 5-sided CAVE-like installation that offers researchers to experience their research uniquely. The V2C CAVE has 5 sides with a 2,7 m x 2,7 m screen area. Each screen has two projections merging. The installation is driven by 12 nodes, where 10 of them are projecting to the cave while 2 nodes, so-called CAVE-master nodes, there to control the other nodes as well as update the screen. There are 4 cameras to track the user from above and 4 from below. [LRZb] The technical details are as following :

- Cluster with 12 nodes
    - · each 2 x Intel 8core Xeon
    - · each 256 GB
    - · each Nvidia Quadro P6000
- 5 x 2,70 m x 2,70 m screen area
- 10 x CHRISTIE DLP-Projector 3-chip Simulation Mirage WU-3
    - · 1920 x 1200 Resolution
    - · 3.000 ANSI Lumen brightness per projector
- ART TrackPack4 + optional 4 cameras

Chapter 3

Implementation

In this chapter, the information about implementation in detail will be provided. The chapter is composed of three sections: Improving the main script in a way to fit the computational steering approach, computational steering implementation to mpFluid, visualization pipeline and CAVE oriented tuning.

3.1 Improving the main script of mpFluid

For each simulation in mpFluid, a so-called main script for the simulation is prepared. The main script contains the mesh parameters, geometries, boundary conditions, simulation parameters that are necessary for the simulation. Furthermore, the main simulation loop is also found in the main script.

In the beginning, the geometries, boundary conditions, parameters and many other important and necessary entities for the simulations were either hardcoded or taken from the command line interface in mpFluid. Since within the scope of this work, some of the parameters are to be altered from the visual front end that will be implemented, they are needed to be implemented more flexibly.

To do so, three new structures are implemented. The first one is for geometry, the second one is for the boundary conditions, and the third one is for general configurations.

The geometry and boundary condition are implemented in a way to contain the necessary information such as bounding box, temperature, translation and rotation, and if necessary, the type (e.g. inlet).On the other hand, the information related to domain boundaries and meshing parameters belongs to the general configurations.

The geometry processing part of the main script was divided into two parts. The first part is to load the geometries and the second one is to merge them. The master computing process

loads all the geometries, and merges all geometries to create one single geometry. Finally, the geometry is ready for the meshing process.

The domain discretization part is separated for an existing geometry case, and a non-existing geometry case. If a geometry exists, the cells which contain geometry, are assigned to be solid.

The boundary condition implementation is changed to a computing process function. The boundary conditions are already known by the computing process, therefore, it is no longer necessary to have it in the main script. This is also valid for the wall boundary conditions. In the function to apply boundary conditions, first, temperature boundary condition is applied to the solid cells which belong to a geometry. Then converting fluid cells to desired boundary condition cells. This process is followed by setting the temperature of the part of solids if necessary. The final application is to set the initial conditions of the fluids. The pseudo-code for the function can be seen on Algorithm 1.

Algorithm 1 Applying Boundary Conditions Algorithm

```
1: for each Gridindomain do
2:     for i ← NumberOfCellsInX do
3:         for j ← NumberOfCellsInY do
4:             for k ← NumberOfCellsInZ do
5:                 if cell(i,j,k) is SOLID then // Set temperatures on geometries
6:                     Find the geometry and apply the temperature
7:                 end if
8:                 for each BCinBoundaryConditions do
9:                     if Not apply BC to Solid Cells and cell(i,j,k) in BC then
10:                        (cell(i,j,k)->type) ← (BC->type)
11:                        (cell(i,j,k)->velocity) ← (BC->velocity)
12:                        (cell(i,j,k)->temperature) ← (BC->temperature)
13:                    end if
14:                end for
15:                if cell(i,j,k) is SOLID then // Set temperatures on solids
16:                    for each BCinBoundaryConditions do
17:                        if Apply BC to Solid Cells and cell(i,j,k) in BC then
18:                            (cell(i,j,k)->velocity) ← (BC->velocity)
19:                        end if
20:                    end for
21:                end if
22:            end for
23:        end for
24:    end for
25: end for
```

Applying the wall boundary conditions has also become the part of domain function as it can be altered after restarting the application. There are always six boundary conditions at

walls (East - West - North - South - Top - Bottom). The wall boundary conditions are set exactly in the same manner of normal boundary conditions, however, the difference is that the cells are not domain cells but ghost cells.

3.2 Computational Steering in mpFluid

The main application of this work is to provide data communication through the application and to keep the application running while the parameters are changed. To do so, the initial channel is not changed but extended. Both steering types (global steering and application-level steering) are implemented. In this section, the endpoint (visualization and steering control) is assumed to be the point that provides the necessary signal. In the following sections, those parts will be explained.

The channel for communication is the currently implemented channel. At each time step, the computing processes checks if there exist any order from the neighbourhood server. Therefore, the implementation of a new steering function is done there except for the precomputation functions such as starting a signal for the simulation and waiting for the initial configuration.

3.2.1 Global steering

As stated before, both global and application steering is implemented. In this section, the global steering commands will be considered.

The global steering commands give the user more control without modifying the result of the simulation.

The global steering functions implemented to the application are "Connect", "Start Simulation", "Get Initial Configuration", "Pause", "Continue", "Restart", "Checkpoint", and "Kill". These commands can be divided into two, before meshing and after meshing. Before meshing commands are "Start Simulation" and "Get Initial Configuration". These commands are just executed by the master computing node. After meshing parameters are "Pause", "Continue", "Restart", "Checkpoint", and "Kill".

"Connect" command is defined for connecting to the SWCP with a TCP protocol. The massive parallel fluid code can run as a steering and a non-steering type application. While running a non-steering type, it does not require a signal to start the simulation. The user may still desire to use the sliding window for outputting in-situ data. To do so, using the connect signal, the user could connect to the application and could require data.

In the sense of steering, after starting the application, the user may disconnect from the application as the simulation runs, and would like to come back later. For this situation, the

user does not need to stay connected to the simulation, may disconnect, and still connect back. Therefore, they can use the "Connect" command in runtime, after meshing. Below the global steering commands are explained.

Start simulation

This is the first point of the master computing node directly waiting for a command to start the simulation.

When the start signal is received by SWCP, it sends a signal to the neighborhood server to change the flag of "Start simulation" true. At the same time, the master computing process is already in a loop of checking this flag. When this flag is true, it breaks the loop and the simulation continues to the next step which is "Get Initial Configuration"

Get Initial Configuration

At this point, the main computing node expects to receive the initial geometries, boundary conditions, wall boundary conditions, and the general configurations.

The signal of initial configurations starts again from SWCP. The conversion of a data structure to a one dimensional stream of bits is called serialization, and the opposite of this process, the conversion of one-dimensional stream of bits to a data structure, called deserialization [TGMB12].

In this routine, the neighborhood works as a data exchange point. When the data reaches the neighborhood server, it is stored for a small moment, and the flag is changed to signal to the master computing process. Then the master computing process acquires the data and cleans it up from the neighborhood server. Finally, the data is deserialized at the computing process and starts computing.

Five MPI messages that contain different data come during this session. The first message contains the signal to break the loop of waiting for the signal. The second message contains the data for geometries. The geometry string is turned into geometry objects. Then it is stored for the point of meshing. The third message contains boundary conditions. Like geometry string, it is also converted to its object which is this time the boundary conditions objects.

The boundary conditions are followed up by the wall boundary conditions. The wall boundary conditions are also converted to the same boundary condition class objects as the boundary conditions, but the container is a separate one since the usage of it is different.

And the last string contains the general configurations. It is the one that has all the simulation parameters, the domain limits, the mesh parameters are some of the information in the general configuration object.

When this command is received, the simulation continues setting parameters, meshing, distributing the cells to the worker processes, assigning boundary conditions, assigning initial conditions and broadcasting the parameters to all worker processes. After this point, all the global steering commands are followed by all of the computing processes.

Pause and Continue

The pause and continue signals are two complementary signals that pause and continue the simulation, respectively.

When the signal of pause simulation reaches to all of the computing processes, the processes are stuck into an infinite loop. In that loop, all of the computing processes expect a signal to escape the loop.

The continue signal is the signal letting these computing processes escape out of that loop. Then the simulations continue from the point it is paused.

Restart

The restart command also follows the same way until computing processes. To each process, a flag of restart is added with a false value. The restart signal simply turns this flag true.

When the restart flag is true, first all the grids are cleaned up. Then all the grids registered to the neighborhood server are deleted. Finally, the loops for time stepping and computing are broken. Consequently, the restarted simulation reaches to the point where the code expects initial configuration.

For the last step, it turns the restart flag back again to false. Therefore, if the simulation time exceeds the maximum time desired, the simulation ends without a problem.

Checkpoint

The "Checkpoint" was previously implemented signal, which dumps out checkpoint files to restart from that point and VTK files to visualize the checkpoint.

Kill

The kill command is implemented to kill the simulation. As the simulation is an MPI based implementation, in order to kill the simulation, reaching to SWCP is more than enough. In SWCP, an MPI Abort call is implemented to kill the simulation in no time.

3.2.2 Application level steering

Similar to the global steering commands, the application level steering commands are implemented. The aim of these commands is rather than globally controlling the situation of the simulation, making changes to the solution and the domain. The commands implemented are :

- Add geometry
- Delete geometry
- Add boundary condition
- Delete boundary condition

The modification of any geometries or the boundary conditions is implemented as pairs of both add and delete commands. First, the entity is cleaned from the simulation. Then the modified entity is added.

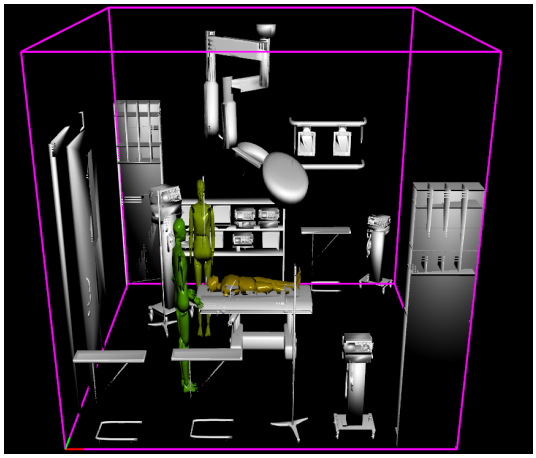
Add Geometry and Delete Geometry

The adding geometry and the deleting geometry signals reach to sliding window server and they are followed up by a serialized string. The serialized string is nothing but a serialized geometry object. The signal and string travel through MPI channels until the computing processes. When the signals reach the computing server, the string is converted back to a geometry object, which contains the necessary information such as geometry file name, translations, rotations and more, with the help of the BOOST library[1].

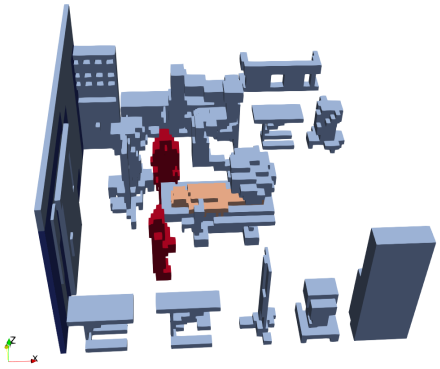
The second step after receiving the geometry object is to create a so-called dummy domain. This dummy domain is the same as the current computing domain.

In this domain, the geometry that is being added is meshed as the third step. First, the geometry is loaded to the dummy server, then it is translated, rotated, scaled according to the geometry object. The meshing parameters are the same ones for the currently running main simulation. Here the consistency of the meshing functions provides the advantage that the current simulation's and the dummy domain's mesh grid ids are identical, therefore, the only "solid" labeled cells are the cells containing the facets of the newly added or deleted geometry. In figure 3.1 and figure 3.2 one can see the resultant scenes for both mpFluid (in which geometry is represented by the solid cells of the domain) and visual front end (where the triangular representation is shown) for both geometry deletion and addition, respectively.

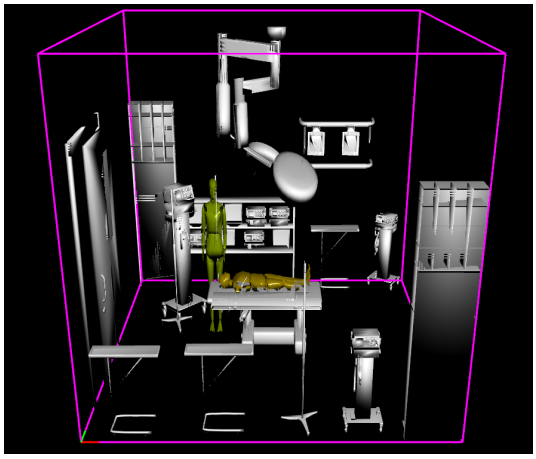
[1] https://www.boost.org/



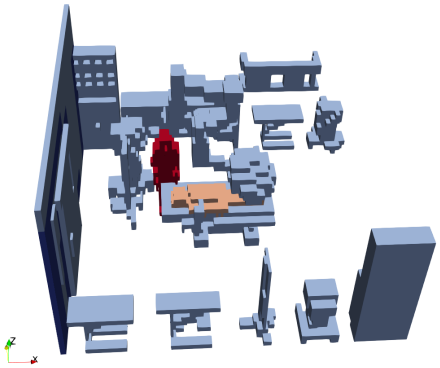
(a) View in front end.



(b) mpFluid's view in Paraview



(c) View in front end.



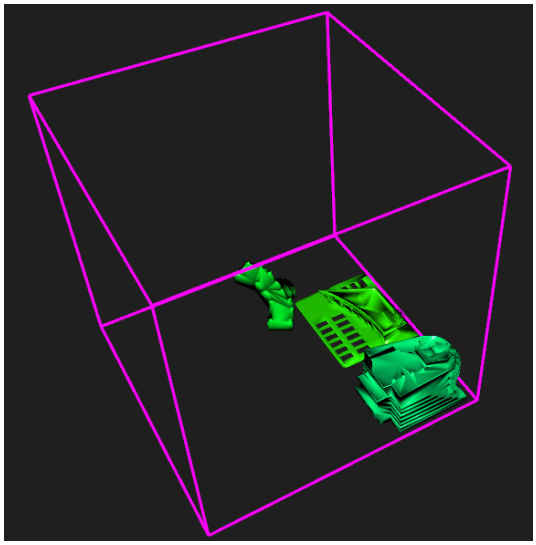
(d) mpFluid's view in Paraview

Figure 3.1: Deleting Geometry. The left the views are from the front end, while the right views are from Paraview, as an output of mpFluid. One can see initial and final situation after deleting a geometry.

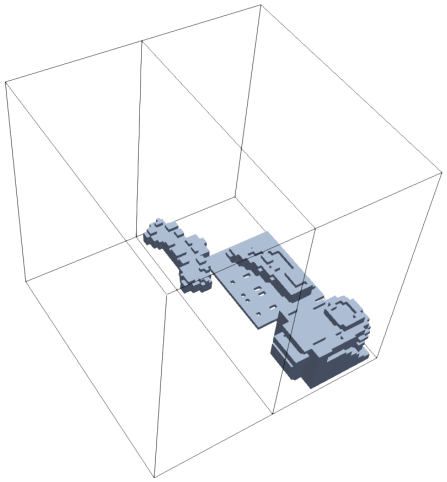
Add Boundary Condition and Delete Boundary Condition

Adding boundary conditions, and deleting the boundary conditions are other complementary steering functions that are implemented. While the workflow is similar to how it is in adding or deleting geometries, the boundary condition object that are, again, serialized by the BOOST and deserialized when it reaches to a computing process.

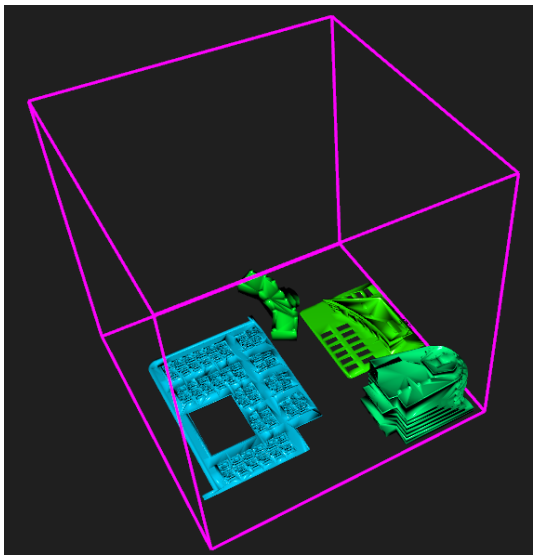
Adding or deleting a boundary condition is changing the boundary condition type of the domain of which the boundary condition object covers. All the computing processes loop over their cells, compare the cell bounding box to the boundary condition's boundary box. If the cell bounding box is inside the boundary conditions boundary box, the cell type is altered to the boundary condition type according to the boundary condition object's boundary type. Then the value is altered.



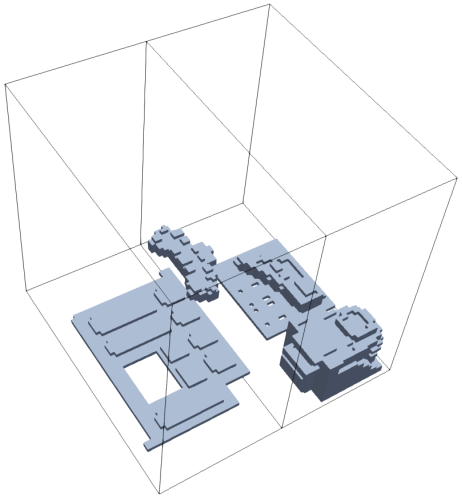
(a) View in front end.



(b) mpFluid's view in Paraview



(c) View in front end.



(d) mpFluid's view in Paraview

Figure 3.2: Adding Geometry. Left the views are from the front end, while the right views are from Paraview, as an output of mpFluid. One can see initial and final situation after adding a geometry.

For example, if we are adding a solid boundary condition with a 30 degrees temperature, first the cells are found that are inside the boundary condition object's bounding box, then the temperature is changed to 30 degrees.

3.3 Visualization Pipeline

A large part of this thesis deals with giving raw data a visual meaning, a visualization, which users can perceive easily. Therefore it is quite important to know all of the visualization pipeline.

Visualization is done in order to give insight into a process by creating different types of images. The visualization process consists of numerous steps that are followed to create these images. The step-following procedure is a reflection of the divide and conquer method in the visualization field. This data transformation process from raw data to a rendered image is called the visualization pipeline which is illustrated in Figure 3.3. The pipeline has basic stages which are data acquisition, filtering/enhancement, visualization mapping and finally, rendering.

Data Acquisition

Data acquisition can be defined as the stage of creating/receiving data from the data sources. It is possible to collect the data sources under three categories which are simulation, databases and sensors. Some exemplary data sources are global climate simulation, weather radar data, sensors such as thermocouples, and spatial databases. The data acquired after the data acquisition stage can be called raw data.

In this work, the data is generated in massive parallel fluid code. The generated data consists of cell data including point spatial data x, y, z; velocity data for each direction and magnitude, v_x, v_y, v_z and $||V||$; temperature data, T; and data regarding to massive parallel fluid code's data structure such as *cell type*, *depth*, *grid id* and, *process rank*. The raw data is conveyed to the first stage of the pipeline through a TCP connection in the data format of the Visualization Toolkit (VTK)'s unstructured grid. From this point on, the steps will be explained with the generation of streamlines through this pipeline.

Filtering and Enhancement

The Filtering and enhancement stage can consist of many different stages. This is where the raw data is converted into derived data. The user should decide the important aspect or features in the raw data. To do this, many operations can be applied. Some of which operations are data format conversion, resampling to the grid, interpolation/approximation of mission values, data reduction or clipping.

In this thesis, the data reaches the visualization pipeline as an unstructured grid of VTK. The VTK data consist of all velocities at the cell centers. Each cell contains 8 points. The velocities are interpolated to these points.

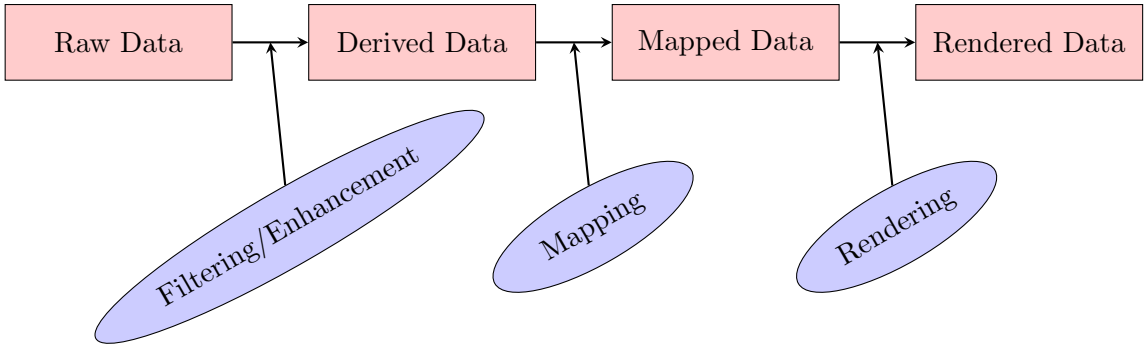


Figure 3.3: Visualization pipeline.

Mapping Data

The resulting data from the previous stage should be the data that was expected by the user for exploration of interest. When the derived data acquisition is accomplished, the data should be mapped to the visual domain. This step is called mapping. The new domain that the data is mapped to contains perceivable elements such as shape, position, size, color, texture, illumination, and motion. This can be thought of as the most crucial step affecting the final view.

In the streamlines of the current work, first, the data passed through a lookup table to generate the colors, which are in visual channels of this stage, is according to their corresponding velocity magnitudes. After the coloring is finished, the geometrical shapes, which are lines for this case, are generated.

Rendering

The last stage of the visualization pipeline is the rendering stage. This stage is getting the mapped data and combining it with some user-specified viewing parameters such as the viewpoint and rendering. Through rendering parameter alteration, the user can navigate through the output, explore it freely. In this navigation and user interaction, such parameters like viewpoints or light position may change, the mapped data from the previous stage stays the same.

3.4 From Raw Data to Visualization in CAVE

The current work is an extension of mpFluid and is written in C++ using a third-party library called Magnum Graphics[2]. Magnum Graphics is a *Lightweight and modular C++11/C++14 graphics middleware for games and data visualization.* according to the definition of Vladimír Vondruš, who is the main developer of the open-source code. Magnum engine is a high-level code, utilizing the functions of the Open Graphics Library(OpenGL). OpenGL is a 2D and

[2]https://magnum.graphics/

3D application programming interface (API) which is developed by Khronos Group and extensively used in many fields. The advantage of Magnum Graphics was that it has full functionality of OpenGl API as well as extended functionalities such as scene graph, and built-in shaders.

The implementation is designed as a client-server model. The client-server model is a type of distributed application. The server is the service provider, and the clients are the service requesters. At the visualization center of the LRZ, the server is called "Cave master" and the clients are the 10 rendering nodes in the CAVE which are illustrated in 3.4. The main duty of the server is to do the synchronization between all 10 nodes.

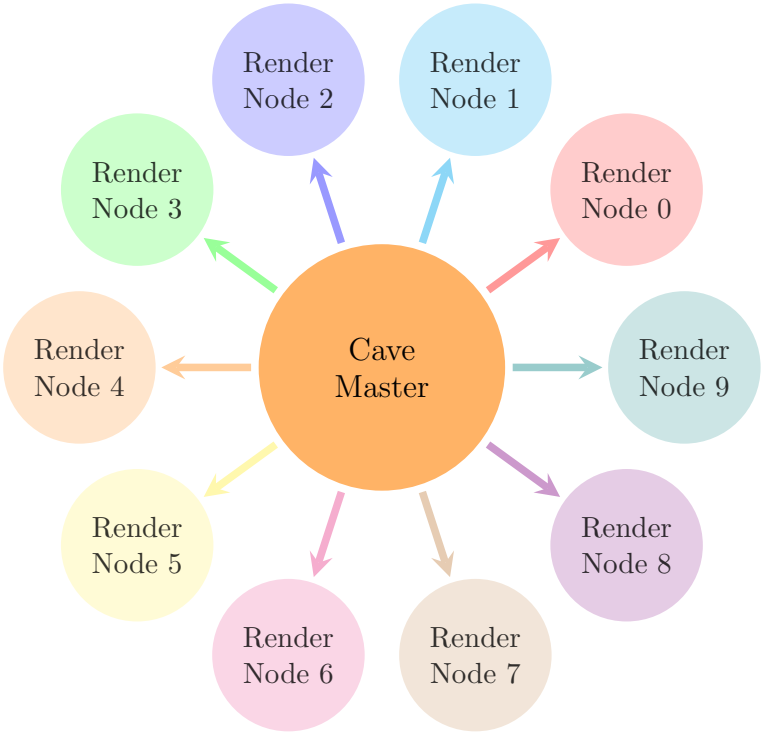


Figure 3.4: Client-Server model in the LRZ

The communication between server and client is done one-way, from server to client using the LRZ's still under development library. The library is built on Boost Asio which provides TCP and User Datagram Protocol (UDP). All communication in the working visualization code is provided through UDP channels.

In order to create a visualization tool, one needs to have low-level access to audio, keyboard, mouse, joystick, and graphics hardware. In this application, GLFW[3] is used to create the windows and context and receiving inputs and events.

[3]An open source API for creating windows, contexts and surfaces, receiving input and events. https://www.glfw.org/

As explained, the code is made of two separate parts, client and server. The server is called Controller which runs on the computer called Cave Master, and the render node's code is called Node. In the following, these parts will be explained.

3.4.1 Controller

The Controller is the program taking care of all synchronizations of the 10 render nodes, and the connection between mpFluid and the CAVE which is schematically presented in Figure 3.5. The signals and the data needed for mpFluid for computational steering is generated at the Controller.

The Controller has two parts, the front part, in which the communication between the render nodes as well as the user happens. The second part, which is the back end part is where the signal and data are prepared to send to mpFluid.

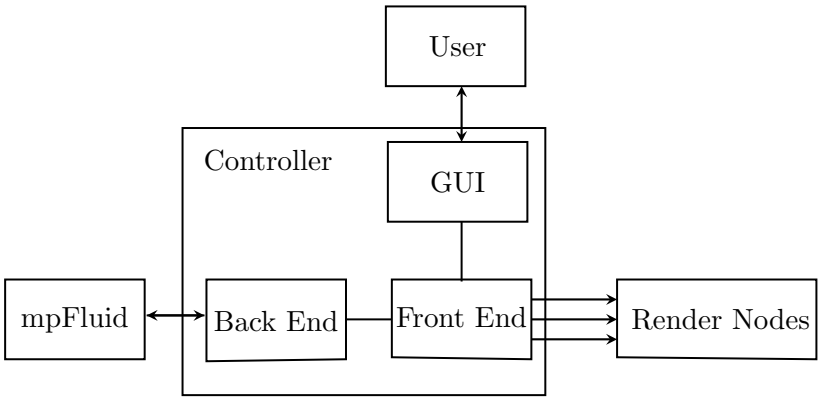


Figure 3.5: The Controller diagram

To have full control, a Graphical User Interface(GUI) is developed and connected to the frontend. The GUI can be seen in Figure 3.6 The GUI is developed using DearImGui[4]. The GUI is made of 10 subparts: Basic Signals, General Steering, General configurations, Boundary conditions, Geometries, Wall Boundary Conditions Sliding Window, Camera View, Slice Field, and Save/Load Configurations.

- **Basic Signals**: These contain three signals. The first one is to signal to connect to mpFluid. The second and third one are to request data from mpFluid for either velocity of temperature.
- **General Steering**: These signals are the signals explained in the previous section, to steer the application globally.
- **General configurations**: In general configurations, the domain boundaries, and meshing parameters can be set.

[4]https://discourse.dearimgui.org/

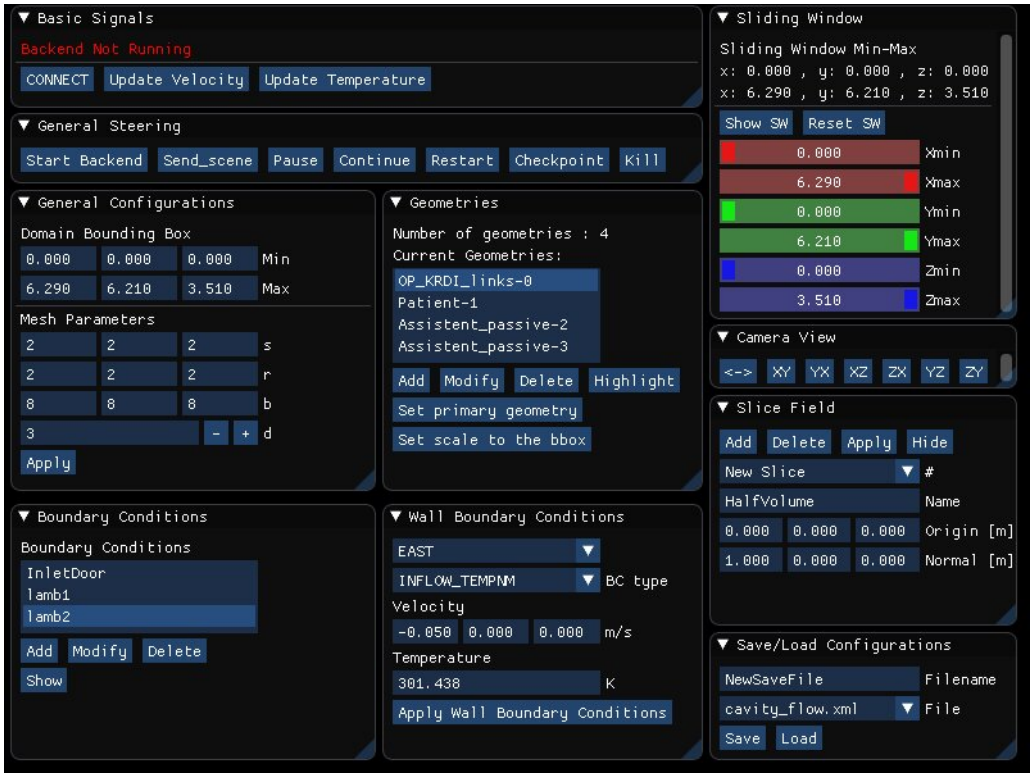


Figure 3.6: GUI view of Cave Master.

- **Geometries**: In this part, the user can add, delete, or modify the geometry.
- **Boundary conditions**: In this section, one can add, delete, or modify the boundary conditions
- **Wall Boundary Conditions**: In the Wall Boundary Conditions, the user can modify the wall boundaries.
- **Sliding Window**: Here the user can specify the region of interest to request a smaller domain from mpFluid.
- **Camera View**: Some geometries and configurations may come with different orientations. This is to change the view of the CAVE.
- **Slice Field**: The user may generate a slice of the currently available field using this part.
- **Save/Load Configurations**: The user can save or load the desired configuration.

The front part the code is responsible for making sure the code receives the orders from the GUI and sends them to the render nodes. And if necessary, it sends the orders to the backend, where it is sent to the mpFluid. For example, the update of the velocity field is as follows:

1. The user clicks update velocity after setting the sliding window limits.
2. The GUI backend updates the program frontend, and the necessary function is called.
3. The function first sets the sliding window limits of the backend. Then, it requests data.
4. When the data is prepared, it signals the render nodes that the data is ready to be picked.
5. Render nodes generate the streamlines.

The backend part creates the data transmission and connection between the visualization and mpFluid. It registers the backend geometries and boundary conditions, thus, if there is an order to send the scene back, it is the one comparing, if the entity is already at mpFluid, if it is needed to be added, deleted or modified (first deleted, then added). It is also a responsibility of the backend to provide the data needed to render nodes to render the data. All serializations of the data using Boost is happening also at this part of the code.

3.4.2 Node

Node is the program that runs in the rendering nodes. It needs the Controller program to start properly. Its duty is to visualize the data coming from the Controller as well as providing the visualization for the CAVE environment. The explanation of the Node program follows below.

Scenegraph

A visual application may have many different objects to show at the same time, and not all of these objects are in the camera eyesight, therefore, the objects actually do not have to be drawn. A smart drawing may save a lot of performance. The scenegraph is a type of data structure that helps to organize the objects according to their front-to-back order for z-buffering (a method to decide which object will be rendered to the scene first. [Str74] or for material order for color rendering. A scene graph data structure is generally nothing but a tree structure such as octrees or kd-trees. In the current scene graph of the application, there exist visualization objects, the camera, and the light objects at the nodes.

Geometry Objects

The visualization objects are created either through raw data coming from mpFluid or importing the data from an outer file. The imported files are the solid geometries used in the simulation. The efficient algorithms group of the Chair of Computation in Engineering department uses Efficient Algorithms Fileformat (EAF) which is a simplified version of Standard Triangulation/Tesselation Language (STL). A parser is implemented to read such a file

type. Using some third-party libraries, it can be extended for other file formats such as .obj or .stl, however, for the scope of this work, it is not necessary.

At the last step of the parsing of .eaf file, a further step is applied to calculate vertex normals. Vertex normals are necessary for the proper lighting of the geometry. The calculation of the vertex normals can be seen in the algorithm 2

Algorithm 2 Weighted Vertex Normal Generation Algorithm

Require: Array of Facets of the imported geometry.

1: Initialize storage array, *Vertex Normals*, for all vertices.
2: **for each** facet in Facets **do**
3: Calculate normalized facet normal using 3 vertices.
4: Calculate normalized vertex-to-vertex vectors.
5: Calculate angles between vertex-to-vertex vector.
6: Calculate weighted normal of the vertex according to current facet and add it to vertex normal in *Vertex Normals*.
7: **end for**
8: **for each** normal in *Vertex Normals* **do**
9: Normalize normal.
10: **end for**

The imported data is stored as mesh and submitted to graphics hardware. In order to visualize this mesh, the vertices are transformed, lit and the projected to the clip space. A shader is a type of code used for shading the objects. Many shaders are written in OpenGL Shading Language (GLSL)[5] which is a C-type language. After the vertices are submitted to the hardware, the vertex shader does the transformation of the vertices according to the user-specified view as well as the fragment shader does the painting of the geometry. The imported geometries in this work are drawn by Phong shader. Phong shader is an in-built shader of Magnum graphics, which uses the Phong illumination model [Pho75].

Phong illumination model is the most common illumination model employed. It is based on the Phong illumination model. Phong illumination model is the sum of the three terms: Ambient light, diffuse reflection, and specular reflector.The three terms and the final view can be seen in Figure 3.7

Ambient light is a fixed intensity, fixed color light that all objects at the scene are affected equally. The surface variations on space do not affect the view. The ambient light is calculated through the equation :

$$C = k_a * C_a * O_d$$

where C_a is the color of ambient light, O_d is the object color, and k_a is the ambient reflection coefficient. The value of k_a vary from 0 to 1.

[5] https://www.khronos.org/opengl/wiki/Core$_{L}anguage_{(}GLSL$)

Diffuse reflection is the reflection of the light from the surface in different angles depending on the surface normal, light source normal, the viewers normal. The equation related to diffuse reflection is :

$$C = k_d * C_p * O_d * cos\theta$$

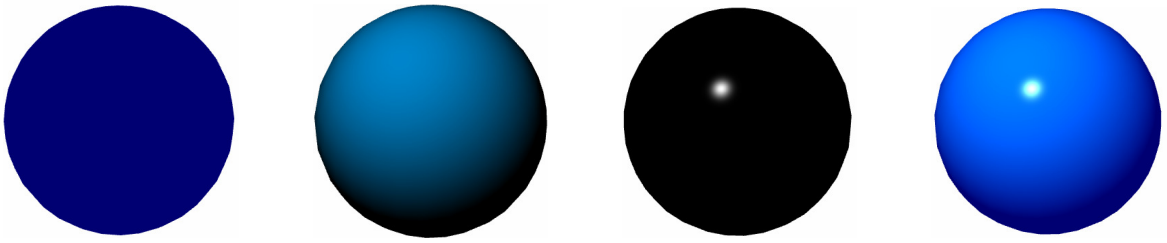
where C_p is the color of the diffuse light, O_d is the object color, and k_d is the diffuse reflection coefficient. The value of k_d varies from 0 to 1. $cos\theta$ in this equation is the angle between the light vector and surface normal. It is possible to get shaded areas in diffuse reflection when the angle between surface normal and light source is more than $\pi/2$.

Specular reflector is creating a reflection of the light source and highly dependent on the viewer's point.

$$C = k_s * C_s * O_d * cos^n\theta$$

where C_s is the color of the specular light, O_d is the object color, and k_s is the specular reflection coefficient. $cos\theta$ is the reflected ray while n is the shininess factor.

The Phong lighting is finally the sum of all three color channels. The separate 3 channels and the final Phong illuminated model can be seen in the picture below.



(a) Ambient lighting **(b)** Diffuse lighting **(c)** Specular lighting **(d)** Phong model

Figure 3.7: Phong Illumination Model

The Phong illumination algorithm is in the fragment shader of the in-built Phong shader. The vertex shader, on the other hand, deals with translating the part according to the viewpoint of the user. For each call of drawing the object the parameters of the shaders such as transformation matrix and projection matrix are set.

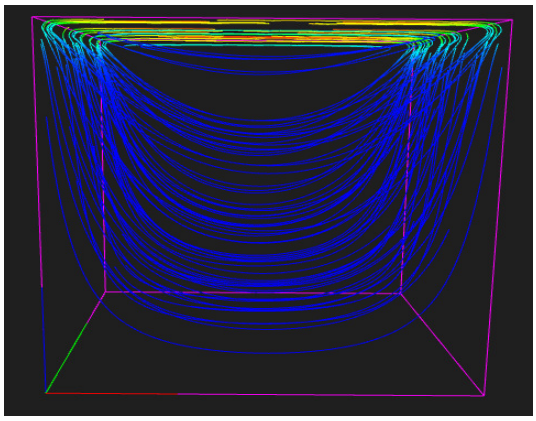
Flow Data Visualization

The data visualization is implemented for two different types of data, velocity, and temperature respectively. The data signal is sent from Controller. The data can be processed in three ways: Streamlines, temperature point cloud, and slices. All data processing (Data filtering and mapping) is done through VTK functions. The data, is then, extracted and sent to generate the geometry primitives such as lines and triangles.

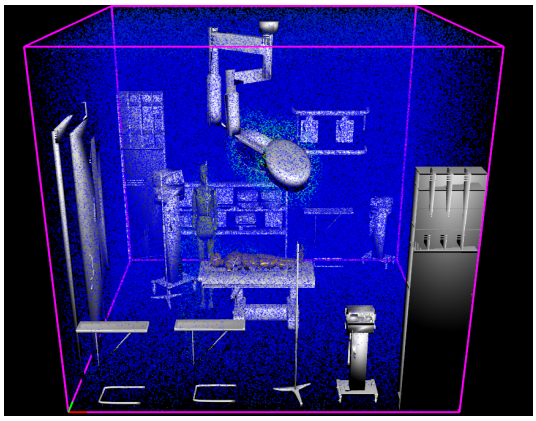
Streamlines, as explained in the data visualization section, are generated by using line strips. The generated colors from look-up tables are sent applied using the vertex points. The color between the two vertices is interpolated according to the color variables of the vertices.

In order to give ideas of the scalar attributes in the CAVE such as temperature, semi-transparent point cloud is used which is created through VTK. The point cloud is created to fit the sliding window. It is interpolated through the domain in order to have the interpolated correct color. It is then rendered. The resultant figure can be seen in figure 3.8b. The view in the CAVE is as flying

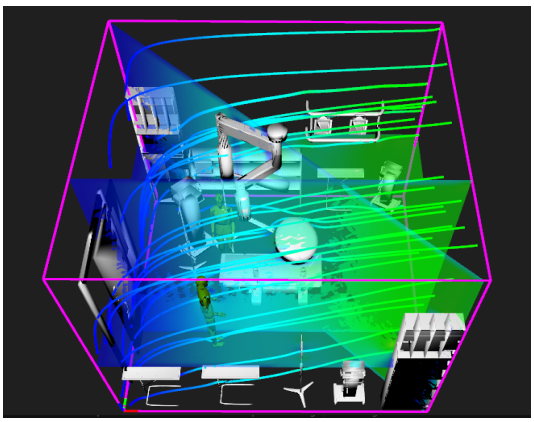
Slices are the representative slice of the domain with scalar value based color. A slice contains vertex colored triangles which represent the scalar value that are asked for. The origin and the normal of the slice are sent from the Controller according to the wish of the user. The code slices the domain according to the previous update from the mpFluid. If it contains velocity data, then a velocity slice is generated. If it contains temperature data, then the slice is a temperature slice.



(a) Streamlines of lid driven flow in a cavity.



(b) Temperature interpolated point cloud



(c) Slices in flowfield loaded scene.

Figure 3.8: Visualization variations of render node.

Rendering the scene in the CAVE

Rendering the scene is calling the "draw" functions of the scene graph objects if the objects are needed to be drawn. However, when it is to be drawn in the CAVE, some additional considerations are needed. As explained earlier, inside the CAVE there should be two drawings at once, one for the left eye and one for right. To achieve this there are two ways: Having two separate cameras, one for each eye, and transforming the camera between images. For this application, the latter is selected.

It is important to mention here another library in use. The "Virtual-Reality Peripheral Network" (VRPN)[6] provides the communication with the tracking tools. As the main user's glasses in the CAVE are also connected to the head tracker, the data necessary to get the position as well as the rotation of the head is communicated, VRPN library helps crucially.

During the rendering event in the CAVE, two projection matrices are requested from the rendering nodes in which are left and right projection matrices. These matrices are the matrices to transform the camera for the eyes individually. Sequentially, the processes are as follows. In the beginning of each rendering, the scene is cleared. Then the projection matrices are requested. The camera's projection matrix is set first for the left eye. Then the objects are drawn. The scene is cleaned. This time everything is redone for the right eye. Finally, the nodes are synchronized with the master. Then the drawing event is repeated. The algorithm can be seen in algorithm 3. This functionality is provided with OpenGL's quad-buffer flag which also provides two extra buffer for back-left and back-right for both eyes. The resulted 10 scenes from 10 render nodes, which can be seen in Figure 3.9, are clipped through third-party libraries and finally projected. The resulted scene is a stereo scene of how it would look in 1 render node.

Algorithm 3 Rendering algorithm of a render node.

```
1:  function RenderScene( )
2:      Clear the frame buffer.
3:      Enable depth test, face culling, and blending.
4:      Get right and left projection matrices.
5:      set the camera's projection matrix for the left eye.
6:      Render the current scene.
7:      Clear the frame buffer.
8:      set the camera's projection matrix for the right eye.
9:      Render the current scene.
10:     swap buffers to synchronize
11:     call RenderScene again.
```

[6] https://vrpn.org

Figure 3.9: The view from all of the rendering nodes. **Upper**: Left wall. **Middle** : First pair is bottom wall, second pair is front wall, third pair are top wall. **Lower**: Right wall.

3.4.3 Scaling the Dimensions for the CAVE and the Sliding Window

There are two types of scaling applied for visualization. The first one is the scaling for the CAVE, and the second one is the scaling according to the sliding window.

The mpFluid is written for length dimensions in meters. All the calculations, I/O processes, and communication are done in meters. On the other hand, the CAVE is working with the dimensions of centimeters. Therefore, the scene was scaled to fit the CAVE. The dimensions of the CAVE is 270 cm x 270 cm x 270 cm, which is unit wise 270 x 270 x 270. At the beginning of the simulation, everything is scaled to fit into the CAVE. This CAVE fitting looks best when the domain is also a cube. When the domain is a slender domain, it looks as if it is stretched. However this feature is left intentionally to make the user explore the whole domain, without the need for translation of the domain into the CAVE.

The second scaling is also applied in order to increase the immersion of the virtual reality. The idea of the sliding window is to extract the necessary data to explore in more detail. To reach that goal, the sliding window visualization is implemented to fit the CAVE whenever

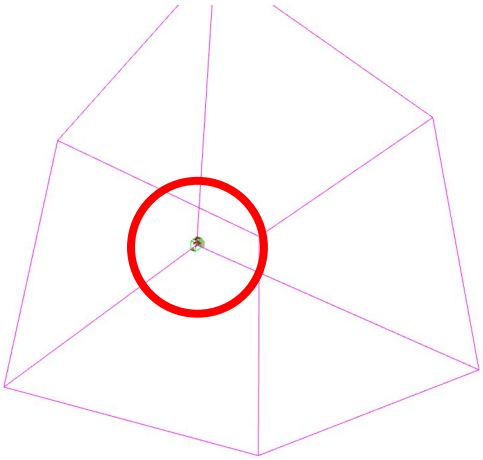
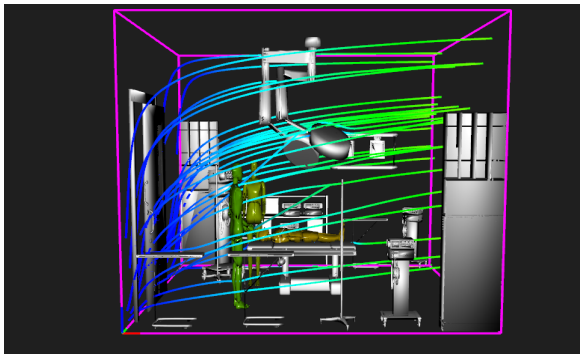
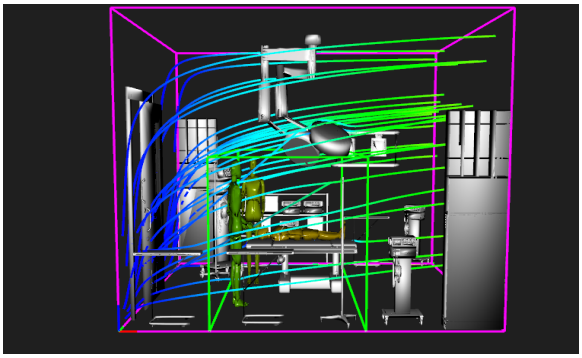


Figure 3.10: The view of the objects and the CAVE without scaling. The CAVE geometry is the purple cube while the real geometry is inside the green prism at the center of the red circle.

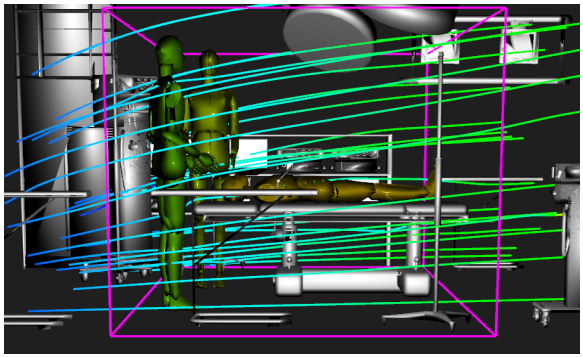
the new visualization data is requested from the massive parallel fluid code. This scaling is a little bit trickier than only scaling the size. It also includes translating.



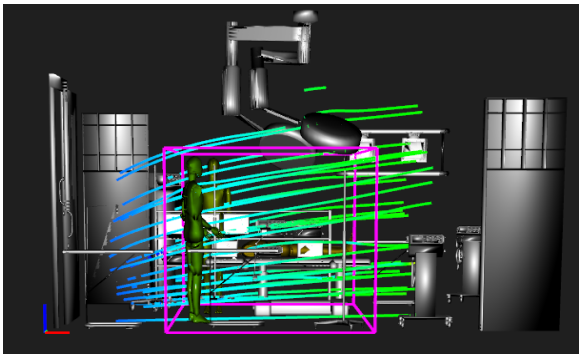
(a) Full scene.



(b) Sliding window in the scene.



(c) The updated scene.



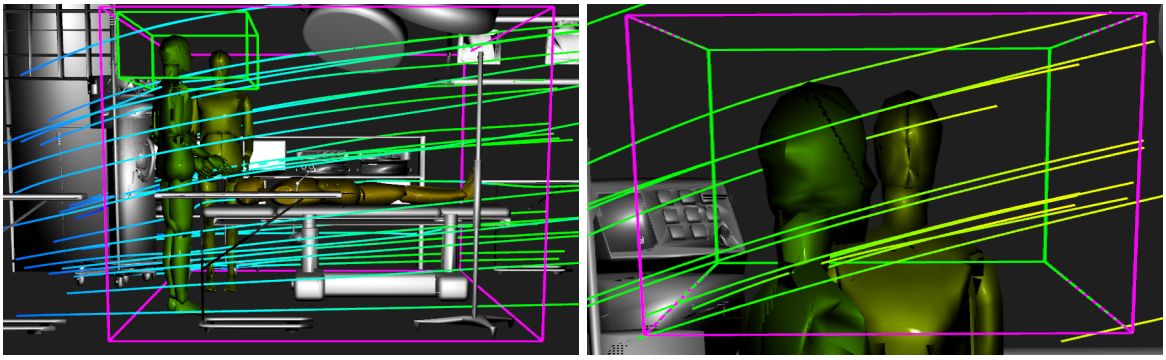
(d) The updated full scene

Figure 3.11: The sliding window application.

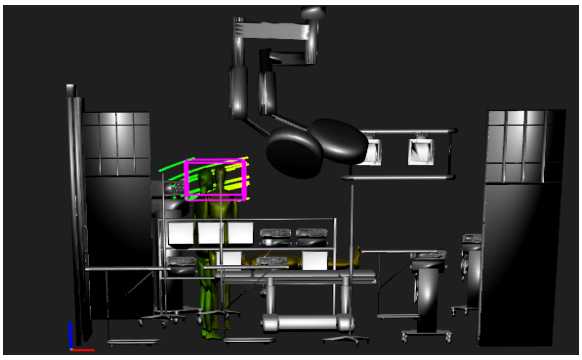
After each data request from the mpFluid, the sliding window is resized to fit the CAVE perfectly. The former sliding window dimensions (before the mpFluid data request is done) and the center of gravity of that is saved. As the new sliding window, which is also the cave geometry is known, the scaling values are updated according to that. Then the position of the old sliding window's center of gravity to the CAVE's is compared. A translation vector from the old position to the new position (CAVE center) is defined and all the objects are translating according to that.

The updated scaling factors, as well as the translation vector, are applied to each drawable group. The final view is provided the goal of the sliding window application in the immerse virtual reality environment.

An example can be seen in figure 3.11. The first figure (a) shows the figure fully fit the CAVE (Purple cube). In the second figure, a domain of interest is selected with a sliding window (Green prism). The third and forth figures show the updated view. The domain of interest is scaled and translated to the CAVE. Also, the streamlines are just around the domain. The comparison between the first and the fourth figures shows clearly the scaling effect.



(a) Second sliding window. **(b)** Secondly updated scene.



(c) Second full scene.

Figure 3.12: Sliding window further application.

In figure 3.12, a second domain exploration is applied on the situation of the figure (d) of 3.11. The sliding window this time is concentrated around the head of the human figure. In figure (b) the streamlines can be seen. As the sliding windows not scaled volume decreases, the number of cells in the domain also decreases, if the domain depth is at the same depth of the previously selected domain. Therefore, the number of streamlines also decreases. The figure (c) of 3.12 shows the CAVE and the fully scaled domain.

Chapter 4

Conclusion and Outlook

Gordon Moore claims that the number of transistors on a microchip doubles every two years.[Moo65] That can be read also this way: The simulations taking hours will take minutes in a few decades. Therefore, the computational steering and visualization will stay as important as it is today in both the academy and the industry. VR and augmented reality(AR) are already in use for marketing, education, communication, hobby, and visualization purposes. The importance of these tools will increase much more as quality, complexity, and performance increases. Today in many sectors in industries such as automotive, and aerospace, the VR and AR technologies are actively used. This thesis provides a view of how a computational steering system can be used in a virtual reality environment to explore the simulation outcomes, as the simulation still runs.

To do so, first, the visualizing code for virtual reality is implemented in server-client fashion. The code is implemented as an extension of the mpFluid code. The visualization is designed to fit the CAVE directly, therefore, the sliding window approach is scaling the view in a way that the user always stays in the sliding window. The VTK library is implemented for processing the raw data until generating the colors, and with Magnum Engine, the data is visualized using geometric primitives. Also, a GUI is created in the Master program of the visualizing code to communicate with mpFluid while that runs. Through the GUI, the user may specify global parameters, create a simulation scenario from the scratch in the limits of mpFluid, apply boundary conditions, and wall boundary conditions, add-modify-delete geometries. After submitting the scenario, the user may again interact with the mpFluid by applying add-modify-delete geometries or boundary conditions, or with any of the global steering commands such as pause or continue. The user may request data to visualize in the boundary sliding window defines. Velocity streamline, scalar point cloud, or a slice is possible.

In mpFluid, the main script is refactored to be more flexible for different inputs and geometries. Third-party libraries are used to support MPI communication with serialization

and deserialization. Global and on-the-fly type computational steering control algorithms are implemented.

The implementation and development in this thesis have created a basis to develop the code further. In the following, some open parts that can be used as a starting point for further development are pointed out.

Feedback the CAVE master from mpFluid

In the current code, everything is assumed to be working perfectly. While the GUI sends some orders back to mpFluid, the only data it receives is the simulation domain's data. One crucial future extension is to extend this feedback mechanism by implementing a listening function to CAVE-master, as well as implementing proper feedback channels to mpFluid.

Improving on-the-fly steering

In mpFluid, when it is necessary to add, modify or delete a geometry, the geometry is remeshed in a dummy domain to generate necessary grid ids in order to locate the cells that need to be altered. A better idea might be to label the cells with some strings or colors, so that, when the order comes from the CAVE master, the alteration can be done faster.

For instance, let us assume a simulation will be run. While processing the geometry, the cells that contain "Geometry A" can have a color attribute of blue. To delete this geometry, now you just need to convert the blue labeled solid cells into fluid cells.

Improving the Main Script

The main script in this work is already improved much to work in a flexible way. However, the scope of the fluid simulations is so wide. The limit of the extensions can be the limit of mpFluid. mpFluid has the limits of solving two-phase flows for example. Therefore the script can be extended in a way that two materials and initial conditions can be conveyed. Furthermore, rather than a building flag, usage of the thermal coupling can also be an option of the main script.

Controlling the GUI from CAVE

The communication library between render nodes and the CAVE master works one way. Meaning that the cave master broadcasts all the data to render nodes. The SYNCH library does not provide a solution for feeding back to the cave master. The main problem here is the GUI is in the cave master, which is out of CAVE. There are two ways of having a GUI in the CAVE.

The first way is to have a third program. The visualization center is able to provide a WIFI connection between cave master and a third party android tablet. Therefore, the GUI may

be moved to a tablet or smartphone. Another variation of this is to create a web socket and deploying the code online, therefore, not even a WIFI connection would be necessary.

The second way is to create a GUI in render nodes, and all the time synchronizing it with cave master's GUI. The CAVE hardware which is trackers and flystick feeds the CAVE master. The location of the flystick and its orientation can be obtained and a line-plane intersection can be applied. The position found can be mapped on GUI of the cave master. If it actually presses a button, the cave master can broadcast the intersection point and the render nodes may update their individual GUIs.

Increasing the VTK usage

The VTK is a very broad library having so many functions. There are so many possibilities for improvements in this code using VTK. VTK provides *vtkExternalOpenGLRenderWindow* class that provides rendering the VTK scene inside an openGL frame buffer. This is a possible investigation point because, at this point, one can create all the scenes with VTK functionalities. such as slicing, clipping, volume rendering, thresholds and much more. In the literature, there are CAVE applications with VTK and Paraview, however, they are always direct visualizations, and not in the sense of what this work is trying to provide. The best would be mixing the Magnum Engine's and OpenGL's capabilities and objects with VTK's capabilities.

Increasing the Number of The Virtual Reality Indicators

Throughout this work, the main idea is to take the user inside the simulation. To do so the visualizations shall be as realistic as possible. Possible improvements that may be applied are the following.

- Adding the walls, floor and ceiling. Especially a floor gives the user the feeling of actually stepping on the location somewhere that belongs to the simulation.
- Shadow rendering. The items inside are illuminated according to their locations with respect to the light in the room. However, no shadow is rendered and that is a missing point of reality.
- Texture mapping. The geometries are one color without any clue of material. For geometries such as human, clothes, and skin may be textured. For chairs, for example, a wood texture can be added.
- Haptic feedbacking. The user in the CAVE can see in 3D. However, the user does not feel anything if he is walking in cold or hot locations of the simulations. A cloth with thermal haptic feedback may increase this feeling enormously.

List of Figures

Bibliography

[ATW+19] Zaib Ali, James Tyacke, Rob Watson, Paul G. Tucker, and Shahrokh Shahpar. Efficient preprocessing of complex geometries for cfd simulations. *International Journal of Computational Fluid Dynamics*, 33(3):98–114, 2019.

[BWFR05] André Borrmann, Petra Wenisch, Jérôme Frisch, and Ernst Rank. Collaborative hvac design using interactive fluid simulations: A geometry-focused collaboration platform. 2005.

[BWvTR06] André Borrmann, Petra Wenisch, Christoph van Treeck, and Ernst Rank. Collaborative computational steering: Principles and application in hvac layout. *Integrated Computer-Aided Engineering*, 13:361–376, 2006.

[CN92] Sandin D. DeFanti T. Kenyon R. Hart J. Cruz-Neira, C. The cave®: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35:65–72, June 1992.

[CNSD+92] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The cave: Audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, June 1992.

[FMR13] Jérôme Frisch, Ralf-Peter Mundani, and Ernst Rank. Parallel multi-grid like solver forthe pressure Poisson equation in fluid flow applications. In *Proceedings of the IADIS International Conference on Applied Computing 2013 : Fort Worth, Texas, USA 23 - 25 October 2013 / Organised by IADIS, International Association for Development of the Information Society ; Edited by Hans Weghorn*, pages 139–146. IADIS International Conference on Applied Computing, Fort Worth (USA), 23 Oct 2013 - 25 Oct 2013, IADIS Press, Oct 2013.

[Fri14] Jérôme Frisch. *Towards Massive Parallel Fluid Flow Simulations in Computational Engineering.* Dissertation, Technische Universität München, München, 2014.

[JB18] A. Janson and C. Butson. Targeting neuronal fiber tracts for deep brain stimulation therapy using interactive, patient-specific models. *Journal of Visualized Experiments*, (138), Aug 2018.

[JP95] Christopher R. Johnson and Steven G. Parker. Applications in computational medicine using scirun: A computational steering programming environment. In *In Supercomputer '95*, pages 2–19. Springer-Verlag, 1995.

[KMR+12] Jovana Knežević, Ralf-Peter Mundani, Ernst Rank, Ayla Khan, and C. Richard Johnson. Extending the scirun problem solving environment to large-scale applications. 2012.

[Kne13] Jovana Knežević. *A high-performance computational steering framework for engineering applications*. Dissertation, Technische Universität München, München, 2013.

[KOU03] Michal KOUTEK. *Scientific Visualizationin Virtual Reality:Interaction Techniques andApplication Development*. Dissertation, Delft University of Technology, Delft, 2003.

[KU07] K. D. Kim and Rizwan Uddin. A web-based nuclear simulator using relap5 and labview. *Nuclear Engineering and Design*, 237(11):1185–1194, 6 2007.

[LM00] P. J. Love and Jeremy M. R. Martin. Steering High-Performance Parallel Programs: a Case Study. In Peter H. Welch and Andrè W. P. Bakkers, editors, *Communicating Process Architectures 2000*, pages 99–108, sep 2000.

[LRZa] LRZ. https://www.lrz.de/services/v2c_en/installations_en/. Stand: 2019-10-21.

[LRZb] LRZ. https://www.lrz.de/services/v2c_en/installations_en/cave_en/. Stand: 2019-10-21.

[LRZc] LRZ. https://www.lrz.de/wir/lrz-flyer/lrz-flyer.pdf. Stand: 2019-10-21.

[MFVR18] Ralf-Peter Mundani, Jérôme Frisch, Vasco Varduhn, and Ernst Rank. A sliding window technique for interactive high-performance computing scenarios. *CoRR*, abs/1807.00092, 2018.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[MvLvW98] Jurriaan D. Mulder, Robert van Liere, and Jarke J. van Wijk. Computational steering in the cave. *Future Gener. Comput. Syst.*, 14(3-4):199–207, August 1998.

[MvWvL99] Jurriaan D. Mulder, Jarke J. van Wijk, and Robert van Liere. A survey of computational steering environments. *Future Gener. Comput. Syst.*, 15(1):119–129, February 1999.

[NAS97] NASA. Stereo image taken by the imp of the rover, 1997.

[oOS09] Millodot: Dictionary of Optometry and Visual Science. Snellen fraction, 2009.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[Pre08] Cambridge University Press. Cambridge online dictionary, cambridge dictionary online, 2008.

[PvTB+07] Michael Pfaffinger, Christoph van Treeck, André Borrmann, Petra Wenisch, and Ernst Rank. An interactive thermal fluid simulator for the design of hvac systems. 2007.

[Str74] Wolfgang Strasser. Schnelle kurven- und flächendarstellung auf grafischen sichtgeräten. 1974.

[TGMB12] Clarence J. M. Tauro, Narayan Ganesan, Saumya Ranjan Mishra, and Anupama Bhagwat. Object serialization: A study of techniques of implementing binary serialization in c++, java and .net. 2012.

[VS97] J. Vetter and K. Schwan. High performance computational steering of physical simulations. In *Proceedings 11th International Parallel Processing Symposium*, pages 128–132, April 1997.

[WvTB+07] Petra Wenisch, Christoph van Treeck, André Borrmann, Ernst Rank, and Oliver Wenisch. 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*, 22(4):275–291, 2007.

[WvTR04] P. Wenisch, C. van Treeck, and E. Rank. Interactive indoor air flow analysis using high performance computing and virtual reality techniques. In *Proceedings Roomvent 2004*, 2004.

Statutory Declaration

With this statement I declare, that I have independently completed this Master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

München, 01. November 2019

Uğurcan Sarı

Uğurcan Sarı
ugurcan.sari@tum.de