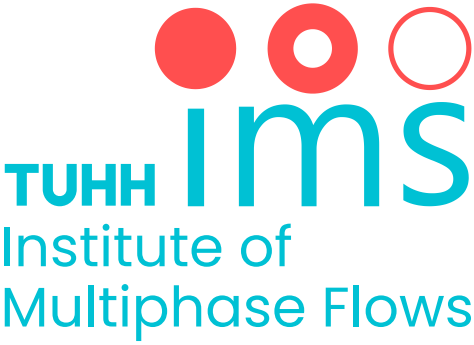

Master Thesis

Development of an Extended Reality Tool for CFD-aided Design and Development of a Shell-Tube Heat Exchanger

Jeswin Kannampuzha Francis, B.Tech.

First Examiner: Prof. Dr.-Ing. Michael Schlüter

Second Examiner: Prof. Dr. rer. nat. Andreas Liese

Supervisors:

Torben Frey, M.Sc.

Daniel Niehaus, M.Sc.

Hamburg, July 2023 - January 2024

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Hamburg, _____ Signature: _____

Contents

List Of Symbols

Symbol	Meaning	Unit
ρ	Density	$\mathrm{kg/m^3}$
ν	Kinematic viscosity	$\mathrm{m^2/s}$
∇p	Pressure gradient	Pa/m
$\mathbf{F}$	Force vector	N
$\mathbf{u}$	Velocity vector	m/s
k	Turbulent kinetic energy	$\mathrm{m^2/s^2}$
ε	Turbulent dissipation rate	$\mathrm{m^2/s^3}$
ω	Specific turbulent dissipation rate	$\mathrm{s^{-1}}$
R	Universal gas constant	$\mathrm{J/(mol \cdot K)}$
U	Velocity	m/s
T	Temperature	K
P_{rgh}	Dynamic pressure	Pa

Zusammenfassung

In dieser Arbeit wird eine digitale Umgebung vorgestellt, in der interaktive CFD-Simulationen (Computational Fluid Dynamics) integriert sind, die auf die fortgeschrittene Ausbildung in der Chemietechnik zugeschnitten sind. Die Architektur nutzt ein Client-Server-Netzwerk, das eine automatische bidirektionale Verbindung zwischen CFD-Solvern und einer Spiel-Engine herstellt. Bei der Entwicklung wird ein modularer Software-Ansatz verfolgt, der die Schaffung dauerhafter und miteinander verbundener digitaler Anwendungen gewährleistet. Um die Komplexität der Berechnungen und die ressourcenintensiven Funktionen zu überwinden, wurde ein Content-Delivery-Network aufgebaut, das die Endbenutzer von der Last der komplizierten Berechnungen befreit. Anhand einer Fallstudie mit einem Rohrbündelwärmetauscher wird die Leistung der Anwendung bewertet, wobei die Ergebnisse auf die Wirksamkeit von Augmented reality (AR) bei der Visualisierung von Simulationen untersucht werden. Die Diskussionen konzentrieren sich auf die Anwendbarkeit des Systems und skizzieren mögliche zukünftige Forschungsrichtungen und Anwendungsrichtungen

Abstract

This study introduces a digital environment integrating interactive computational fluid dynamics (CFD) simulations tailored for advanced education in chemical engineering. The architecture employs a client-server network, establishing an automated bidirectional connection linking CFD solvers and a game engine. The development adopts a modular software approach, ensuring the creation of enduring and interconnected digital applications. To surmount computational complexities and resource-intensive features, a content delivery network is structured, alleviating the burden of intricate calculations from end-user devices. A case study involving a shell and tube heat exchanger assesses the application's performance, with results scrutinized for the efficacy of augmented reality (AR) in visualizing simulations. Discussions centre on the system's applicability and outline potential future research and application directions.

1 Introduction

The European Union (EU) is currently funding large-scale research projects to help bring digital tools into the ever-changing field of education and training. The 2021–2027 Digital Education Action Plan shows this determination, which comes from the EU's desire to develop digitally skilled workers [Dig20]. This detailed plan lays out a long-term goal for creating a successful digital education ecosystem. It stresses the significance of augmented reality (AR) and virtual reality (VR) technologies, along with producing high-quality teaching materials and tools that are easy to use [Sol22].

The significance of computing tools is increasing, particularly in the fields of engineering and education. Computational fluid dynamics (CFD), computer aided engineering (CAE), and process modeling tools are considered fundamental components of engineering software. CFD involves utilizing computers to solve the fundamental equations of fluid flow (Navier-Stokes) along with conservation equations for energy, species, radiation flux, electric and magnetic potential, etc. This approach is employed to address practical problems in the real world. The models needed to address common unit activities, such as mixing, reaction, separation, drying, etc., exhibit significant variability in complexity and necessitate the incorporation of numerous sub-models to achieve equation closure. The universality, complexity, and level of validation of such models exhibit significant variations [Fle22].

Commercial CFD solutions typically need significant financial resources, making them unaffordable for individuals, small businesses, and educational institutions with limited budgets. Moreover, The

complex details of CFD modeling require specialist expertise, which poses a challenge for users who are not specialized in this field. The exclusive nature of financial and technological aspects presents a major obstacle to the widespread use of CFD applications, restricting their adoption in various industries and hindering the potential for innovation.

The incorporation of cutting-edge technologies like AR has the capacity to transform the depiction of simulation outcomes and comprehension in CFD simulations. AR is being utilized more and more as a disruptive tool in engineering analysis and simulation. This is done to address the restrictions that are commonly connected with traditional simulation settings, which are mainly designed for experts [Kim18]. AR is a revolutionary technology that enables users to observe simulation outcomes within the physical environment. It achieves this by offering a convenient and effortless method to engage with a technologically enhanced physical environment. This form of immersion enhances individuals' comprehension and analysis of intricate datasets by providing them with precise and captivating experiences, hence enhancing the effectiveness, importance, and engagement of the learning process [Gun21].

Shell and tube heat exchangers are often used and widely accepted equipment in many process industries. They are key components in critical sectors such as chemical and petrochemical plants, oil refineries, power plants, and metallurgical activities. These devices are essential in various processes, including heating, cooling, condensation, and boiling. Due to the widespread use of heat exchangers in various industries, they provide an ideal setting for the implementation of CFD simulations. By doing CFD analyses, it is possible to thoroughly evaluate the fluid flow and heat transfer occurring in these heat exchangers, providing useful insights for many industrial processes [Pal16].

The objective of this work is to establish a connection between

augmented reality and CFD simulations. This is an environment that uses simulations for learning and engineering shell and tube heat exchangers. The project seeks to improve traditional techniques for viewing and comprehending simulations while also transforming the utilization of fluid dynamics by incorporating CFD solvers into a user application created in a gaming engine, thereby making it widely accessible.

1.1 Research Goals

This thesis aims to tackle many objectives in the field of AR applications for CFD simulations. The main goal is to create and execute a user-friendly AR interface that improves accessibility for users with different levels of experience. This entails the creation of user-friendly controls and interactive elements, such as gesture-based interactions and manipulation of 3D models, in order to establish a smooth and captivating simulation environment. By reaching this goal, the study hopes to democratize the application of CFD models, making them accessible to a broader audience beyond experts in the area.

The second primary objective entails incorporating open-source software for CFD simulations with the purpose of mitigating budgetary obstacles linked to proprietary solutions. This involves the investigation and utilization of effective open-source technologies, evaluating their abilities, and verifying their compatibility with the AR interface. The goal is to create an efficient and robust basis for fluid dynamics simulations, encouraging the use of AR applications in educational institutions, small businesses, and research environments.

The final objective of this project is to create a specialized tool for the research and investigation of shell and tube heat exchangers. This involves developing a user-friendly and interactive application that utilizes CFD simulations within a game engine environment.

The goal is to create a thorough and user-friendly platform that enables a meticulous analysis of fluid dynamics principles, specifically designed for the complex operations of shell and tube heat exchangers.

2 Background And Review

This section offers a concise overview of CFD and AR. The text discusses the process of incorporating CFD and AR into the design of shell and tube heat exchangers. The paper presents a concise summary of the pre-processing, modeling, and post-processing procedures, while also emphasizing the challenges that occur. The tale explores the merging of real and enhanced elements in AR, emphasizing its transformative potential in industrial processes and educational environments.

2.1 Computational Fluid Dynamics

A fundamental concept behind a shell and tube heat exchanger is the movement of two fluids at distinct temperatures, which are kept apart by a partition. Due to the disparity in temperature, heat is transferred from a fluid at a higher temperature to a fluid at a lower temperature through the processes of conduction and convection. The movement of fluid on the shell side of a shell and tube heat exchanger is intricate. The presence of perpendicular inlet and outlet in relation to the main flow direction introduces intricacies in shell flow. Furthermore, the inclusion of baffles adds to the complexity of the flow. Therefore, it is advantageous to comprehend the flow pattern and have the capability to forecast the heat transfer process throughout a broad spectrum of heat load and mass velocities on both surfaces. Due to advancements in numerical methods and faster computers, scientists have increasingly adopted CFD as a rapid and effective tool for evaluating the distribution of flow and heat transfer in different heat exchangers [Pal16].

In the realm of CFD, the Navier–Stokes equation (NSE) holds a position of paramount importance. To streamline the complexity, The study focuses on the simplified NSE tailored for non-compressible fluids:

$$\frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho} + \nu \cdot \nabla^2 \mathbf{u} + \mathbf{F} \tag{2.1}$$

Here, $\frac{D}{Dt}$ represents the material derivative, indicating temporal and spatial changes. The variables are defined as follows: ρ for density, ∇^2 as the Laplace operator, ν for kinematic viscosity, ∇p as the pressure gradient, $\mathbf{F}$ for the force vector, and $\mathbf{u}$ for the velocity vector. The solution of this equation, achieved through methods like finite difference or lattice Boltzmann (LBM), provides crucial insights into pressure and velocity distributions. It's noteworthy that LBM equations, although distinct from NSE, offer a mesoscopic perspective, and the recovery of NSE from LBM is feasible through the Chapman–Enskog analysis. While the intricacies of this derivation won't be explored further in this context, they remain integral to understanding the fluid dynamics principles under investigation [Pal16].

Figure 2.1 illustrates the overall workflow of a computational fluid dynamics simulation. A simulation consists of three primary components: pre-processing, modeling, and post-processing. In the pre-processing stage, it is crucial to determine the physical properties of the fluid, generate the mesh, and establish the control volume. Discretization is a crucial process in numerical analysis that transforms a continuous fluid domain into a collection of finite components. During the simulation phase, the solver utilizes the input data to create a mathematical model that accurately represents the movement of mass, fluid, and heat. This stage will teach you how to comprehend intricate fluid dynamics scenarios, including the interaction of fluids with various objects and flow patterns in heat exchanges, reactors,

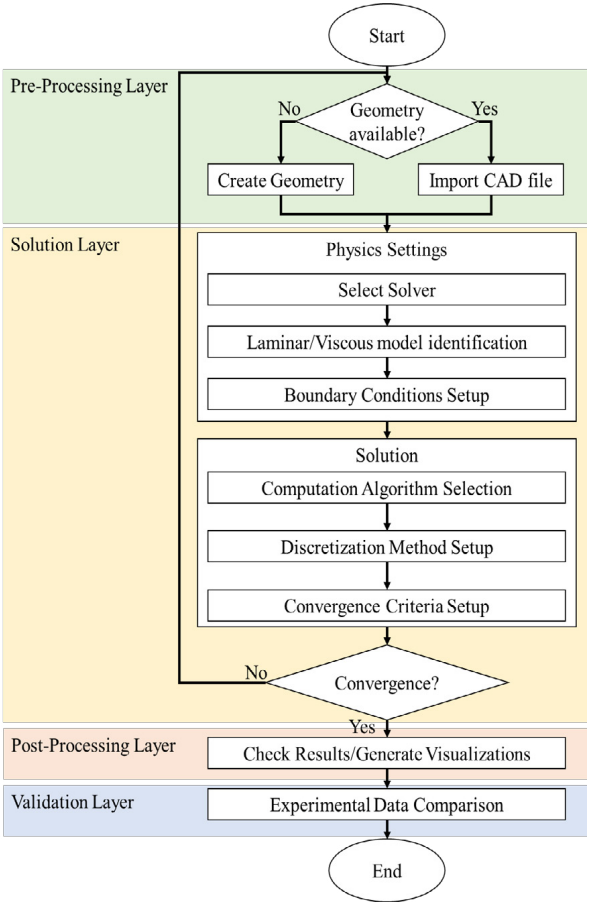

Figure 2.1: CFD workflow depicted in the literature [Mou22].

and chemical reactions. During the post-processing phase, scientific visualization tools are employed to comprehend and present the simulation outcomes, often provided in the form of datasets. The primary

objective of this stage is to provide a lucid and uncomplicated depiction of intricate fluid flow phenomena. [Bor08].

Software	Specialty
ANSYS Fluent	Versatile commercial software with a broad range of physical models, robust pre-processing capabilities, and advanced post-processing tools. Known for its user-friendly GUI.
COMSOL Multiphysics	Multiphysics simulation, allowing users to couple different physical phenomena. It excels in simulations involving the interaction of multiple physics, such as fluid flow coupled with heat transfer or structural mechanics.
OpenFOAM	Open-source software provides flexibility and customization for solving complex fluid dynamics problems. Known for its extensibility, it's widely used in research and industries where customization is crucial.
STAR-CCM+	Commercial software with excellent parallel processing capabilities. It is known for its ease of use, making it accessible for complex simulations and offers detailed post-processing and visualization tools.
Autodesk CFD	Integrates seamlessly with CAD tools, simplifying geometry preparation and post-processing. It's suitable for engineers working within the Autodesk design environment.

Software	Specialty
SimScale	A cloud-based simulation platform that supports CFD, FEA, and thermal analysis. It is accessible through a web browser, making it suitable for collaborative and remote simulations.

Table 2.1: Overview of Popular CFD Softwares.

As a modeling and visualization tool, CFD is used a lot in engineering and many other fields. Analyzing building HVAC systems, looking into how things interact with air, water, or particle mixtures around them, studying combustion in boilers or propulsion systems, and simulating complex flows in heat exchanges and chemical reactors are some examples. The most well-known commercial and open-source programs currently dominating the market are included in the table 2.1. ANSYS Fluent and COMSOL Multiphysics are renowned commercial CFD software solutions that offer a comprehensive range of capabilities, including turbulence modeling, heat transfer, and chemical reactions. Open-source options like OpenFOAM provide a flexible and extensible framework, allowing users to customize simulations and access the source code. STAR-CCM+ by Siemens is recognized for its parallel processing capabilities and ease of use in complex simulations. Autodesk CFD integrates seamlessly with CAD tools, simplifying geometry preparation and post-processing. Each software package has its strengths, catering to diverse user needs and preferences in terms of user interface (UI), simulation features, and computational efficiency. Choosing the most suitable software depends on factors such as the complexity of the simulation, user expertise, and specific application requirements.

Although CFD simulations have some advantages, such as lower prices, quick responses to changes in the design, and full modeling of data and conditions, there are also some problems to think about. It takes a long time to compute for big domains, and mistakes happen

when flow models and boundary conditions are oversimplified [Sil21]. Problems like the expert-centric design of conventional simulation platforms, the requirement for advanced abilities, and the constraints of conventional learning settings must also be resolved in order to properly use CFD simulations in education [Sol23a].

2.2 CFD Visualisation With AR

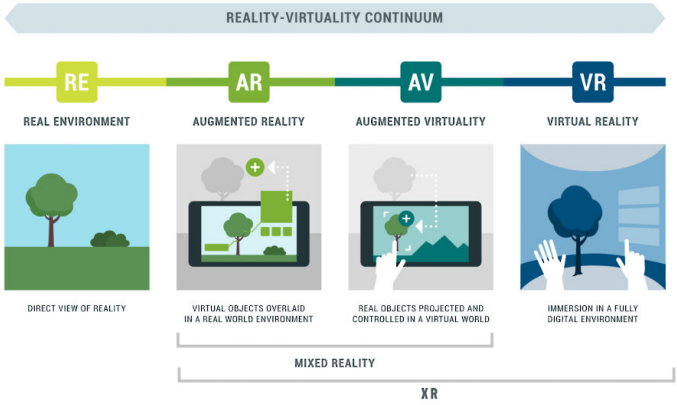

Figure 2.2: Relation between XR technologies [FB18].

Extended reality (XR) comprises a wide range of technologies, such as VR and augmented Virtuality, commonly known as mixed reality (MR). MR is the complex integration of real and virtual aspects, where the virtual enhances the real and vice versa. Augmented virtuality, a concept within the field of AR, enhances the degree of virtual experience by incorporating more synthetic features. This approach enables the development of unique ways to visualize

and demonstrate new products and procedures. VR allows users to be completely immersed in a virtual world, separate from the real environment. This technology is useful for several tasks, including product development, marketing, training, ergonomics, and visualizing digital factories during the design process.

AR is a revolutionary technology that enhances the user's visual experience by seamlessly incorporating task-specific information into their actual surroundings. AR seamlessly integrates actual and virtual things, allowing for interactive and real-time experiences while ensuring precise alignment of these aspects. Due to its capacity to enhance human cognitive functions and memory, this advanced technology is widely used in several fields, including remote guidance, instructional visualization, and complex assembly [FB18].

Figure 2.3: HMD [Kir22]

Figure 2.4: HHD [Gru23]

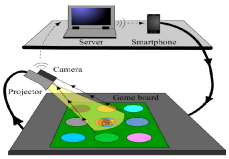

Figure 2.5: P-SAR [Fra11]

Figure 2.6: Examples of AR devices.

The devices used for AR displays are classified into three primary categories: head-mounted displays (HMD), hand-held devices (HHD), and spatial displays such as desktop and projection displays. Examples of these devices are shown in figure 2.6 [Li17]. Microsoft hololens 2 and magic Leap 2 are both instances of head-mounted display (HMD) AR systems. Additionally, many smart mobile phones and tablets are also classified as handheld AR devices. Projector-based spatial augmented reality systems are established by employing cus-

tomized cameras and projectors to generate engaging augmented reality experiences.

AR has proven to be useful in the field of chemical engineering on a global scale. The center for innovation through visualization and simulation at Purdue university Calumet utilizes AR to integrate CAE with lifelike 3D visuals. This integration improves understanding of complex industrial processes, such as power plant training [Mor13]. Furthermore, the university of Rochester has investigated the use of AR in creating virtual representations of chemical reactor settings, employing common objects such as coffee mugs [Gun21]. An impressive undertaking entailed the development of a virtual and immersive crude distillation unit (CDU) for undergraduate classroom teaching. This comprised the integration of interactive activities and virtual plant tours. A quantitative user research demonstrated a favorable influence on students' comprehension of CDU procedures [Kum21].

The understanding of scientific and engineering principles, such as CFD, is difficult because of their abstract nature. Visualization has a vital role in improving comprehension and reducing misunderstandings. Although there has been significant focus on the accuracy and convergence of numerical analytic methods in CFD simulations, the investigation of visualizing CFD simulation data has been relatively restricted in academic research. There is a gradual transition towards the utilization of AR and VR in this field, but it is still in its initial phases [Kim21].

There are multiple factors impeding the widespread adoption of AR/VR as a method for strengthening CFD. One of the reason is the need for real time simulations that to be visualised instantly. The inherent trade-off between precision and the speed at which frames are processed poses challenges for real-time CFD simulations. The present state of the field classifies accuracy levels based on physics-based and fluid-like modeling methodologies, as indicated in Figure

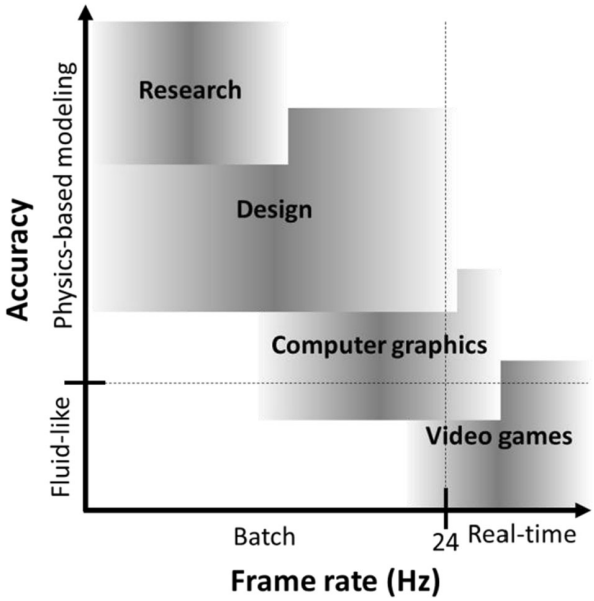

Figure 2.7: Continuum of CFD simulations [Sol21].

2.7. Real-time simulations, requiring a minimum of 24 frames per second, have demonstrated potential in several applications, but their effectiveness in complex multi-physics CFD models remains limited. Presently, the majority of CFD simulations are conducted in a batch manner, followed by the visualization of the obtained findings [Sol21]. The exorbitant expense of AR/VR technology serves as a limitation on the widespread adoption and utilization of these gadgets. An augmented reality gadget like the microsoft holoLens 2 has a price tag of approximately 3500 euros, making it unaffordable for the average individual. Furthermore, there are fewer user applications that specifically focus on conducting CFD simulations that are accessed via this technology. The development of such applications is expensive and

necessitates a proficient comprehension of programming and game design. One of the goals of this thesis is to create an open-source application with minimal coding.

Traditional post-processing environments provide difficulties, particularly for novice users such as engineering students. The rapid growth in computer capacity and the incorporation of AI offer great potential for the future of CFD simulations. Many instances in the literature demonstrate efforts to visually represent CFD simulation outcomes in an interactive manner, starting with civil and architectural design and later expanding to chemical engineering. In earlier research, [Mal05] integrated CFD with AR to enhance simulation boundary conditions by incorporating sensory input. AR visualization has made progress by utilizing cloud computing to enable real-time monitoring and by customizing the outcomes to be visualized on different platforms [Kim18].

The researchers in [Sch21] combined CFD with mixed reality displays to showcase the real-time processing of data using machine learning algorithms for simplifying 3D meshes. A different research conducted by [Mor21] suggested the use of projector-based spatial augmented reality (P-SAR) as a means of facilitating natural interaction with simulations using gestures and tool manipulations.

The epidemic has placed significant emphasis on adapting extended reality technologies in education. A study made efforts to achieve platform-independent learning by utilizing Hololens technology [Bha22]. The openvisFlow framework developed during this time enhanced the visualization of fluid flow in simulation code. This framework integrates markerless visualization capabilities based on AR-Core for smart phones and tablets [Teu22].

A recent study highlighted the structural framework of AR/VR environments and proposed a system design that utilizes components and incorporates an automated data coupling technique. This tech-

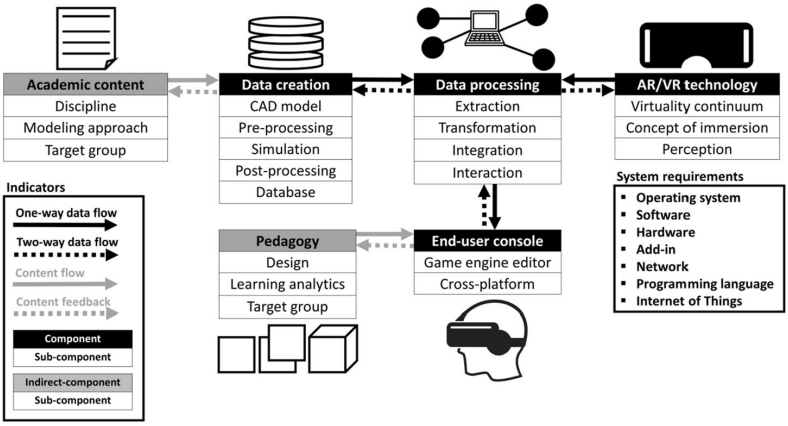

Figure 2.8: Generic system architecture to integrate CFD simulations with AR/VR [Sol21].

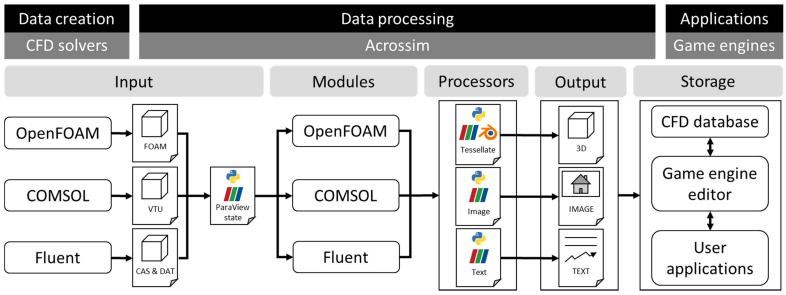

Figure 2.9: Acrossim toolkit for integrating CFD simulation data into game engines [Sol23b].

nique allows for the smooth integration of various platforms and workflows [Sol21]. The integration technique utilized a content delivery network to enhance communication between digital platforms, facilitating the creation of interactive augmented reality applications

using computational fluid dynamics in the domain of chemical engineering. Figure 2.8 illustrates the suggested basic framework, which serves as a foundation for future studies focusing on creating a user application that connects computer aided engineering simulation tools with the AR/VR platform [Sol22]. A similar study [Sol23b] developed a python based tool for integrating CFD simulation data into game engines. This modular, component-based toolkit incorporates data from widely used CFD solvers in engineering as shown in Figure 2.9.

These examples highlight the promise of AR as a transformative technology in industrial settings. AR goes beyond basic visualization and serves as a powerful tool for teaching, training, and operational knowledge in different industries.

3 Methodology

When considering the system architecture for this thesis, it is crucial to examine several architectural choices in order to find the best balance between computing efficiency and user engagement in the AR application. The selected architecture must effectively combine the functionalities of standard augmented reality devices with the necessary processing capacity for conducting CFD simulations. An exhaustive assessment of client-server models, peer-to-peer architectures, and other distributed systems is required to determine the best suitable solution. The major goal is to improve the user experience by transferring resource-intensive operations to a backend system while still preserving the lightweight and portable characteristics of generic AR devices. The chosen architecture facilitates a smooth and instantaneous transfer of data between the AR device and the computing backend, promoting an engaging and participatory simulation experience.

Within the domain of information technology, a range of client-server architectures are designed to meet different system needs and operating frameworks. A commonly used classification is the unidirectional client-server architecture, also known as one-way architecture, in which the client initiates queries and the server answers accordingly. This concept is commonly found in conventional online applications, where users engage with a server to obtain information or carry out actions. Conversely, the bidirectional client-server design facilitates communication in both directions. The exchange of data in both directions is most noticeable in applications that require real-time processing, collaborative platforms, and sophisticated interactive systems, enabling the dynamic interchange of information

between clients and servers. The two-way architecture is commonly linked to technologies like WebSockets, which provide uninterrupted communication between clients and servers. These varied architectures provide IT professionals with the option to select the most appropriate model according to the individual requirements of the application, prioritizing either simplicity and direct communication or a more dynamic and interactive data interchange.

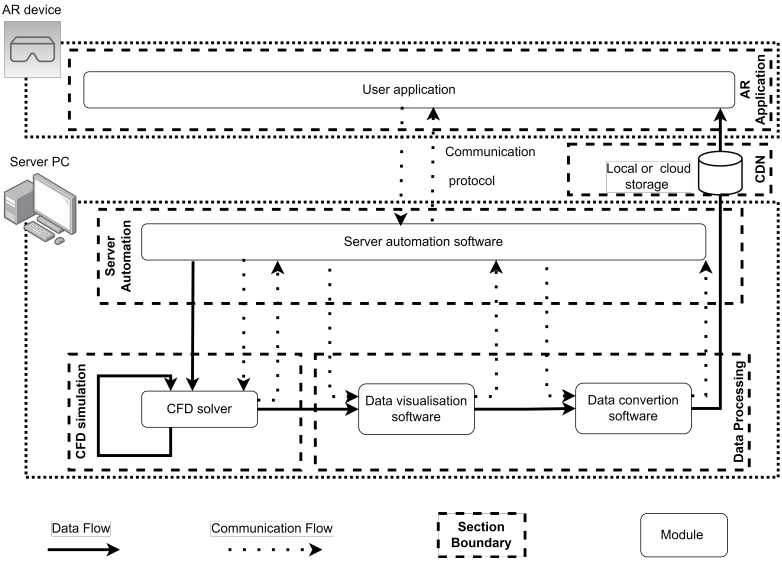

Figure 3.1: System architecture proposed by the study.

One of the earlier studies has already explored a general architecture for creating such a system, which is depicted in Figure 2.8. In this section, he suggests a modular and adaptable design that may be tailored to the application's individual requirements. This is an excellent starting point for anyone creating a client-server system for integrating process software with AR/VR visualization. This

architecture has also inspired the architecture proposed in this study. There were also other elements to consider for the system's creation, such as the type of data to be communicated, the communication protocols to be employed, and so on.

This study's client-server architecture is depicted in Figure 3.1. This architecture was created with all of the aforementioned features in mind, and it is quite effective for building this system with minimal coding and a highly efficient presentation of the simulation results. The network consists of six data connections and eight communication channels. The client-server architecture is partitioned into different parts, each of which possesses its own distinctive functions. The server automation section deals with the comprehensive management and coordination of a server computer and its interactions with many software applications. The CFD simulations section performs simulations and generates simulation datasets. The data processing section is responsible for processing the CFD datasets, which involves visualizing the data and converting it into appropriate data formats. The content delivery system facilitates the transport of 3D models from the converted data to the AR device. Subsequently, the AR application relays user input to the server PC and retrieves the final findings from the simulation.

3.1 Server Automation

The automation of server duties is an essential necessity, and there are multiple approaches that may be utilized depending on the specific conditions and preferences. Scripting languages such as Bash, Python, or PowerShell are frequently employed to automate repetitive tasks on servers. These scripts are programmed to run at regular intervals, guaranteeing the seamless execution of recurring activities. Another method involves utilizing node-based programming paradigms, which provide a user-friendly graphical interface for creating and editing

automation flows without requiring substantial coding knowledge. The flexibility of this system is beneficial for coordinating intricate server operations and incorporating several services, protocols, and APIs.

3.1.1 Server Automation Software

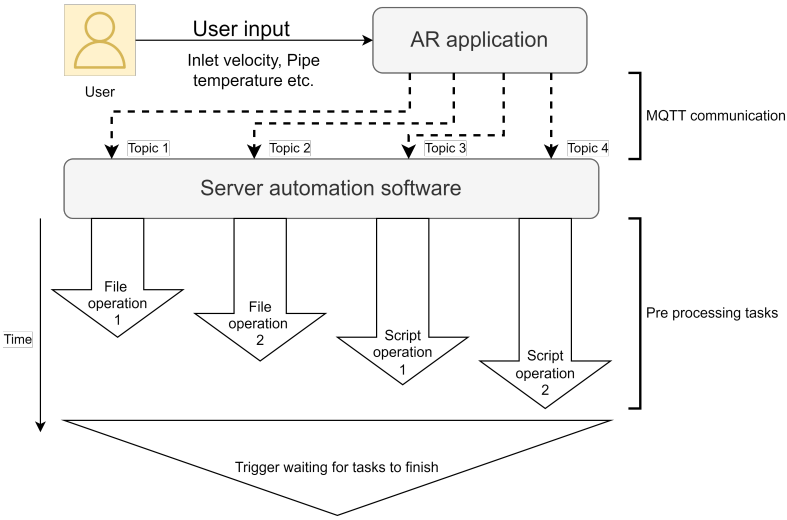

Figure 3.2: User input processing in the system through automation software.

Automation is a crucial component of this study as it enables the server to respond in accordance with the user input received from the AR application. These are accomplished through several methods, with two prominent approaches being the use of scripts and visual-based programming. Scriptwriting typically necessitates the expertise of a professional, and this research endeavours to di-

minish the intricacy associated with constructing such settings. The automation solution is recommended to be both open-source and compatible with the operating system. Automation software possesses the capability to initiate and regulate other software programs. Additionally, it performs essential file actions and provide feedback upon completion of a task as shown in Figure 3.2. It has the capability to incorporate the communication protocol employed by both the AR device and the server PC. To enhance automation capabilities beyond traditional scripts, the study adopted an open-source visual-based programming tool called Node-red, leveraging its advantages. The chosen automation software is implemented on the server PC. This software is initiated through the terminal, providing access to a browser-based graphical user interface (GUI).

The automation software incorporates a variety of built-in nodes representing different tasks relevant to the project, allowing for seamless interconnection and automation. In this study, distinct flows were designed to oversee the overall simulation process and post-processing, as well as to manage MQTT communication and essential preprocessing for simulations. The framework's functionality includes terminal access through nodes, enabling the execution of commands directed towards various components, such as CFD solvers and containers of the software. Additionally, it supports the execution of python scripts for essential operations in tools for data processing.

The software utilizes a template node to modify files dynamically based on the payload from commincation channel, facilitating adaptive adjustments. Multiple communication nodes correspond to specific topics, each representing parameters requiring modification in the simulation. For example, a topic might involve altering the temperature of the inlet in the simulation. Upon receiving messages in the communication channel, the flow transitions to task nodes, such as changing velocity values in a file. Measures have been implemented to prevent simulation initiation until all preprocessing is complete.

The automation software includes error-handling capabilities facilitated by a function node supporting javascript programming. These functions prevent the execution of subsequent steps in the event of an error and enable the relay of error messages back to the AR application, informing users of any issues.

3.1.2 Communication Protocol

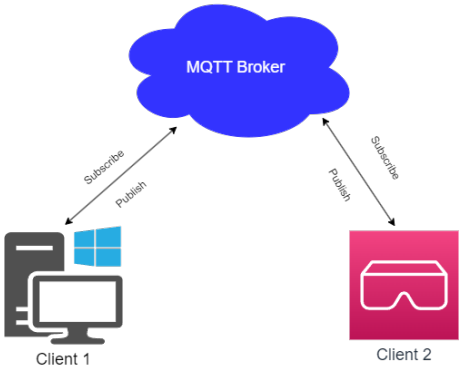

Figure 3.3: Architecture of MQTT communication protocol.

Different communication protocols enable the transfer of data between servers and clients in the information technology (IT) environment, with each protocol designed to meet distinct needs. Hypertext Transfer Protocol Secure (HTTPS) is commonly used to ensure secure communication over the Internet. It achieves this by applying encryption methods such as TLS/SSL. Message Queuing Telemetry Transport (MQTT), is particularly notable for its effectiveness in situations that demand efficient, instantaneous, and dependable communication. Contrary to the request-response mechanism of HTTPS,

MQTT functions based on a publish-subscribe paradigm as shown in Figure 3.3, enabling clients to subscribe to particular topics and get updates whenever pertinent information is released. MQTT's asynchronous and event-driven nature makes it well-suited for applications that require frequent updates and dynamic data, such as real-time scenarios like augmented reality simulations. Moreover, Considering the specific AR application mentioned, where immediate engagement and timely updates are of utmost importance, MQTT stands out as the optimal solution due to its lightweight characteristics, efficient publish-subscribe framework, and compatibility with dynamic, event-driven communication.

3.2 CFD Simulation

Choosing a suitable CFD program for this thesis requires carefully considering multiple criteria to ensure that the software's capabilities align with the unique requirements of the research endeavour. The complexity of the simulations and the necessary degree of customization are crucial factors. Open-source alternatives, such as OpenFOAM, provide a wide range of customization options but may need a more challenging learning process. Commercial software such as ANSYS Fluent or COMSOL Multiphysics typically offers a more user-friendly interface with pre-designed features that are ideal for a wide range of simulations. It is important to examine the computational resources that are available, as some applications may be designed to work well with parallel processing, which directly affects the performance of simulations. Another crucial factor to consider is the integration compatibility with supplementary products, such as Computer-Aided Design (CAD) software or post-processing apps like Paraview. When deciding between open-source and commercial alternatives, financial limitations may also be a factor. Open-source alternatives are generally more cost-effective, however they may require additional tools to achieve a comprehensive simulation pipeline.

In addition, it is important to recognize the significance of having a well-developed support infrastructure and a strong user community. These elements greatly enhance troubleshooting capabilities and deepen overall understanding. Thoroughly examining these many factors assisted in identifying openFOAM as the most suitable CFD software for the specific requirements and objectives of the augmented reality application.

3.2.1 Geometry And Mesh Setup

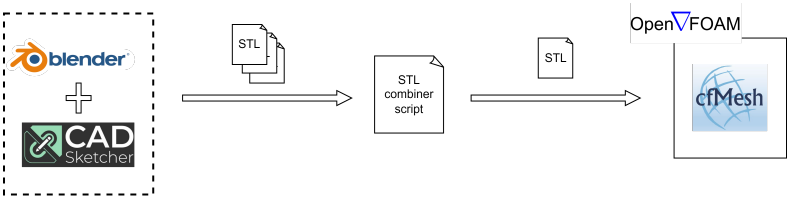

Figure 3.4: Example mesh creation from the 3D geometry files for opensource CFD solvers.

The intricate relationship between geometry and meshing processes is often seamlessly integrated into commercial software solutions. However, The adoption of an open-source approach necessitated the selection of distinct tools for each phase. In the realm of 3D modeling for geometry creation, various open-source software options are available, among which FreeCAD and Blender stand out. These platforms facilitate the development of intricate geometries, subsequently converted into suitable formats for meshing within open-source environments. Notably, for meshing in the context of openFOAM, prominent tools include cfMesh and snappyHexMesh, renowned for their ability to automate mesh generation with minimal manual intervention. The adaptability of these meshing tools allows for tailored adjustments to meet the specific requirements of diverse simulations.

Additionally, the software affords mechanisms for inspecting crucial mesh parameters, such as non-orthogonality and skewness, ensuring the mesh's compliance with predefined quality standards. In this study, Blender, a widely recognized program equipped with an extensive array of plugins for geometry creation, was selected. The strategic choice of minimizing software dependencies within the thesis serves to streamline the overall process, contributing to a reduction in complexity. Moreover, this decision aligns with the overarching goal of leveraging Blender, a software employed for various other aspects of the thesis, promoting cohesion and efficiency in the research workflow. The STL files were methodically created using blender to accurately depict the many geometries required for the ultimate use. The several files were later combined into one comprehensive STL file as shown in Figure 3.4.

3.2.2 Automated CFD Routines And Data Extraction

Automating CFD routines is crucial for optimizing the simulation process, removing the requirement for human data input, and expediting the overall workflow. Traditionally, in the simulation environment, the process of specifying initial values and boundary conditions, as well as selecting the solver, has been done manually. Manually configuring this process is a lengthy task, as it involves updating individual files that are relevant to the simulation. This is challenging because some open-source softwares does not have a GUI. The standard procedure entails choosing an appropriate solution according to the characteristics of the geometry, flow, and other relevant factors. Although the flexibility enables substantial customization, the manual editing of files requires a more efficient method.

This is the second automated task in this study, which shares similarities with the automation of the server PC. The thesis tries to automate chores in a simple manner as shown in Figure 3.5, and it once again chooses a visual-based programming technique. Scripts

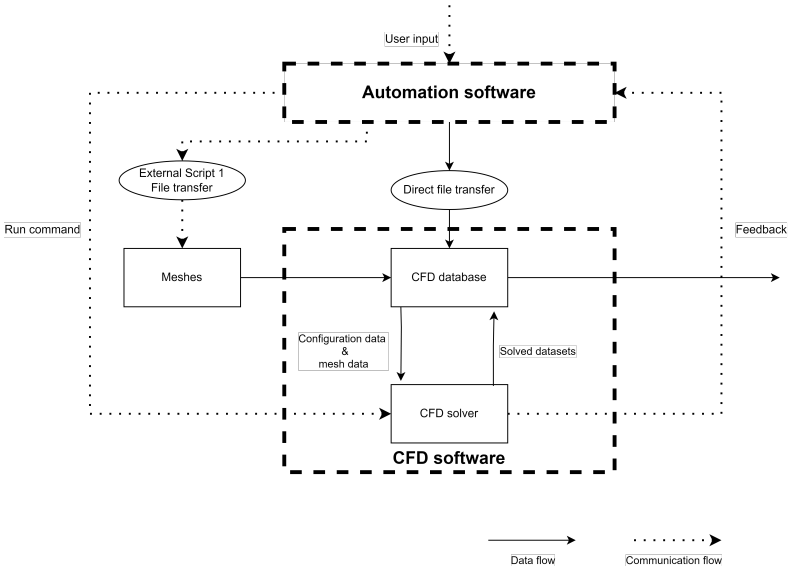

Figure 3.5: Interaction between CFD solver and automation software.

were developed to configure various parameters of the simulations, and automation software is employed to modify and execute these scripts as required. In order to accomplish a complete integration of the AR gadget and server PC, it is necessary to automate all manual activities. Node-RED is an excellent choice for this assignment due to its built-in template nodes and its ability to be customized to meet specific requirements.

3.3 Data Processing Software

The CFD datasets generated by the CFD solvers are not immediately suitable for direct visualization in an AR application. The CFD data is initially shown in accordance with the specific requirements

of the investigation. Many scientific visualizations are employed to assess the simulations. Heatmaps and streamlines are among the available options. Data visualization in open-source software such as OpenFOAM is accomplished using other open-source tools specifically designed for data visualization. Paraview is a highly esteemed software in this domain. Data visualisation software transforms CFD datasets into coherent and informative visual representations in either two-dimensional or three-dimensional formats. Oftentimes, the 3D assets generated by the data visualization tool are not compatible with the game engine for visualization in the AR application. The components developed have a larger file size due to the complexity involved in the visualization process. It is necessary to decrease the dimensions of the visualization elements and generate compatible files for the AR application. A different data conversion software is utilized to do this. Examples of such software include Blender, which specializes in 3D modeling. Specific toolkits are designed to integrate this tasks without much complexity. Acrossim is one of such toolkits that are apt for integrating CFD simulation data into game engines as discussed in section 2.2.

The post-conversion data format is critical as it must preserve all the vital aspects of the CFD visualisation. The success of the study depends on the compatibility between the data format and the content delivery system and game engine, which results in accurate outcomes. Prior studies have utilized several formats, such as FBX and X3D. After a systematic process of trial and error, it was concluded that GLTF 2.0, specifically with a .glb extension, was appropriate for the present inquiry. This format enables the preservation of all pertinent information of the visualization, including geometry, normals, texture, etc., in a unified file, without any loss of data.

3.4 Content Delivery System

The main objective of designing a resilient content delivery system is to facilitate online updates of the information and promote interactiv-

ity inside the application. In the absence of such a system, developers are obliged to include 3D data within the application itself, so restricting the possibilities for immediate updates and interactive user experiences. Furthermore, the content distribution system functions as a storage facility for safeguarding valuable content, enabling its retrieval when necessary and preventing the duplication of simulations with similar parameters. In order to achieve smooth integration with AR devices, the content delivery system must offer accessibility via APIs, enabling effective communication between the AR application and the stored material. The importance of a well-designed content distribution system is emphasized in improving the functionality and responsiveness of the AR application.

A Content Delivery Network (CDN) utilizing addressable resources stands out as a robust option for seamless content delivery and efficient asset management. However, its intricate setup may present challenges for those seeking a streamlined solution. Alternatively, cloud storage services such as public cloud storage options offer reliable alternatives. Nevertheless, opting for a self-hosted, open-source platform provides notable advantages, including greater control over data, enhanced privacy, and cost-effectiveness. The seamless integration with various protocols and APIs ensures compatibility with the AR application's data retrieval needs, makes Node-red an appealing choice for developers prioritizing reliability and user-centric content delivery.

Creating a robust database is essential for maintaining the efficient operation of a client-server architecture, serving as an organized storage system for data retrieval and storage. While more complex applications may require relational databases or distributed systems, simpler scenarios benefit from a file-based system, linking simulation parameters and outcomes with distinct file names for easy retrieval and storage.

In the initial project setup, a local server was established using a

server application, with a designated folder serving as the server space. Data was stored in this folder, and files were downloaded within the local network range. However, to enable access from anywhere globally with an internet connection, a transition to cloud storage became imperative. In the current solution, after postprocessing, the generated files are sent to cloud storage.

3.5 Design Of User Application

The implementation of this study is highly dependent on attaining an exact and accurate depiction of simulation outcomes. This is achieved by seamlessly integrating visualizations of CFD data into the AR application. The creation of an AR application depends on the specific platform being used. For example, AR devices such as the microsoft hololens 2 operate on the universal windows platform (UWP). Therefore, the AR application needs to be customized to be compatible with this platform, guaranteeing the best possible user experience. The development of interactive elements to improve the usability of the program requires a strong focus on human-centric design concepts. Additionally, the selected AR development software enables efficient development with limited coding requirements, while also being compatible with the data format utilized in the content distribution system.

Game engines, widely recognized for their ability to create interactive 3D applications, are becoming increasingly necessary technologies for satisfying these demands. While open-source alternatives like Godot are available, their capabilities may not be adequate for creating sophisticated AR experiences. The unity game engine, which is widely acclaimed for its use in developing AR applications, is highly regarded as an exceptional choice. The reasons why unity is a chosen choice include its widespread popularity, compatibility with UWP, provision of prebuilt UI elements for simpler development,

and support for complicated interactions through C-sharp coding. Moreover, Unity's adaptability is further amplified by the abundance of extensions that simplify the manipulation of data formats intrinsic to simulation.

The interface of the AR application is crucial in displaying all adjustable parameters of the simulations carried out in the study. The user inputs are subsequently transformed into the necessary format for transmission over a communication protocol to the server PC. The received data, after being translated, is smoothly integrated into the application without any notable delays as seen in Figure 3.2, following its transmission via the content delivery network (CDN). The data visualization includes essential interactivity to augment the user experience beyond conventional 2D screen observation. The AR application offers comprehensive information regarding the simulation solver employed and other crucial elements. Implementing a feedback loop guarantees immediate notifications regarding the ongoing activities on the server PC, hence improving the user's comprehension of the simulation's advancement.

4 Implementation

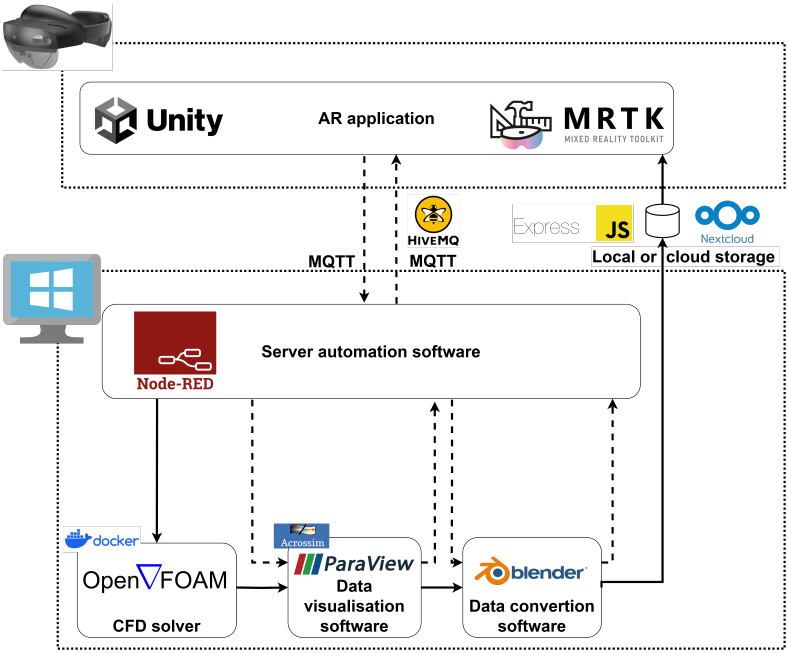

Figure 4.1: System architecture depicting the interconnection between softwares in this study.

Microsoft HoloLens 2 serves as the AR hardware interface in this project, flawlessly combining virtual simulations with the physical surroundings. The server PC is a Dell-based PC running Windows

11 Pro v22H2 64-bit and powered by an Intel Core i9-10900X CPU @ 3.70GHz with 64 GB RAM and an NVIDIA GeForce RTX 3080 10GB graphics processor. For extensive simulations, the software stack includes strong tools such as the open-source CFD software OpenFOAM v2212 inside a docker v23.0.5 container. The simulation operations are automated in Node-RED v3.0.2, assuring efficiency and reproducibility. 3D visualization is handled by ParaView v5.8.0 and Blender v3.5.1, which transform raw simulation data into interactive models. Unity v 2022.3, in conjunction with MRTK 2.8.0, enables the creation of an AR application that allows users to control simulation parameters and gain real-time insights into heat exchanger behaviour. Visual studio community 2022 was used for the deployment of a unity solution to the Hololens 2. Next cloud was used for cloud storage and Hive MQTT was used for the MQTT broker. This holistic integration of hardware and software contributes to advancing the field of heat exchanger simulation in chemical engineering. The connection between all these softwares inside system architecture is depicted in Figure 4.1.

Development Target	**Software**	**Purpose**
AR device	Unity	Designing the user interface and creating the AR application.
	MRTK	Mixed Reality Toolkit for Unity, used for UX design in Hololens app.
	Visual Studio	Integrated Development Environment (IDE) for building the Hololens app.
	GLTFUtility	Extension to load GLTF format files in Hololens 2.

Development Target	**Software**	**Purpose**
Server PC	OpenFOAM	CFD software for simulating fluid flow.
	Acrossim	Python script tool for interacting with Paraview and Blender APIs for post-processing and converting results to 3D formats.
	Blender	3D modelling software for creating geometry files for CFD simulations, post-processing, and converting files to GLTF format.
	Paraview	Visualization and post-processing tool for CFD simulation results.
	Nextcloud	Cloud drive provider for storing simulation files online.
	Express Server	Back-end framework for creating a local server.
	HiveMQTT	Online MQTT broker for communication between devices.
	Node-RED	Flow-based development tool for automating tasks on the server PC.
	Docker	Platform for developing, shipping, and running applications in containers. Used to containerize OpenFOAM.
	Miniconda	Package management system used for setting up Acrossim.

Table 4.1: Software tools used in the thesis and their purposes.

4.1 CFD Simulation

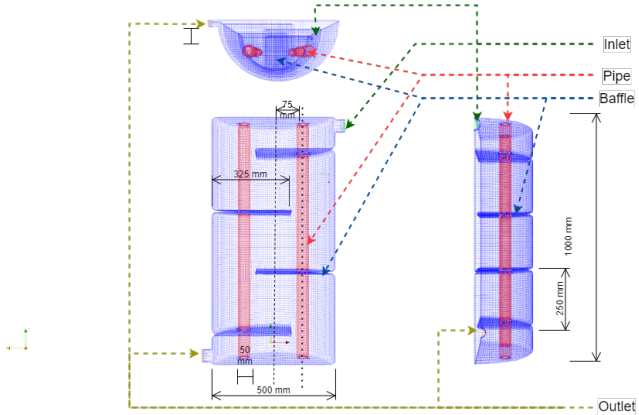

Figure 4.2: Mesh created for a shell and tube heat exchanger with 4 pipes and 4 baffles.

Parameter	HE12	HE14	HE22	HE24
No. of pipes	2	2	4	4
No. of baffles	2	4	2	4
Baffle spacing	500 mm	250 mm	500 mm	250 mm
Pipe spacing	250 mm	250 mm	250 mm	250 mm
Cells	228,864	247,120	242,672	261,124
Aspect Ratio	20.8745	19.4418	22.1036	22.46
Max Non-orthogonality	62.3735	60.9023	62.3714	60.8643
Max Skewness	2.44629	2.24974	2.44075	2.25647

Table 4.2: Mesh parameters for different configurations.

Parameter	Patch	Boundary Conditions
Temperature	Wall	Zero gradient
	Symmetry	Symmetry
	Pipe	Fixed value
	Outlet	Zero gradient
	Inlet	Fixed value
Velocity	Wall	No slip
	Symmetry	Symmetry
	Pipe	No slip
	Outlet	Fixed value
	Inlet	Zero gradient
Pressure	Wall	Zero gradient
	Symmetry	Symmetry
	Pipe	Zero gradient
	Outlet	Zero gradient
	Inlet	Fixed value
Thermal diffusivity	Wall	Zero gradient
	Symmetry	Symmetry
	Pipe	Jayatilleke thermal wall function
	Outlet	Zero gradient
	Inlet	Zero gradient

Table 4.3: Boundary conditions of different parts in the mesh.

The primary objective of this thesis is to explore the viability of a system architecture that seamlessly integrates CFD with AR. To achieve this goal, there is a trade-off between the complexity of the geometry and meshing quality, with a focus on representing a shell and tube heat exchanger. The selected geometry comprises a shell with dimensions of 1000 mm length and 500 mm diameter, featuring inlets and outlets on opposite sides. Inlets, outlets, and pipes have a diameter of 50 mm and are strategically placed within the configuration. Baffles, 325 mm in height (0.65 times the shell diameter), are utilized. Four different geometries are created, allowing users to

choose based on specific requirements. One of the four geometries is depicted in Figure 4.2. Mesh parameters of the four meshes are given in table 4.2 and selected boundary conditions are given in the table 4.3. Geometry was divided into different patches such as inlet, outlet, pipe, wall and symmetry based on the requirement for meshing. Baffles were fused with the wall of the shell to decrease the complexity of the simulation. Computational efficiency is enhanced by halving all geometries and a patch named symmetry represents this in the geometry.

The study performed CFD studies using the open-source program openFOAM. Although openFOAM demonstrated its efficacy for CFD analysis, its extensive process necessitated the incorporation of supplementary software components. To guarantee the ability to replicate and deploy on other platforms, a docker container was utilized. Container is basically a virtual machine created to serve the software as a package.The utilization of containerization enabled the encapsulation of openFOAM and its dependencies, guaranteeing a uniform environment for CFD simulations. The container was instantiated from a configuration file, facilitating the effortless replication of the entire system at another location by constructing the identical container with the identical settings.

The shell and tube heat exchanger was simulated using a buoyantBoussinesqSimpleFoam solver. OpenFOAM's buoyantBoussinesqSimpleFoam solver is specifically developed to address simulations involving buoyancy-driven flows with temperature fluctuations, making it well-suited for scenarios such as heat exchanger simulations. In the present study, buoyancy dependence was not taken into account and necessary changes in the configuration were made to remove this dependency. The simulation configuration is tabulated in the table 4.4. User-configurable parameters such as pipe temperature and inlet flow velocity are accessible through the AR application integrated into CFD database using a automation software. The thesis focuses on simulation and visualization, omitting validation

Parameter	**Value**
Fluid	Water
Flow	Laminar
Laminar viscosity	$1.00 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}$
Laminar Prandtl number	6.7
Scheme	SteadyState
Grad scheme	Gauss Linear
Solver	PCG (Preconditioned Conjugate Gradient)
Preconditioner	DIC (Diagonal Incomplete Cholesky)
Relaxation factors	
p_{rgh}	0.7
U	0.3
T	0.5
k	0.7
ϵ	0.7
R	0.7
Residual control	
p_{rgh}	1.00×10^{-2}
U	1.00×10^{-2}
T	1.00×10^{-2}
k	1.00×10^{-2}
ϵ	1.00×10^{-2}
ω	1.00×10^{-2}

Table 4.4: Configuration of the simulation.

due to their limited significance in this context.

Considering the user experience, the simulation speed is a crucial factor, ranging from milliseconds to days. To optimize this, a liberal convergence criterion is selected, and a parallel processing approach is implemented, utilizing eight out of ten processor cores. Automation scripts control decomposing, simulation, and mesh reconstruction, streamlining the entire process. To further expedite simulations, the

initial simulation time is set to a higher value, achieved by transferring pre-simulated time files at the simulation's start. These files, previously processed with converged meshes, facilitate quicker equation solving during the simulation reducing the simulation time to minutes.

4.2 Augmented Reality Application

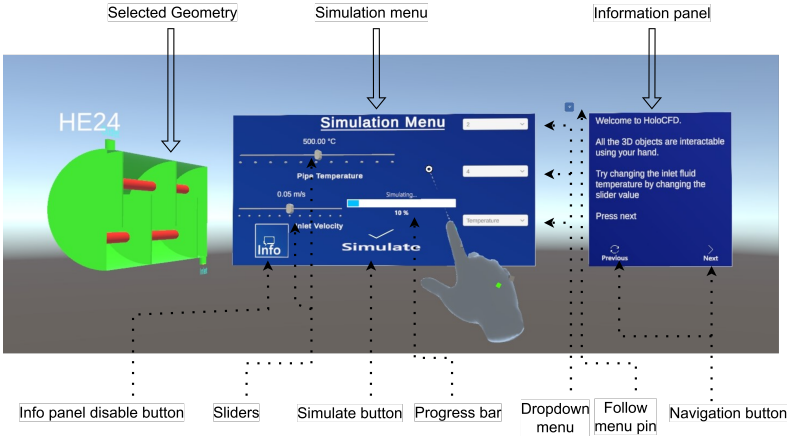

Figure 4.3: Design of the user interface in the AR application.

Developing applications for AR devices, such as the Microsoft HoloLens 2, has distinct issues that need to be considered during the design process. The development process was greatly simplified by utilizing a game engine and a specialized toolkit designed for programming augmented reality, such as the Mixed Reality Toolkit (MRTK). These toolkits are designed to seamlessly connect to the game engine and offer a range of prebuilt tools that facilitate the integration of game design into the AR application. The toolkit

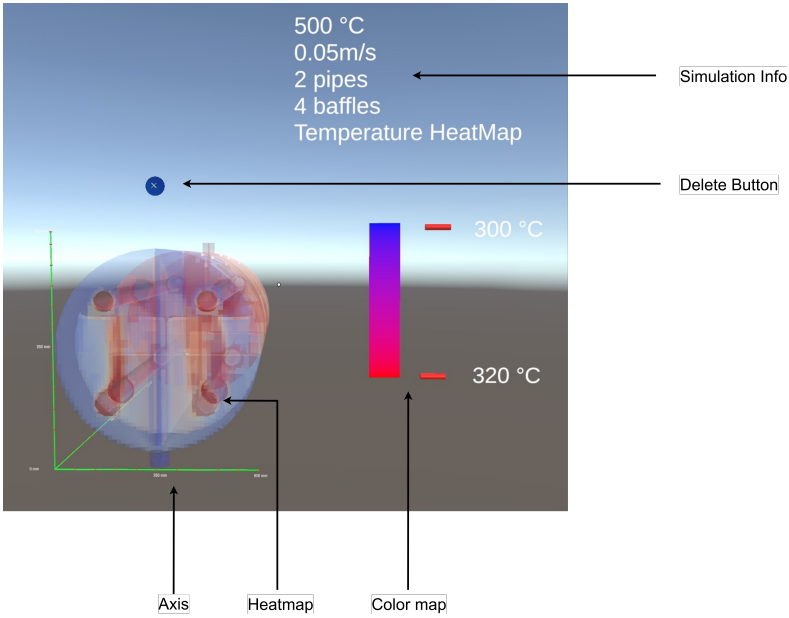

Figure 4.4: Game view inside unity of the CFD temperature heatmap of the heat exchanger

offered preexisting routines for different components, such as hand interactions, menu interactions, etc., but programming was necessary to provide further functionality. Scripts were developed to enhance the efficiency of the user input process and visualization. The ultimate implementation entailed transforming the project into an application that is effortlessly launched into the AR device.

The AR application incorporated canvas elements, buttons, dropdown menus, and sliders to augment the overall user experience. A simulation menu was constructed using these components, which bears a resemblance to the GUI menus commonly found in simulation

software as shown in Figure 4.3. One of the best things about the app is how well it uses real-life human interactions. Interactive 3D models show this by letting users move, rotate, and scale simulation results by touching them.

The model's visual representation was improved by incorporating informative features such as axes, color maps, and simulation parameters as shown in Figure 4.4. Visual clues were integrated to assist users in comprehending the spatial elements of the simulation. The AR application successfully incorporated interactive elements, such as progress indicators, to offer vital feedback on the simulation's advancement. The feedback serves the purpose of informing the user about the completion of the simulation or any faults that may occur throughout the process. The interface design and user interactions of the program were meticulously designed to improve usability and provide an engaging AR experience.

An information panel, providing general user guidance within the application, has been included as shown in Figure 4.3. This panel offers further details regarding the solver and fluid utilized for the simulation. The information panel is equipped with an interactive interface that includes two buttons, allowing users to easily browse different sections of the manual.

The simulation results are stored in the cloud after the data conversion process. The AR application utilizes APIs embedded within scripts to reach this cloud. The results are labelled according to the simulation parameters, including temperature, velocity, and specified geometry. The AR application retrieves this data from the cloud, depending on these factors. The AR application receives feedback from the server PC over MQTT regarding the status of the backend process. When the AR application receives a completed message, it is prompted to download and display the result in the application, together with the required visual elements. A script is implemented to monitor these signals and trigger updates to the progress bar and

ultimate result visualization.

4.3 Server PC

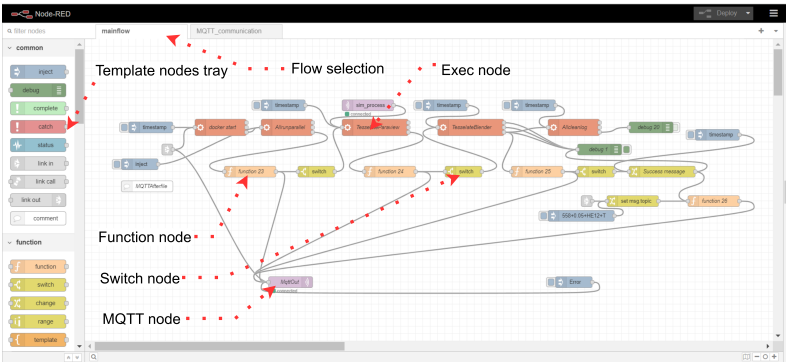

Figure 4.5: Node-red configuration for the automation of server PC.

The automation software is the focal point on the server PC. The implementation was carried out using Node-Red, a programming package that utilizes a visual-based approach. Automation software includes prebuilt template nodes that are utilized to configure the connections between various actions as shown in Figure 4.5. Template nodes are utilized for MQTT communication when there is a requirement to receive or publish mqtt messages. When a message is received at the specified mqtt node, it is then forwarded to the next node for additional processing. Scripting nodes are provided to develop concise scripts for extracting information from MQTT messages and utilizing them to configure the parameters of the CFD simulation. In Node-Red, this may be achieved through the utilization of concise JavaScript code, which is created without extensive programming expertise. Specific nodes are employed to selectively

screen communications or transition to an alternative task node contingent upon the received message. It is necessary to provide feedback messages to the AR application in order to notify the user about the outcome, whether it is an error or success, of a specific task. A modular approach was implemented to establish the transmission of user input to the CFD. This approach is easily scaled up to accommodate changes in parameters in the CFD simulations. Automation software initiates the execution of a script or piece of software by gaining access to the fundamental command terminal of the operating system. Specific nodes facilitate the automation software's ability to access this terminal. The thoroughness of the automation was a significant parameter that the study encountered during the design process. During the design process, several server states are conceptualized, and all potential dead ends or uncontrolled loops are eliminated. The messages transmitted over MQTT are modified according to the simulation settings employed. The scripts in the AR application subsequently read these parameters to present the desired result.

Multiple scripts were employed to achieve automation, with the purpose of alleviating the programming burden on the automation process. The decision of whether to create a script or include it in automation software depends on the nature of the activity at hand. If the task is performed repeatedly in each simulation without any alterations, it is classified as programmed. If the task necessitates extensive manipulation, it is mostly incorporated into the automation program. Multiple scripts are employed on the server for this specific objective. The scripts are composed in the necessary programming language according to the specific application. It is emphasized that these scripts are not same as the javascripts inside the automation program and are implemented outside the automation software.

A script was employed during the preprocessing stage to duplicate mesh files according to the chosen geometry. This script also gets the latest time files presimulated for faster processing. Another script

was utilized for simulation, assuming the responsibilities of automating the simulation and implementing parallel processing.

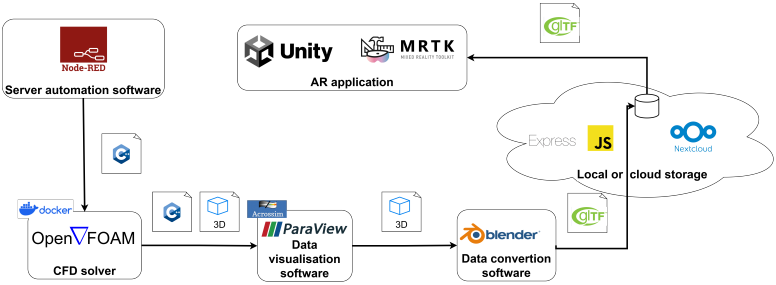

Figure 4.6: Processing of data in various modules of the system.

The automation of postprocessing in data visualization software is achieved by employing python scripts that utilize the APIs of both the data visualization software and the data conversion program inspired from Acrossim. The project provides a python environment and a script that makes use of the blender and paraView APIs. To initiate the execution of this script for this particular study, python script files are first created under the data visualization software. These files include the precise code needed to reproduce the desired data visualization. Subsequently, 3D files are generated using the same software through a script that utilizes the CFD datasets. These files are then imported into data conversion software, where they are transformed into a format that is compatible with the AR application. There are two scripts that involved in the post-processing tasks, One of the scripts is to do data visualization tasks, while the other is specifically designed for data conversion and uploading to the cloud. The processing of data in various modules of the system is depicted in Figure 4.6 This patch resolves compatibility issues that may arise when utilizing more recent versions of post-processing software, guaranteeing smooth integration. Finally, an additional

script is employed to purge all the produced CFD datasets as the essential files have already been transferred to the cloud, thereby avoiding superfluous file removal and conserving memory.

5 Results

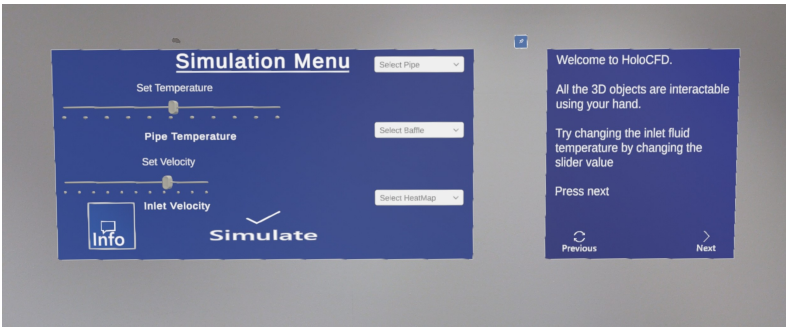

Figure 5.1: Simulation menu at the start of the AR application in Microsoft Hololens 2.

The utilization of Microsoft HoloLens 2 in creating and executing an AR application has resulted in promising results. The simulation menu design, depicted in Figure 5.1, displays a user-friendly interface made with prebuilt functions from the MRTK. The application is equipped with many functions such as buttons, sliders, and drop-down menus, which greatly improve its accessibility. The incorporation of natural hand gestures, as demonstrated in Figure 5.4 for selecting the simulation parameters, enhances the application's intuitive user-friendliness.

The geometry model presented near to the simulation menu gives the user a clear concept of the design of the pipes and baffles in the heat exchanger as seen in Figure 5.5. Panel for showing user

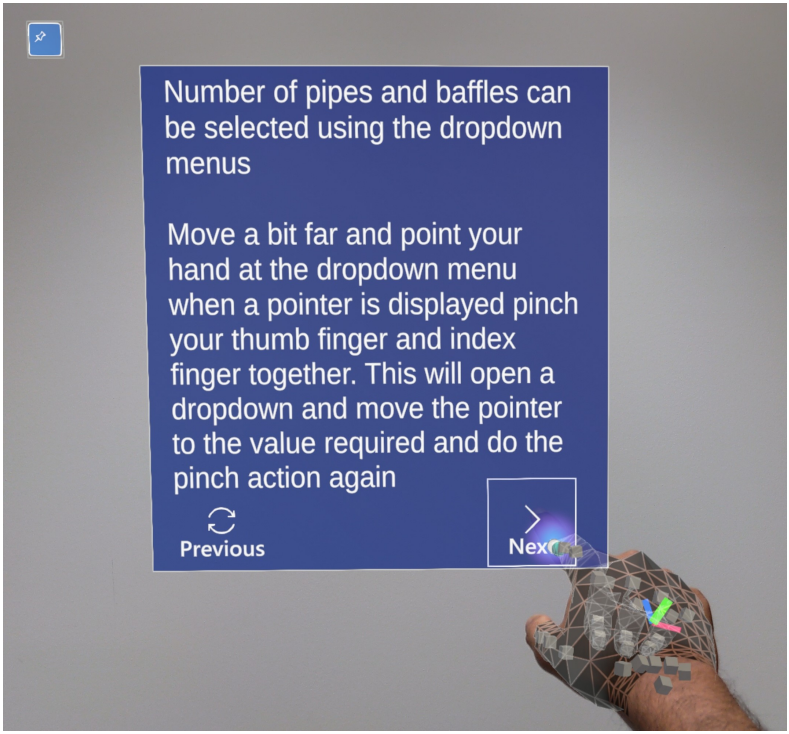

Figure 5.2: The panel for displaying the user guide and simulation information.

instructions makes it easy for a new user to get accommodated to the UI elements in the program as shown in Figure 5.2. The panel also presents data regarding the simulation solvers and the fluid employed. The menu is hidden as needed by utilizing a button located in the lower left corner. Visual cues are smoothly incorporated into the AR application through the utilization of different prebuilt features offered by the MRTK. The MQTT communication from the AR

Figure 5.3: Dropdown menu for selecting the parameters for the geometry in the AR application.

application is exceptionally fast. The straightforward menu design enables accessibility even for non-expert users, removing the need for substantial prior knowledge.

The backend processes on the server PC demonstrate remarkable efficiency, completing in less than 5 minutes to process and generate visualizations. Within this timeframe, the simulation constitutes the majority, taking approximately 2-3 minutes, while subsequent blender and paraView operations each consume less than 1 minute. Preprocessing tasks exhibit minimal time requirements. These noteworthy processing speeds underscore the proof of concept for developing reliable applications using this architecture with minimal coding and configuration efforts. Results are swiftly accessible in the cloud, typically below 20 MB in size, ensuring quick transmission and remote accessibility, contingent upon internet speed.

The simulation menu, illustrated in Figure 5.6, incorporates a

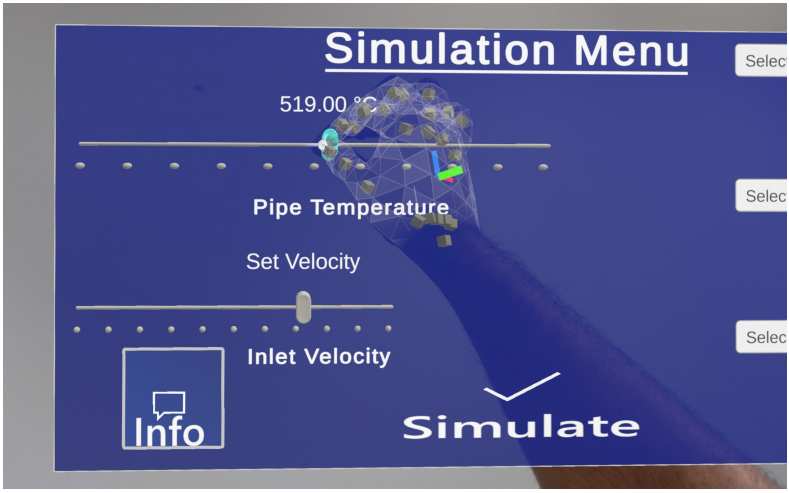

Figure 5.4: Slider hand UI interaction to set the temperature of the simulation.

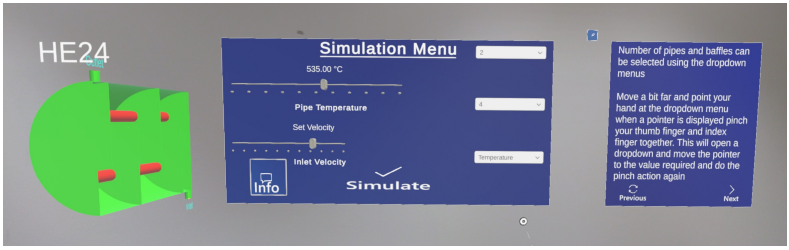

Figure 5.5: Simulation menu of the AR application with the model of the geometry.

progress bar that dynamically updates based on MQTT communication from the server PC. This serves as user feedback, enabling multitasking or reviewing previous results while the simulation is

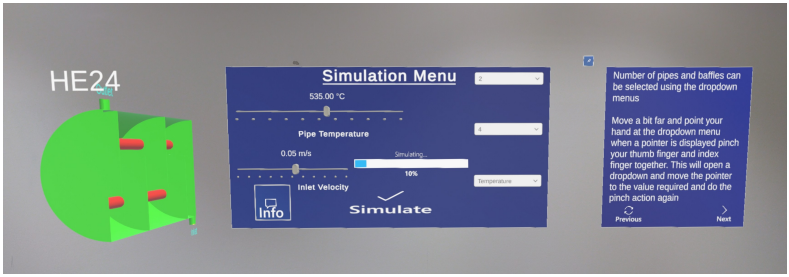

Figure 5.6: Progress bar displaying the updates in the server PC.

underway. Upon completion, the progress bar disappears, and the extracted visualization model loads into the AR application, as depicted in Figure 5.7. Additional visual cues, such as axis, colormap, and simulation parameters, are displayed above the model for easy reference in Figure5.7. Users interact with the model using one or both hands to move, rotate, or scale as desired. Concurrently, temperature and velocity maps are generated, allowing users to swiftly switch between simulation results with minimal delay, enhancing the accessibility and exploration of the data. The velocity map, shown in Figure 5.10, provides an additional layer of insight, allowing users to study heatmaps side by side with interactions and gain a deeper understanding of the simulation results as seen in Figure 5.12. Intuitive interactions, such as moving around and zooming in, contribute to a more immersive user experience. The software accommodates a button for closing simulation results as shown in Figure 5.7, ensuring a clutter-free environment. Notably, the CFD results remain viewable even in highly-lit conditions as shown in Figure temp model. At low-light conditions CFD simulations are much more visible and clear as shown in Figure 5.8. This innovative approach broadens possibilities for engineers, enabling them to transcend the constraints of 2D screens and explore designs in a more immersive and engaging manner.

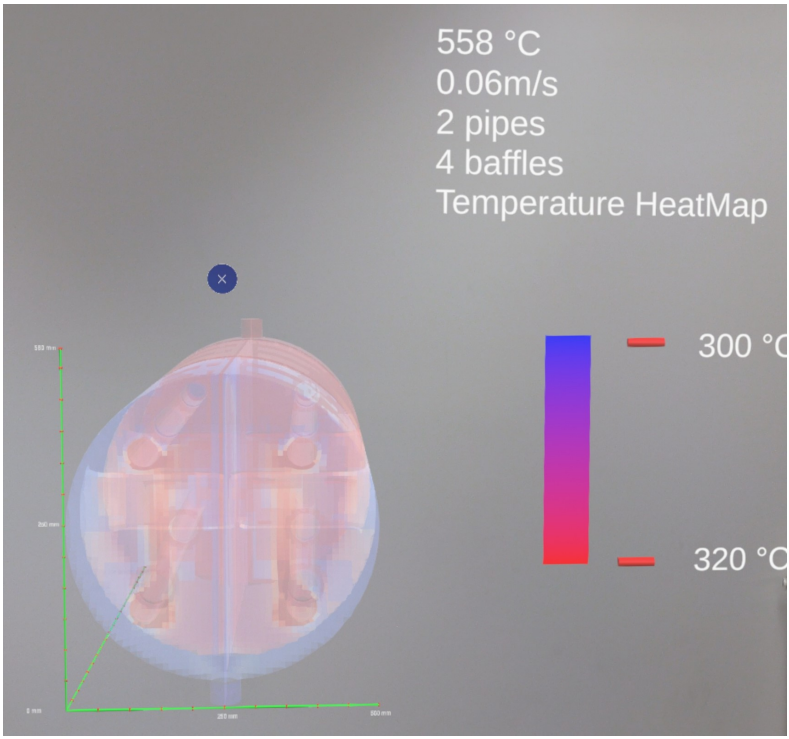

Figure 5.7: Temperature heatmap of a CFD simulation in the AR application.

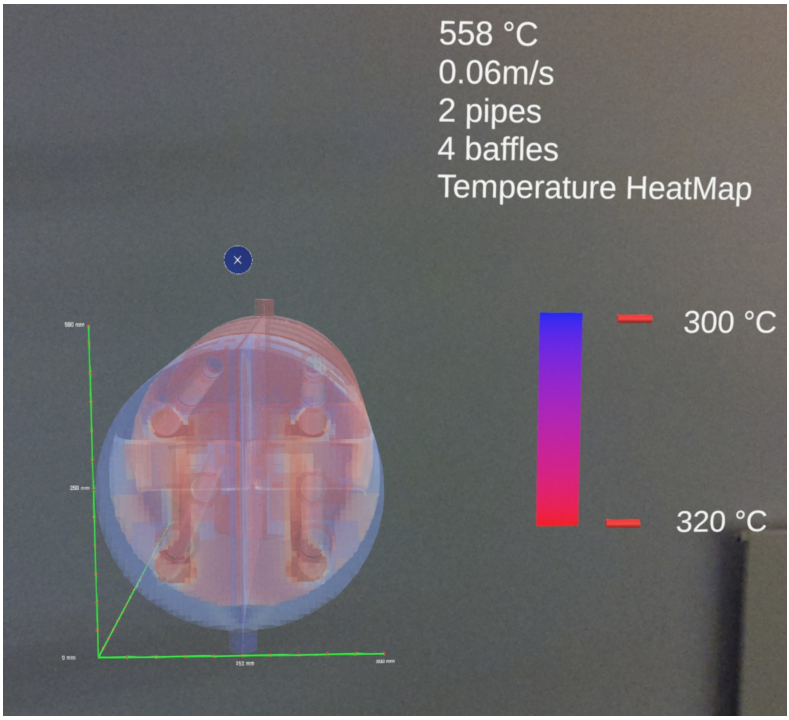

Figure 5.8: View of temperature heatmap at minimal lighting environment.

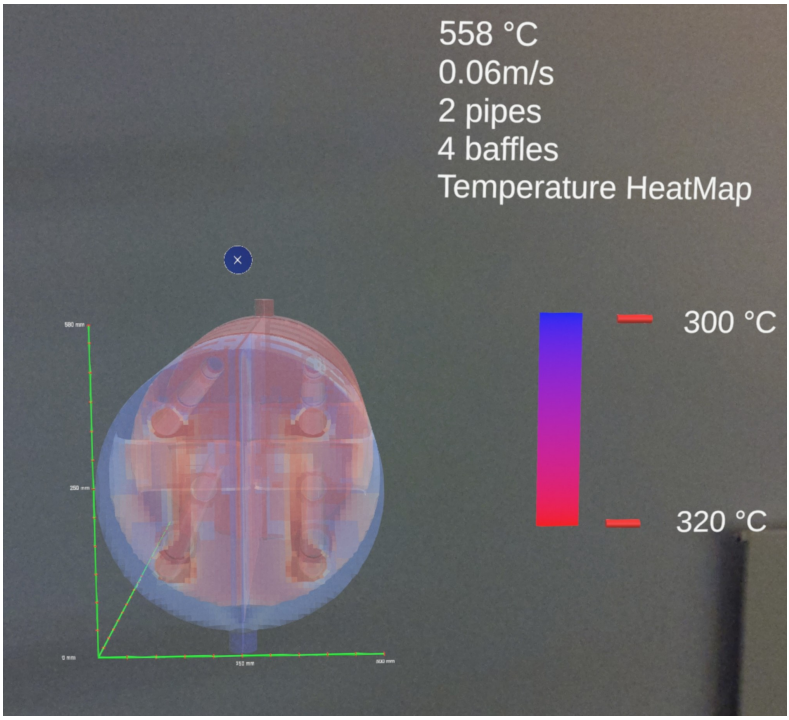

Figure 5.9: Sideview of temperature heatmap.

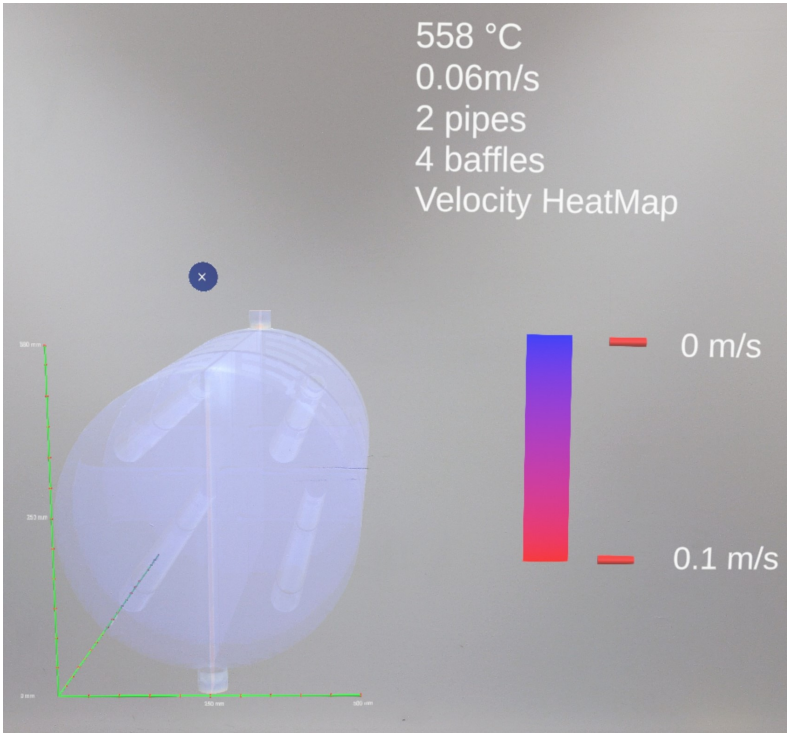

Figure 5.10: Velocity heatmap of a CFD simulation in the AR application.

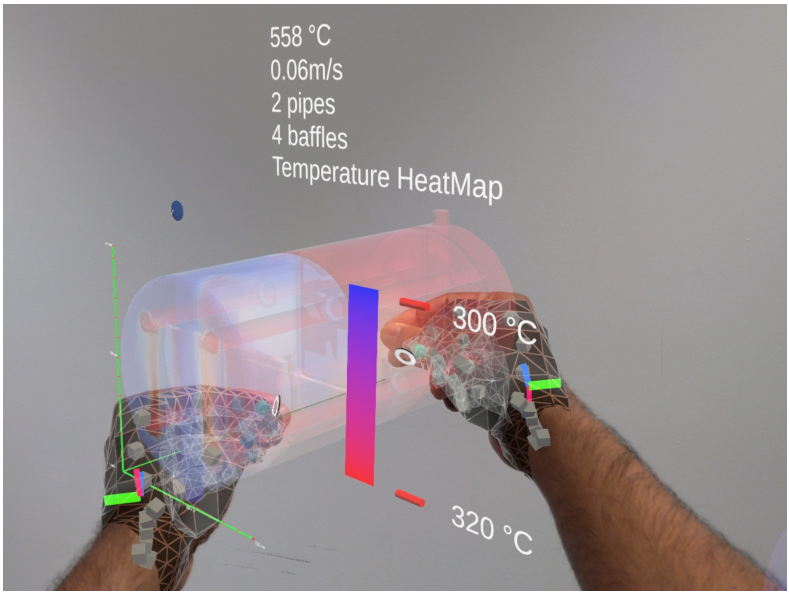

Figure 5.11: Two hand interaction of the CFD temperature heatmap for scaling, rotation and moving.

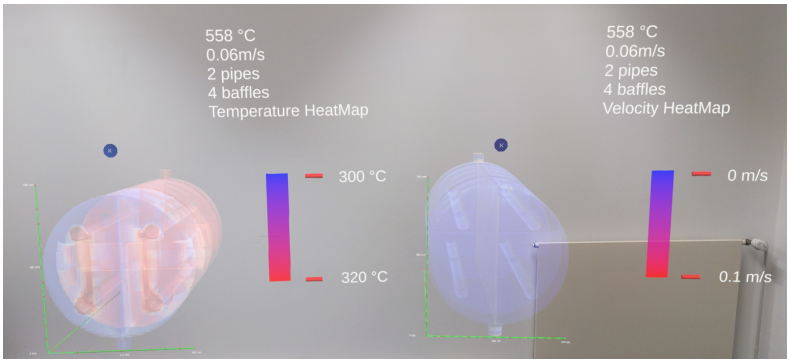

Figure 5.12: View of multiple heatmaps side by side inside the AR application.

6 Limitations And Future Developments

While the current system is functional, it is not without its limitations. Presently, the system operates exclusively with predefined meshes, necessitating users to manually create and insert new meshes into the openFOAM simulation folder if additional variations are desired. This constraint restricts users to the predefined set of meshes integrated into the application. Moreover, the configuration of all software components on the server PC is currently performed manually, posing challenges for system replication and consistency across different server setups.

The visualization capabilities of the system are currently limited to temperature and velocity maps. However, for practical utility, a more comprehensive set of post-processing visualizations is required. The current approach involves creating state files for each specific post-processing requirement, a method that is cumbersome and lacks a dedicated menu for configuring post-processing parameters akin to the simulation parameters.

Additionally, the computational demands for visualizing .glb files in the Hololens prove taxing, resulting in momentary blackouts during model loading. Although brief, these interruptions affect the overall usability of the application. The Hololens 2's computational limitations further exacerbate this issue, as it is not the most potent AR headset available in the market.

Furthermore, the simulation menu presently offers a limited set of

configurable options. While this is expanded by introducing more MQTT topics and file operations in Node-RED, the management of numerous file operations and MQTT topics may become unwieldy. An alternative solution is sought to handle applications with an extensive array of possible parameter changes more efficiently.

The existing system is constrained by the use of a single solver, making it challenging to simulate scenarios involving chemical reactions or transition between laminar and turbulent flow based on Reynolds numbers. The limitations extend to the maximum values permissible for temperature and velocity, restricting the range of simulations that is effectively performed.

Finally, the current simulation settings prioritize quick convergence for reduced user waiting time. However, if higher-resolution and more accurate CFD results are imperative, the existing tolerance levels may lead to prolonged simulation durations. This delineates the current limitations inherent in the application.

The adoption of AR applications for CFD simulations faces obstacles beyond hardware costs, extending to the financial considerations associated with software development. The development and maintenance of such sophisticated applications demand skilled software developers who need to be compensated adequately. These costs, coupled with the expenses related to the acquisition of high-end PCs and AR devices like the HoloLens 2, collectively contribute to the overall financial barriers. Small-scale enterprises, educational institutions, or individual researchers may find these financial commitments challenging, hindering the widespread integration of AR-based CFD simulations into various domains. Striking a balance between technological advancements, cost-effectiveness, and developer compensation will be crucial to making these immersive applications more accessible to a broader user base in the future.

Future research endeavours are suggested to focus on developing

a sophisticated user interface comparable to traditional CFD software GUIs, aiming to enhance the user experience and streamline interactions with the AR application. Implementing a system that dynamically downloads example cases based on user-defined parameters, allowing the selection of flow models according to velocity and other parameters, is recommended to significantly enhance the adaptability and user-friendliness of the application.

Consideration is to be given to integrating an online geometric modelling tool within the AR application, leveraging the extensive capabilities of 3D interfaces. Existing modeling applications in augmented reality can serve as a reference, and exploration of plugins like PiXYZ or other Unity extensions, as suggested in previous studies, endorse the way for seamless geometric modeling within the application.

A potential improvement is incorporating a dedicated post-processing menu, separate from the simulation menu. Direct linking of the Paraview API to the application is recommended to eliminate the need for creating state files for each post-processing scenario. This approach might open avenues for animated and interactive post-processing, providing users with more detailed insights by zooming in on specific areas of interest.

Another avenue for future exploration is the integration of Artificial Intelligence (AI) into the system. This suggestion leads to real-time simulations or, at the very least, a reduction in simulation time, thereby improving the accuracy and efficiency of analyses.

To simplify deployment, streamlining the installation process by employing a Docker container with a YAML file for server system setup is recommended. Additionally, incorporating a database management system, particularly within an Amazon Web Services infrastructure, is suggested to facilitate efficient analysis of numerous results and enable rapid responses to user inputs. Exploring fully cloud-based computing services is recommended to potentially elimi-

nate the need for a dedicated server computer, enhancing the overall system efficiency.

7 Conclusion

As a result of this research, a reliable client-server architecture at the confluence of CFD and AR has been developed. A novel platform has been created by combining microsoft holoLens 2 and open-source CFD tools such as openFOAM, blender, and paraView. This client-server framework, which is powered by automation tools such as node-red, docker containers, and nextcloud, lays the groundwork for an immersive and approachable augmented reality experience in fluid dynamics simulations.

The usability of the AR application has been emphasized, with the HoloLens 2 client-side interface providing an intuitive and interactive experience. Users modify simulation settings using natural hand motions, allowing them to gain a better understanding of complex fluid dynamics. The application's design, which incorporates interactive components and uses the MRTK, provides user-friendliness and instructional value.

While the built system demonstrates noteworthy efficiency in terms of speedy simulation timeframes, real-time user input, and cloud-based accessibility, its limits must be acknowledged. Manual mesh management, Hololens 2 computational limits, and the requirement for a more broad range of display options are all problems that highlight the developing character of this interdisciplinary area.

Future prospects include addressing these restrictions and enhancing the system. The client-server architecture is suggested to be automated further to simplify simulation setup and deployment. To give a more immersive experience, hardware capabilities, particularly

those of AR devices, might be enhanced. The creation of a broader set of post-processing and visualization options will increase the system's usefulness across many simulation settings.

As the study is wrapped up, it is clear that the engineering domains will be significantly impacted by the synergistic combination of AR and CFD, particularly in applications such as heat exchanger simulations. This thesis lays the groundwork for revolutionary advancements in simulation-driven engineering, while simultaneously furthering understanding of fluid dynamics through immersive visualization and interactive capabilities. This work lays the groundwork for further investigation, creativity, and improvement in the dynamic junction of AR and CFD.

Bibliography

[Bha22] Bhatia, N. and Matar, O. K. (2022): *Learning and Teaching Fluid Dynamics using Augmented and Mixed Reality. 2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*.

[Bor08] Bordegoni, M., Ferrise, F., Ambrogio, M., Caruso, G., Bruno, F., and Caruso, F. (2008): *Environment based on augmented reality and interactive simulation for product design review. 6th Eurographics Italian Chapter Conference 2008 - Proceedings*, 27–34. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/027-034.

[Dig20] (2020). *Digital Education Action Plan (2021-2027)*. https://education.ec.europa.eu/focus-topics/digital-education/action-plan.

[FB18] Fast-Berglund, S., Gong, L., and Li, D. (2018): *Testing and validating Extended Reality (xR) technologies in manufacturing. Procedia Manufacturing*, 25:31–38.

[Fle22] Fletcher, D. F. (2022): *The future of computational fluid dynamics (cfd) simulation in the chemical process industries. Chemical Engineering Research and Design*, 187:299–305. ISSN 0263-8762. doi:https://doi.org/10.1016/j.cherd.2022.09.021.

[Fra11] Franzen, D., Avellino, I., and Mauri, F. (2011): *Magic wako-user interaction in a projector-based augmented reality game.*

[Gru23] Gruber, B. (2023). *Augmented Reality: Das ist neu in ARKit 2.* https://www.macwelt.de/article/974500/augmented-reality-das-ist-neu-in-arkit-2.html.

[Gun21] Gunawan, P., Kwan, J., Cai, Y., and Yang, R. (2021): *Augmented Reality Application for Chemical Engineering Unit Operations. Virtual and Augmented Reality, Simulation and Serious Games for Education*, 29–43.

[Kim18] Kim, M., Yi, S., Jung, D., Park, S., and Seo, D. (2018): *Augmented-Reality Visualization of Aerodynamics Simulation in Sustainable Cloud Computing. Sustainability*, 10(5):1362.

[Kim21] Kim, M., Seo, D., Park, S., and Chan, J. S. (2021): *Visualization of simulation results based on mobile augmented reality.* 12(09):877–882. ISSN 21852766. doi: 10.24507/icicelb.12.09.877.

[Kir22] Kirschstein, A. (2022). *Top 10 AR Headsets 2022 und ein Blick in die Zukunft.* https://www.paradys.net/post/top-10-ar-headsets-2022-und-ein-blick-in-die-zukunft.

[Kum21] Kumar, V. V., Carberry, D., Beenfeldt, C., Andersson, M. P., Mansouri, S. S., and Gallucci, F. (2021): *Virtual reality in chemical and biochemical engineering education and training. Education for Chemical Engineers*, 36:143–153.

[Li17] Li, W., Nee, A., and Ong, S. (2017): *A State-of-the-Art Review of Augmented Reality in Engineering Analysis and Simulation. Multimodal Technologies and Interaction*, 1(3):17.

[Mal05] Malkawi, A. M. and Srinivasan, R. S. (2005): *A new paradigm for Human-Building Interaction: the use of CFD and Augmented Reality. Automation in Construction*, 14(1):71–84.

[Mor13] Moreland, J. R., Wang, J., Liu, Y., Li, F., Shen, L., Wu, B., and Zhou, C. Q. (2013): *Integration of augmented reality with computational fluid dynamics for power plant training.*

[Mor21] Morosi, F., Caruso, G., and Cascini, G. (2021): *Spatial Augmented Reality as a Visualization Support for Engineering Analysis. Lecture Notes in Mechanical Engineering*, 103–115.

[Mou22] Mourtzis, D., Angelopoulos, J., and Panopoulos, N. (2022): *Integration of mixed reality to cfd in industry 4.0: A manufacturing design paradigm. Procedia CIRP*, 107:1144–1149. ISSN 2212-8271. doi:https://doi.org/10.1016/j.procir.2022.05.122. Leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022.

[Pal16] Pal, E., Kumar, I., Joshi, J. B., and Maheshwari, N. (2016): *Cfd simulations of shell-side flow in a shell-and-tube type heat exchanger with and without baffles. Chemical Engineering Science*, 143:314–340. ISSN 0009-2509. doi:https://doi.org/10.1016/j.ces.2016.01.011.

[Sch21] Schweiß, T., Nagaraj, D., Bender, S., and Werth, D. (2021): *Towards Collaborative Analysis of Computational Fluid Dynamics using Mixed Reality. Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.*

[Sil21] Silvestri, L. (2021): *Cfd modeling in Industry 4.0: New perspectives for smart factories. Procedia Computer Science*, 180:381–387.

[Sol21] Solmaz, S. and Van Gerven, T. (2021): *Automated integration of extract-based CFD results with AR/VR in engineering education for practitioners. Multimedia Tools and Applications*, 81(11):14869–14891.

[Sol22] Solmaz, S. and Van Gerven, T. (2022): *Interactive CFD simulations with virtual reality to support learning in mixing. Computers & Chemical Engineering*, 156:107570.

[Sol23a] Solmaz, S., Kester, L., and Van Gerven, T. (2023): *An immersive virtual reality learning environment with CFD simulations: Unveiling the Virtual Garage concept. Education and Information Technologies.*

[Sol23b] Solmaz, S. and Van Gerven, T. (2023): *Acrossim: A toolkit for cross-platform integration of cfd simulation data in computer graphics. SoftwareX*, 24:101585. ISSN 2352-7110. doi:https://doi.org/10.1016/j.softx.2023.101585.

[Teu22] Teutscher, D., Weckerle, T., Öz, M. F., and Krause, M. J. (2022): *Interactive Scientific Visualization of Fluid Flow Simulation Data Using AR Technology-Open-Source Library OpenVisFlow. Multimodal Technologies and Interaction*, 6(9):81.

Appendix

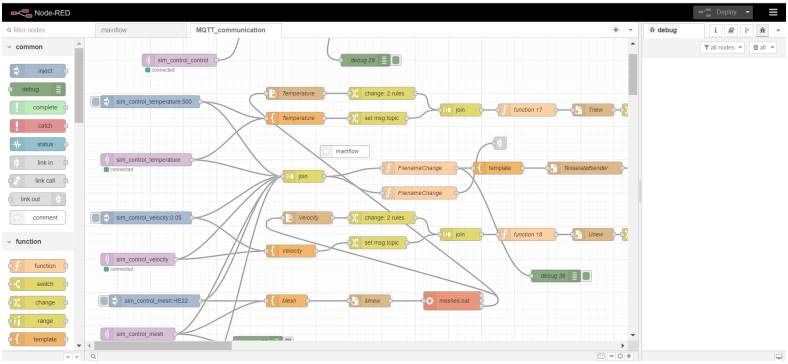

Figure 1: Node red GUI representing the user input flow of the automation.

```
#!/bin/bash

# Define the output file
output_file="HE12.stl"

# Check if output file already exists and remove it to
    avoid appending
if [ -f "$output_file" ]; then
    rm "$output_file"
fi

# Find all STL files in the current directory and
    concatenate them
for stl in *.stl
```

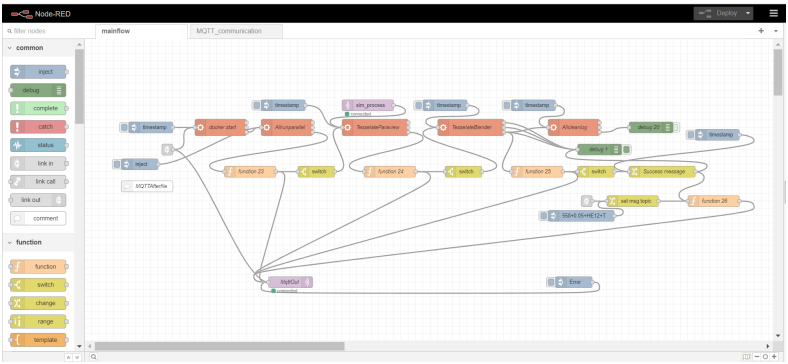

Figure 2: Node red GUI representing the main flow of the automation.

```
do
  # Check if the file is not the output file to avoid
    self-appending
  if [ "$stl" != "$output_file" ]; then
    cat "$stl" >> "$output_file"
  fi
done

echo "All STL files have been combined into $output_file"
```

Listing 1: combinestl

```
@echo off
SETLOCAL EnableDelayedExpansion

REM The name of the source folder inside the "meshes"
    directory
SET "sourceFolderName=HE24"

REM The path to the "meshes" directory
SET "meshesPath=%~dp0meshes"

REM The path to the "constant" directory
SET "constantPath=%~dp0constant"

```

```
REM Full path to the source folder
SET "fullSourcePath=!meshesPath!\!sourceFolderName!"

REM Full path to the destination "polyMesh" folder
SET "polyMeshPath=!constantPath!\polyMesh"

REM Path to the "300" folder inside the "latestTime" directory
SET "latestTimePath=%~dp0latestTime\!sourceFolderName!\300"

REM Path to the "300" folder in the script's directory
SET "destination300Path=%~dp0300"

REM Check if the source folder exists
IF NOT EXIST "!fullSourcePath!" (
    echo Source folder "!fullSourcePath!" does not exist.
    exit /b
)

REM Remove the existing "polyMesh" folder if it exists
IF EXIST "!polyMeshPath!" (
    echo Removing existing polyMesh folder...
    RMDIR /S /Q "!polyMeshPath!"
)

REM Copy the contents from the source folder to the "polyMesh" folder
echo Copying from "!fullSourcePath!" to "!polyMeshPath!"...
XCOPY /E /I "!fullSourcePath!" "!polyMeshPath!"

REM Check if the "300" folder exists in the "latestTime" directory
IF NOT EXIST "!latestTimePath!" (
    echo "300" folder "!latestTimePath!" does not exist.
    exit /b
)

REM Remove the existing "300" folder in the script's directory if it exists
IF EXIST "!destination300Path!" (
    echo Removing existing "300" folder in the script's
```

```
    directory...
    RMDIR /S /Q "!destination300Path!"
)

REM Copy the "300" folder from the "latestTime" directory to the script's directory
echo Copying from "!latestTimePath!" to "!destination300Path!"...
XCOPY /E /I "!latestTimePath!" "!destination300Path!"

echo Done.
ENDLOCAL
```

Listing 2: meshes.bat

```
#!/bin/sh

WM_PROJECT_DIR=/usr/lib/openfoam/openfoam2212  # Set the WM_PROJECT_DIR

cd "${0%/*}" || exit                                # Run from this directory
. ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions        # Tutorial run functions
#     ------------------------------------------------------------------------


restore0Dir

# Extract the last part of the current directory as the folder name
foldername=$(basename "$PWD")

# Generate the .foam filename based on the folder name
foamfilename="${foldername}.foam"

# Use the touch command to create the .foam file
touch "$foamfilename"

# Set the number of processors for parallel processing
NUM_PROCS=8  # Adjust the number of processors as needed

# Decompose the case for parallel processing
```

```
decomposePar -force

# Run the parallel MPI version of the OpenFOAM application
mpirun --allow-run-as-root -np $NUM_PROCS $(getApplication) -parallel

# Reconstruct the case after the parallel run
reconstructPar -latestTime

# Note: $(getApplication) is used to run the application specific to your simulation.
#     ------------------------------------------------------------------------
```

Listing 3: Allrunparallel

```
#!/bin/sh
cd "${0%/*}" || exit                                # Run from this directory
WM_PROJECT_DIR=/usr/lib/openfoam/openfoam2212  # Set the WM_PROJECT_DIR
. ${WM_PROJECT_DIR:?}/bin/tools/CleanFunctions      # Tutorial clean functions
#     ------------------------------------------------------------------------

cleanAuxiliary

# Function to delete folders inside the main directory, excluding specified ones
deleteFolders() {
    for dir in */; do
        dirname=$(basename "$dir")
        if [ "$dirname" != "0" ] && [ "$dirname" != "300" ] && [ "$dirname" != "system" ] && [ "$dirname" != "constant" ] && [ "$dirname" != "meshes" ]&& [ "$dirname" != "latestTime" ]; then
            echo "Deleting directory: $dir"
            if rm -rf "$dir"; then
                echo "Directory $dir deleted successfully."
            else
```

```
                echo "Failed to delete directory: $dir"
            fi
        fi
    done
}
# Call the function to delete folders
deleteFolders
#
    ------------------------------------------------------------------
```

Listing 4: Allclean

```
import bpy
import os
from distutils.dir_util import copy_tree
import timeit
import glob
import shutil

#start2 = timeit.default_timer()
#data format to export visual CFD data from ParaView: x3d
    , vrml, svg and other supported formats
#(optional) exportParaview = input ('Enter the export
    format from ParaView: ')
exportParaview = 'x3d'
export_format_paraview = '.' + exportParaview

import_format_blender = export_format_paraview
#data format to export visual CFD data from Blender: fbx,
     obj, 3ds, ply, stl and other supported formats

exportBlender = 'glb'  # Define the export format for
    Blender
export_format_blender = '.' + exportBlender
#create a directory to collect processed data and
    metadata
path_metadata = 'process/metadata/'
os.makedirs(path_metadata, exist_ok=True)
path_paraview = os.getcwd() + '/' + path_metadata
path_blender = path_paraview
path_unityfor = os.getcwd() + '/' + 'process/'
# Loop through all objects in the scene
#blender starts metadata import & export
```

```
path_to_obj_dir = os.path.join(path_blender)
#get list of all files in directory
file_list = sorted(os.listdir(path_to_obj_dir))
#get a list of files ending in 'obj' in Blender
obj_list = [item for item in file_list if item.endswith(
    import_format_blender)]
#loop through the strings in obj_list and add the files
    to the scene
for item in obj_list:
    path_to_file = os.path.join(path_to_obj_dir, item)
    bpy.ops.import_scene.x3d(filepath = path_to_file)
    #get the current path and make a new folder for the
    exported meshes
    path_blender = bpy.path.abspath(path_blender)
    # Get a reference to the current active scene
    scene = bpy.context.scene
    for obj in scene.objects:
        bpy.ops.object.select_all(action='DESELECT') #to
    deselect all meshes
        # Select the current object

        # Check if the object name starts with "
    Shape_IndexedFaceSet"
        #if obj.name.startswith("Shape_IndexedFaceSet"):
        if obj.name != "Cube":
            # Select the object
            obj.select_set(True)
            print(obj.name)


    bpy.ops.export_scene.gltf(filepath=path_blender + '
    993+0.05+HE24+T' + export_format_blender,
    export_format='GLB', use_selection=True,
    export_tangents=True, export_attributes=True,)
    print("Done")

"""
Data processing analytics for qualitative studies (3)
"""

path_to_obj_dir = os.path.join(path_blender)
file_list = sorted(os.listdir(path_to_obj_dir))
obj_list = [item for item in file_list if item.endswith('
```

```
    .glb')]
size_file_blender = []
for item in obj_list:
    path_to_file = os.path.join(path_to_obj_dir, item)
    size_processing_blender=os.path.getsize(path_to_file)
    print('GLB file sizes in bytes:',
    size_processing_blender)
    size_file_blender.append(size_processing_blender)
    shutil.copy(path_to_file, os.getcwd() + '/' + '
    ExpressServer/ABs/')
    cloud_path = r"Z:\TestCFD"
    shutil.copy(path_to_file, cloud_path)

print('Data processing has successfully been completed!')

#clean up metadata
#metadata = input ('Clean all metadata? [y or n]: ')
metadata = "y"
mypath = path_paraview
if metadata == 'y':
    for mydata in glob.glob(mypath + "Shape*"):
        shutil.copy(mydata, os.getcwd() + '/' + 'process/
    ')
    for metadata in glob.glob(mypath + "*"):
        os.remove(metadata)
    print ('Metadata is cleaned.')
else:
    print ('Metadata is available under the process
    directory.')
#self.text_comments.insert('end', open(filename,'r').read
    ())



"""
The end
"""
```

Listing 5: Tesselate Blender

```
import os
import sys
import timeit
```

```
import glob
import shutil
from distutils.dir_util import copy_tree
from paraview.simple import *

#start2 = timeit.default_timer()
#data format to export visual CFD data from ParaView: x3d, vrml, svg and other supported formats
#(optional) exportParaview = input ('Enter the export format from ParaView: ')
exportParaview = 'x3d'
export_format_paraview = '.' + exportParaview

import_format_blender = export_format_paraview
#data format to export visual CFD data from Blender: fbx, obj, 3ds, ply, stl and other supported formats

exportBlender = 'glb'  # Define the export format for Blender
export_format_blender = '.' + exportBlender
#create a directory to collect processed data and metadata
statefile = "T.py"
path_metadata = 'process/metadata/'
os.makedirs(path_metadata, exist_ok=True)
path_paraview = os.getcwd() + '/' + path_metadata
path_blender = path_paraview
path_unityfor = os.getcwd() + '/' + 'process/'

# Logging of operations from terminal and save in a text file
class Tee(object):
    def __init__(self, *files):
        self.files = files

    def write(self, obj):
        for f in self.files:
            f.write(obj)
            f.flush()  # If you want the output to be visible immediately

    def flush(self):
        for f in self.files:
```

```
            f.flush()

f = open('logfile_tessellate.txt', 'w')
original = sys.stdout
sys.stdout = Tee(sys.stdout, f)
print("Data processing is in progress...")

# ParaView state file operation
# Timer for each section - data processing speed
start1 = timeit.default_timer()
# Import user-defined Paraview state
exec(open(statefile).read())

# Update the view to ensure updated data information
renderView1 = GetActiveViewOrCreate('RenderView')
renderView1.Update()
stop1 = timeit.default_timer()

# Data processing via ParaView pipeline
start2 = timeit.default_timer()
# Create a directory to collect processed data and
    metadata
os.makedirs(path_paraview, exist_ok=True)

# Define the total number of timesteps (1 for steady-
    state solutions)
timestep_sim = 2

# Obtain a list of timesteps with values
animationScene1 = GetAnimationScene()
tsteps = animationScene1.TimeKeeper.TimestepValues
# Check if tsteps is a single float value or a list/array
if isinstance(tsteps, float):
    # If it's a float, make it a list with one element
    tsteps = [tsteps]

# Now you can safely access the last element
animationScene1.AnimationTime = tsteps[-1]
animationScene1.AnimationTime = tsteps[-1]
# ParaView metadata export
ExportView(path_paraview + str(tsteps[-1]) +
    export_format_paraview, view=renderView1,
    ExportColorLegends=1)
```

```
79
80
81 # for x in range(0, timestep_sim):
82 #     # ParaView metadata export
83 #     ExportView(path_paraview + str(x) +
       export_format_paraview, view=renderView1,
       ExportColorLegends=1)
84 #     # Check out the data processed
85 #     SaveScreenshot(path_paraview + str(x) + '.png',
       renderView1, ImageResolution=[1025, 782])
86 #     # Switch to the next timestep
87 #     animationScene1 = GetAnimationScene()
88 #     animationScene1.AnimationTime = tsteps[-1]
89 stop2 = timeit.default_timer()
90
91 """
92 Data processing analytics for qualitative studies (3)
93 """
94
95 #b_x3d = os.path.getsize(path_blender + tsteps[-1]+'.x3d
       ')
96 print('*****Data processing performance & analytics*****'
       )
97 print('Time_ParaView: ', stop1 - start1)
98 #print('X3D file size in bytes:',b_x3d)
99 print('Time_ParaView_Blender: ', stop2 - start2)
100 print('Time_Integration (sec): ', stop2 - start2 + stop1
       - start1)
```

Listing 6: Tesselate Paraview

```
1 # state file generated using paraview version 5.8.0
2
3 #
      ------------------------------------------------------------------
4 # setup views used in the visualization
5 #
      ------------------------------------------------------------------
6
7 # trace generated using paraview version 5.8.0
8 #
```

```
# To ensure correct image size when batch processing, please search
# for and uncomment the line `# renderView*.ViewSize = [*,*]`

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# get the material library
materialLibrary1 = GetMaterialLibrary()

# Create a new 'Render View'
renderView1 = CreateView('RenderView')
renderView1.ViewSize = [1336, 676]
renderView1.AxesGrid = 'GridAxes3DActor'
renderView1.OrientationAxesVisibility = 0
renderView1.CenterOfRotation = [-8.717179298400879e-07, 0.0, -0.49999908248719294]
renderView1.StereoType = 'Crystal Eyes'
renderView1.CameraPosition = [-8.717179298400879e-07, -2.433399528358085, -0.49999908248719294]
renderView1.CameraFocalPoint = [-8.717179298400879e-07, 0.0, -0.49999908248719294]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraFocalDisk = 1.0
renderView1.CameraParallelScale = 0.6298101422825638
renderView1.BackEnd = 'OSPRay raycaster'
renderView1.OSPRayMaterialLibrary = materialLibrary1

SetActiveView(None)

# ----------------------------------------------------------------

# setup view layouts
# ----------------------------------------------------------------


# create new layout object 'Layout #1'
layout1 = CreateLayout(name='Layout #1')
```

```
43 layout1.AssignView(0, renderView1)
44
45 #
       ------------------------------------------------------------------

46 # restore active view
47 SetActiveView(renderView1)
48 #
       ------------------------------------------------------------------


49
50 #
       ------------------------------------------------------------------

51 # setup the data processing pipelines
52 #
       ------------------------------------------------------------------


53
54 # create a new 'OpenFOAMReader'
55 jeswin_test1foam = OpenFOAMReader(FileName='C:\\Users\\XR
       -Lab\\Documents\\Jeswin Files\\Project HoloCFD\\
       openfoamdocker\\jeswin_test1\\jeswin_test1.foam')
56 jeswin_test1foam.MeshRegions = ['internalMesh']
57 jeswin_test1foam.CellArrays = ['T', 'U', 'alphat', 'p', '
       p_rgh']
58
59 # create a new 'Reflect'
60 reflect1 = Reflect(Input=jeswin_test1foam)
61
62 #
       ------------------------------------------------------------------

63 # setup the visualization in view 'renderView1'
64 #
       ------------------------------------------------------------------


65
66 # show data from reflect1
67 reflect1Display = Show(reflect1, renderView1, '
       UnstructuredGridRepresentation')
68
69 # get color transfer function/color map for 'T'
```

```
tLUT = GetColorTransferFunction('T')
tLUT.AutomaticRescaleRangeMode = 'Never'
tLUT.RGBPoints = [300.0, 0.231373, 0.298039, 0.752941,
    310.0, 0.865003, 0.865003, 0.865003, 320.0, 0.705882,
     0.0156863, 0.14902]
tLUT.ScalarRangeInitialized = 1.0

# get opacity transfer function/opacity map for 'T'
tPWF = GetOpacityTransferFunction('T')
tPWF.Points = [300.0, 0.0, 0.5, 0.0, 320.0, 1.0, 0.5,
    0.0]
tPWF.ScalarRangeInitialized = 1

# trace defaults for the display properties.
reflect1Display.Representation = 'Surface'
reflect1Display.ColorArrayName = ['CELLS', 'T']
reflect1Display.LookupTable = tLUT
reflect1Display.Opacity = 0.38
reflect1Display.OSPRayScaleArray = 'p'
reflect1Display.OSPRayScaleFunction = 'PiecewiseFunction'
reflect1Display.SelectOrientationVectors = 'U'
reflect1Display.ScaleFactor = 0.10001217595709022
reflect1Display.SelectScaleArray = 'p'
reflect1Display.GlyphType = 'Arrow'
reflect1Display.GlyphTableIndexArray = 'p'
reflect1Display.GaussianRadius = 0.005000608797854511
reflect1Display.SetScaleArray = ['POINTS', 'p']
reflect1Display.ScaleTransferFunction = '
    PiecewiseFunction'
reflect1Display.OpacityArray = ['POINTS', 'p']
reflect1Display.OpacityTransferFunction = '
    PiecewiseFunction'
reflect1Display.DataAxesGrid = 'GridAxesRepresentation'
reflect1Display.PolarAxes = 'PolarAxesRepresentation'
reflect1Display.ScalarOpacityFunction = tPWF
reflect1Display.ScalarOpacityUnitDistance =
    0.016323514184224466
reflect1Display.ExtractedBlockIndex = 1

# init the 'PiecewiseFunction' selected for '
    ScaleTransferFunction'
reflect1Display.ScaleTransferFunction.Points =
    [-2.985599994659424, 0.0, 0.5, 0.0,
```

```
    2.689850091934204, 1.0, 0.5, 0.0]

# init the 'PiecewiseFunction' selected for '
    OpacityTransferFunction'
reflect1Display.OpacityTransferFunction.Points =
    [-2.985599994659424, 0.0, 0.5, 0.0,
    2.689850091934204, 1.0, 0.5, 0.0]

# setup the color legend parameters for each legend in
    this view

# get color legend/bar for tLUT in view renderView1
tLUTColorBar = GetScalarBar(tLUT, renderView1)
tLUTColorBar.WindowLocation = 'AnyLocation'
tLUTColorBar.Position = [0.7186564371257484,
    0.2559171597633137]
tLUTColorBar.Title = 'T'
tLUTColorBar.ComponentTitle = ''

# set color bar visibility
tLUTColorBar.Visibility = 1

# show color legend
reflect1Display.SetScalarBarVisibility(renderView1, True)

#
    ------------------------------------------------------------------

# setup color maps and opacity mapes used in the
    visualization
# note: the Get..() functions create a new object, if
    needed
#
    ------------------------------------------------------------------



#
    ------------------------------------------------------------------

# finally, restore active source
SetActiveSource(reflect1)
#
    ------------------------------------------------------------------
```

Listing 7: Temperature state file

```
# state file generated using paraview version 5.8.0

#
    ----------------------------------------------------------------

# setup views used in the visualization
#
    ----------------------------------------------------------------


# trace generated using paraview version 5.8.0
#
# To ensure correct image size when batch processing,
    please search
# for and uncomment the line `# renderView*.ViewSize =
    [*,*]`

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# get the material library
materialLibrary1 = GetMaterialLibrary()

# Create a new 'Render View'
renderView1 = CreateView('RenderView')
renderView1.ViewSize = [1336, 676]
renderView1.AxesGrid = 'GridAxes3DActor'
renderView1.OrientationAxesVisibility = 0
renderView1.CenterOfRotation = [-8.717179298400879e-07,
    0.0, -0.49999908248719294]
renderView1.StereoType = 'Crystal Eyes'
renderView1.CameraPosition = [-8.717179298400879e-07,
    -2.433399528358085, -0.49999908248719294]
renderView1.CameraFocalPoint = [-8.717179298400879e-07,
    0.0, -0.49999908248719294]
renderView1.CameraViewUp = [0.0, 0.0, 1.0]
renderView1.CameraFocalDisk = 1.0
renderView1.CameraParallelScale = 0.6298101422825638
```

```
renderView1.BackEnd = 'OSPRay raycaster'
renderView1.OSPRayMaterialLibrary = materialLibrary1

SetActiveView(None)

#
    ------------------------------------------------------------------

# setup view layouts
#
    ------------------------------------------------------------------


# create new layout object 'Layout #1'
layout1 = CreateLayout(name='Layout #1')
layout1.AssignView(0, renderView1)

#
    ------------------------------------------------------------------

# restore active view
SetActiveView(renderView1)
#
    ------------------------------------------------------------------


#
    ------------------------------------------------------------------

# setup the data processing pipelines
#
    ------------------------------------------------------------------


# create a new 'OpenFOAMReader'
jeswin_test1foam = OpenFOAMReader(FileName='C:\\Users\\XR
    -Lab\\Documents\\Jeswin Files\\Project HoloCFD\\
    openfoamdocker\\jeswin_test1\\jeswin_test1.foam')
jeswin_test1foam.MeshRegions = ['internalMesh']
jeswin_test1foam.CellArrays = ['T', 'U', 'alphat', 'p', '
    p_rgh']

# create a new 'Reflect'
```

```
reflect1 = Reflect(Input=jeswin_test1foam)

#
    ----------------------------------------------------------------

# setup the visualization in view 'renderView1'
#
    ----------------------------------------------------------------


# show data from reflect1
reflect1Display = Show(reflect1, renderView1, '
    UnstructuredGridRepresentation')

# get color transfer function/color map for 'U'
uLUT = GetColorTransferFunction('U')
uLUT.RGBPoints = [4.7709398674465147e-08, 0.231373,
    0.298039, 0.752941, 0.038694414911077536, 0.865003,
    0.865003, 0.865003, 0.07738878211275639, 0.705882,
    0.0156863, 0.14902]
uLUT.ScalarRangeInitialized = 1.0

# get opacity transfer function/opacity map for 'U'
uPWF = GetOpacityTransferFunction('U')
uPWF.Points = [4.7709398674465147e-08, 0.0, 0.5, 0.0,
    0.07738878211275639, 1.0, 0.5, 0.0]
uPWF.ScalarRangeInitialized = 1

# trace defaults for the display properties.
reflect1Display.Representation = 'Surface'
reflect1Display.ColorArrayName = ['POINTS', 'U']
reflect1Display.LookupTable = uLUT
reflect1Display.Opacity = 0.38
reflect1Display.OSPRayScaleArray = 'p'
reflect1Display.OSPRayScaleFunction = 'PiecewiseFunction'
reflect1Display.SelectOrientationVectors = 'U'
reflect1Display.ScaleFactor = 0.10001217595709022
reflect1Display.SelectScaleArray = 'p'
reflect1Display.GlyphType = 'Arrow'
reflect1Display.GlyphTableIndexArray = 'p'
reflect1Display.GaussianRadius = 0.005000608797854511
reflect1Display.SetScaleArray = ['POINTS', 'p']
reflect1Display.ScaleTransferFunction = '
```

```
    PiecewiseFunction'
reflect1Display.OpacityArray = ['POINTS', 'p']
reflect1Display.OpacityTransferFunction = '
    PiecewiseFunction'
reflect1Display.DataAxesGrid = 'GridAxesRepresentation'
reflect1Display.PolarAxes = 'PolarAxesRepresentation'
reflect1Display.ScalarOpacityFunction = uPWF
reflect1Display.ScalarOpacityUnitDistance =
    0.016323514184224466
reflect1Display.ExtractedBlockIndex = 1

# init the 'PiecewiseFunction' selected for '
    ScaleTransferFunction'
reflect1Display.ScaleTransferFunction.Points =
    [-2.985599994659424, 0.0, 0.5, 0.0,
    2.689850091934204, 1.0, 0.5, 0.0]

# init the 'PiecewiseFunction' selected for '
    OpacityTransferFunction'
reflect1Display.OpacityTransferFunction.Points =
    [-2.985599994659424, 0.0, 0.5, 0.0,
    2.689850091934204, 1.0, 0.5, 0.0]

# setup the color legend parameters for each legend in
    this view

# get color legend/bar for uLUT in view renderView1
uLUTColorBar = GetScalarBar(uLUT, renderView1)
uLUTColorBar.Title = 'U'
uLUTColorBar.ComponentTitle = 'Magnitude'

# set color bar visibility
uLUTColorBar.Visibility = 1

# show color legend
reflect1Display.SetScalarBarVisibility(renderView1, True)

#
    ------------------------------------------------------------------

# setup color maps and opacity mapes used in the
    visualization
# note: the Get..() functions create a new object, if
```

```
    needed
#
    ------------------------------------------------------------------


#
    ------------------------------------------------------------------

# finally, restore active source
SetActiveSource(reflect1)
#
    ------------------------------------------------------------------
```

Listing 8: velocity state file

```
# Base image
FROM opencfd/openfoam-default

# Set environment variables
ENV DEBIAN_FRONTEND noninteractive

# Install Node.js
RUN wget -qO- https://deb.nodesource.com/setup_14.x |
    bash -
RUN apt-get update && apt-get install -y nodejs

# Install Node-RED
RUN npm install -g --unsafe-perm node-red

# Install additional Node-RED dependencies
RUN npm install -g --unsafe-perm node-red-dashboard
RUN npm install -g --unsafe-perm node-red-contrib-finite-
    statemachine
RUN npm install -g --unsafe-perm node-red-node-sqlite

# Install ParaView
RUN apt-get update && apt-get install -y paraview

# Source OpenFOAM environment in .bashrc
RUN echo "source /opt/openfoam10/etc/bashrc" >> /root/.
    bashrc

# Expose the Node-RED port
```

```
EXPOSE 1880

# Start Node-RED
CMD ["node-red"]
```

Listing 9: YAML file for Docker

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;
using Siccity.GLTFUtility;
using System.IO;
using System.ComponentModel;
using Microsoft.MixedReality.Toolkit.UI;
using Microsoft.MixedReality.Toolkit.Input;
using System.Threading.Tasks;
using UnityEngine.UI;


#if WINDOWS_UWP
using System.Threading.Tasks;
using Windows.Storage;
#endif

public class InstantiateFBX : MonoBehaviour
{
    string assetName = "reference_velocity.py_timestep1_.
    glb";
    string saveToFolder = "";
    string saveTo = "";
    string urlFolder = "";
    public Vector3 spawnPosition;
    public ButtonController controller;
    private LoadAssetFromServer serverScript;
    public GameObject deleteButtonPrefab;
    public GameObject axisPrefabObject;

    public float retryInterval = 5f; // Adjust the retry
    interval as needed
    public int maxCount = 10; // Maximum number of
    retries before giving up
    private int retryCount = 0;

```

```
35      //server variables
36
37      public string nextcloudURL = "https://cloud.tuhh.de";
38      private readonly string username = "czr6402";
39      public string remoteFilePath = "/remote.php/dav/files
        /";
40      public string localFilePath = "/TestCFD/";
41
42      public void Start()
43      {
44          serverScript = transform.gameObject.AddComponent<
        LoadAssetFromServer>();
45          serverScript.OnDownloadCompleted +=
        HandleDownloadCompleted;
46          saveToFolder = Application.streamingAssetsPath;
47          urlFolder = nextcloudURL + remoteFilePath +
        username + localFilePath;
48      }
49      public void CheckLocalFile()
50      {
51          assetName = controller.temperatureSliderValue + "
        +" + controller.velocitySliderValue + "+" +
        controller.dropdown_mesh_Value + "+" + controller.
        dropdown_postprocessValue + ".glb";
52          Debug.Log(assetName);
53          // Create a folder inside UWP's local storage
54  #if WINDOWS_UWP
55          StorageFolder localFolder = ApplicationData.
        Current.LocalFolder;
56          saveTo = Path.Combine(localFolder.Path, assetName
        );
57  #else
58          saveTo = saveToFolder + '/' + assetName;
59  #endif
60          if (System.IO.File.Exists(saveTo))
61          {
62              LoadGLBFile(saveTo);
63          }
64          else
65          {
66              CheckOnline(assetName);
67          }
68
```

```
69      }
70
71      public void CheckOnline(string assetName)
72      {
73          StartCoroutine(SaveAndDownload(assetName));
74      }
75
76      public IEnumerator SaveAndDownload(string assetName)
77      {
78  #if WINDOWS_UWP
79          StorageFolder localFolder = ApplicationData.
        Current.LocalFolder;
80          saveTo = Path.Combine(localFolder.Path, assetName
        );
81  #else
82          saveTo = saveToFolder + '/' + assetName;
83  #endif
84          string url = urlFolder + assetName;
85          Debug.Log(saveTo);
86          serverScript.DownloadAssets(url, saveTo);
87          Debug.Log("beforeloadingfile");
88          while (!serverScript.isFileCheckDone ||
        retryCount < maxCount)
89          {
90              yield return new WaitForSeconds(retryInterval
        );
91              retryCount++;
92          }
93          Debug.Log("OnlineCheckDone");
94          serverScript.isFileCheckDone = false;
95      }
96
97      private void HandleDownloadCompleted(UnityWebRequest
        request, bool isSuccess, string saveTo)
98      {
99          if (isSuccess)
100         {
101             try
102             {
103                 LoadGLBFile(saveTo);
104             }
105             catch
106             {
```

```
107                 controller.OnSimulate();
108                 Debug.Log("File exists but not loadable")
    ;
109             }
110             // Process the downloaded data here if needed
111         }
112         else
113         {
114             Debug.Log("File dont exist publishing");
115             controller.OnSimulate();
116         }
117     }
118 
119     private async void LoadGLBFile(string saveTo)
120     {
121         GameObject model = await Load3DModelAsync(saveTo)
    ;
122         if (model != null)
123         {
124             // Attach a delete button script to the model
125             GameObject deleteButton = Instantiate(
    deleteButtonPrefab);
126             deleteButton.transform.position = model.
    transform.position + Vector3.up * 0.4f; // Position
    above the model
127 
128             // Assign the model reference to the delete
    button script
129             DeleteButton deleteButtonScript =
    deleteButton.GetComponent<DeleteButton>();
130 
131             if (deleteButtonScript != null)
132             {
133                 deleteButtonScript.modelToDelete = model;
134                 // Optionally, you can parent the delete
    button to the model for easier management
135                 deleteButton.transform.parent = model.
    transform;
136                 deleteButtonScript.SetFileNameText(
    assetName);
137             }
138 
139             // Instantiate the axis prefab
```

```
140             GameObject axisPrefab = Instantiate(
    axisPrefabObject);
141 
142             // Determine the bounds of the model to find
    a corner position and dimensions
143             Renderer modelRenderer = model.GetComponent<
    Renderer>();
144             if (modelRenderer != null)
145             {
146                 Bounds bounds = modelRenderer.bounds;
147                 Vector3 cornerPosition = bounds.min; //
    This will get one of the corners of the bounding box
148                 axisPrefab.transform.position =
    cornerPosition; // Position the axis at the corner
149 
150                 // Calculate distances from the axis to
    the far ends of the model in each direction
151                 Vector3 maxDistances = bounds.max -
    cornerPosition;
152                 Vector3 scaleFactors = new Vector3(
153                     maxDistances.x / 2f, // Scale factor
    for x-axis
154                     maxDistances.y / 2f, // Scale factor
    for y-axis
155                     maxDistances.z / 2f  // Scale factor
    for z-axis
156                 );
157 
158                 axisPrefab.transform.localScale =
    scaleFactors; // Scale the axis prefab
159             }
160 
161             // Optionally, you can parent the axis to the
     model for easier management
162             axisPrefab.transform.parent = model.transform
    ;
163         }
164         Debug.Log("Load done");
165         model.transform.position = spawnPosition;
166         MeshCollider meshcollider = model.AddComponent<
    MeshCollider>();
167         meshcollider.convex = true;
168         model.AddComponent<ObjectManipulator>();
```

```
169         model.AddComponent<NearInteractionGrabbable>();
170         //File.Delete(saveTo);
171     }
172
173
174     async Task<GameObject> Load3DModelAsync(string saveTo
    )
175     {
176         ImportSettings importSettings = new
    ImportSettings(); // You can customize import
    settings here
177
178         // Wrap the asynchronous call in a Task to return
     the loaded GameObject
179         TaskCompletionSource<GameObject> tcs = new
    TaskCompletionSource<GameObject>();
180
181         Importer.LoadFromFileAsync(saveTo, importSettings
    , (model, animations) =>
182         {
183             // Use the loaded model and animations as
    needed
184             if (model != null)
185             {
186                 tcs.SetResult(model);
187             }
188             else
189             {
190                 tcs.SetResult(null);
191             }
192         });
193
194         // Wait for the task to complete and return the
    result
195         return await tcs.Task;
196     }
197
198 #if WINDOWS_UWP
199     // Helper method to create a folder asynchronously
200     private async Task<StorageFolder> CreateCustomFolder(
    StorageFolder parentFolder, string folderName)
201     {
202         try
```

```
203         {
204             StorageFolder customFolder = await
    parentFolder.CreateFolderAsync(folderName,
    CreationCollisionOption.OpenIfExists);
205             return customFolder;
206         }
207         catch (Exception ex)
208         {
209             Debug.LogError("Error creating folder: " + ex
    .Message);
210             return null;
211         }
212     }
213 #endif
214
215 }
```

Listing 10: Instantiate model script

```
1  using System;
2  using System.Collections;
3  using System.IO;
4  using UnityEngine;
5  using UnityEngine.Networking;
6
7
8  public class LoadAssetFromServer : MonoBehaviour
9  {
10     public float retryInterval = 5f; // Adjust the retry
       interval as needed
11     public int maxRetries = 10; // Maximum number of
       retries before giving up
12
13     private int retryCount = 0;
14     // Signal to indicate that the file is available
15     public bool isFileCheckDone = false;
16
17
18     private readonly string username = "czr6402";
19     private readonly string password = "7Zdcg-kAeJQ-pEZJH
       -Nt7oL-RcaHH";
20
21
22
```

```
23
24      public event Action<UnityWebRequest, bool, string>
        OnDownloadCompleted;
25
26      // Method to be called from outside the script
27      public void DownloadAssets(string url,  string saveTo
        )
28      {
29          StartCoroutine(DownloadAndSaveFile(url, saveTo, (
        req) =>
30          {
31              if (req.isNetworkError || req.isHttpError)
32              {
33                  Debug.LogError("Download failed: " + req.
        error);
34              }
35              else
36              {
37                  Debug.Log("Download successful. File
        saved at: " + saveTo);
38                  // You can also process the downloaded
        data here if needed.
39              }
40
41              if (OnDownloadCompleted != null)
42              {
43                  OnDownloadCompleted(req, !req.
        isNetworkError && !req.isHttpError, saveTo);
44              }
45          }));
46      }
47
48      private IEnumerator DownloadAndSaveFile(string url,
        string saveTo, Action<UnityWebRequest> callback)
49      {
50          UnityWebRequest request = UnityWebRequest.Get(url
        );
51          request.SetRequestHeader("Authorization", "Basic
        " + Convert.ToBase64String(System.Text.Encoding.ASCII
        .GetBytes(username + ":" + password)));
52          request.downloadHandler = new DownloadHandlerFile
        (saveTo);
53          yield return request.SendWebRequest();
```

```
54          // Wait until the download is complete
55          yield return new WaitUntil(() => request.isDone);
56          if (!request.isNetworkError && !request.
    isHttpError)
57          {
58
59              // Wait until the download is complete (
    progress reaches 1.0)
60              yield return new WaitUntil(() => request.
    downloadProgress == 1.0f);
61              while (!System.IO.File.Exists(saveTo) &&
    retryCount < maxRetries)
62              {
63                  yield return new WaitForSeconds(
    retryInterval);
64                  Debug.Log("File not available yet");
65                  retryCount++;
66              }
67
68              if (File.Exists(saveTo))
69              {
70                  // The file is available, process the
    request here
71                  Debug.Log("File is available!");
72              }
73              else
74              {
75                  Debug.Log("File not found");
76              }
77          }
78
79
80
81          callback(request);
82          isFileCheckDone = true;
83      }
84  }
```

Listing 11: Load asset from server

```
1  using Microsoft.MixedReality.Toolkit.UI;
2  using System.Threading.Tasks;
3  using UnityEngine;
4
```

```
5  public class MessageReader : MonoBehaviour
6  {
7      [SerializeField]
8      private GameObject indicatorObject;
9      private IProgressIndicator indicator;
10
11     public mqttReceiver mqttReceiver; // Reference to the
        mqttReceiver script
12     public Interactable simulateButton;
13     private InstantiateFBX instantiateScript;
14
15     private bool isProcessing = false; // Flag to check
       if processing is ongoing
16
17     private void OnEnable()
18     {
19         mqttReceiver.OnMessageArrived += HandleMessage;
20         instantiateScript = transform.GetComponent<
       InstantiateFBX>();
21     }
22
23     private void OnDisable()
24     {
25         if (mqttReceiver != null)
26         {
27             mqttReceiver.OnMessageArrived -=
       HandleMessage;
28         }
29     }
30
31     private async void Start()
32     {
33         if (indicatorObject != null)
34         {
35             indicator = indicatorObject.GetComponent<
       IProgressIndicator>();
36         }
37         else
38         {
39             Debug.LogError("Indicator object is not
       assigned.");
40         }
41     }
```

```
42
43      private async void HandleMessage(string newMsg)
44      {
45          string[] messages = newMsg.Split(new[] { ',' });
46          if (messages.Length >= 2)
47          {
48              string firstPart = messages[0].Trim();
49              string statusPart = messages[1].Trim();
50
51              if (!isProcessing && firstPart == "Preconfig"
     && statusPart == "Success")
52              {
53                  isProcessing = true; // Set the flag to
    indicate processing has started
54                  await UpdateProgressAsync(); // Start the
     progress update process
55              }
56              else if (isProcessing)
57              {
58                  if (statusPart == "Success")
59                  {
60                      // Handle success message and update
    progress
61                      await UpdateProgressBasedOnMessage(
    firstPart);
62                  }
63                  else if (statusPart == "Error")
64                  {
65                      // If there's an error, reset the
    progress and complete the task
66                      CompleteProgressWithError();
67                  }
68              }
69          }
70      }
71
72      private async Task UpdateProgressAsync()
73      {
74          if (indicator != null)
75          {
76              await indicator.OpenAsync();
77              indicator.Progress = 0.1f;
78              indicator.Message = "Simulating...";
```

```
79          }
80      }
81
82      private async Task UpdateProgressBasedOnMessage(
        string part)
83      {
84          if (indicator == null || instantiateScript ==
        null)
85          {
86              return;
87          }
88
89          switch (part)
90          {
91              case "Simulation":
92                  indicator.Progress = 0.5f;
93                  indicator.Message = "Postprocessing data
        ...";
94                  break;
95              case "Postprocess":
96                  indicator.Progress = 0.75f;
97                  indicator.Message = "Uploading to cloud
        ...";
98                  break;
99              case "Cloudupload":
100                 indicator.Progress = 0.9f;
101                 indicator.Message = "Finishing Up...";
102                 break;
103             default:
104                 Debug.LogError("Unhandled message part: "
         + part);
105                 indicator.Progress = 1f;
106                 indicator.Message = "Loading results";
107                 // Call CheckOnline with the appropriate
        file
108                 instantiateScript.CheckOnline(part + ".
        glb");
109                 await indicator.CloseAsync();
110                 isProcessing = false; // Reset the flag
111                 simulateButton.IsEnabled = true;
112                 return;
113         }
114     }
```

```
115
116     private async void CompleteProgressWithError()
117     {
118         if (indicator != null)
119         {
120             indicator.Progress = 0;
121             indicator.Message = "Error occurred.
    Resetting progress...";
122             await indicator.CloseAsync();
123         }
124         isProcessing = false; // Reset the flag
125         simulateButton.IsEnabled = true;
126     }
127 }
```

Listing 12: MQTT message reader

```
1  using Microsoft.MixedReality.Toolkit.UI;
2  using System.Collections;
3  using UnityEngine;
4
5  public class ButtonController : MonoBehaviour
6  {
7      public GameObject mqttReceiver;
8      public InstantiateFBX buttonReceiver;
9      public sliderTextUpdater velocitySlider;
10     public sliderTextUpdater  temperatureSlider;
11     public TMPro.TMP_Text text;
12     public Interactable simulatButton;
13     public TMPro.TMP_Dropdown dropdown_mesh_pipe;
14     public TMPro.TMP_Dropdown dropdown_mesh_baffle;
15     public TMPro.TMP_Dropdown dropdown_postprocess;
16     string dropdown_mesh_Topic = "sim_control_mesh"; //
       MQTT topic for the dropdown_mesh_pipe value
17     string dropdown_postprocess_Topic = "
       sim_control_postprocess"; // MQTT topic for the
       dropdown_postprocess value
18
19     string velocity_Topic = "sim_control_velocity";
20     string temperature_Topic = "sim_control_temperature";
21     public string velocitySliderValue;
22     public string temperatureSliderValue;
23     public string dropdown_mesh_Value;
24     public string dropdown_postprocessValue;
```

```
25
26      public void Start()
27      {
28          velocitySlider.onValueChanged.AddListener(
        OnVelocityValueChanged);
29          temperatureSlider.onValueChanged.AddListener(
        OnTemperatureValueChanged);
30          dropdown_mesh_pipe.onValueChanged.AddListener(
        OnDropdownMeshValueChanged);
31          dropdown_mesh_baffle.onValueChanged.AddListener(
        OnDropdownMeshValueChanged);
32          dropdown_postprocess.onValueChanged.AddListener(
        OnDropdownPostprocessValueChanged);
33      }
34
35      public void OnVelocityValueChanged(string value)
36      {
37          velocitySliderValue = value;
38      }
39
40      public void OnTemperatureValueChanged(string value)
41      {
42          temperatureSliderValue = value;
43      }
44
45      private void OnDropdownMeshValueChanged(int index)
46      {
47          // Update the dropdown_mesh with the concatenated
         values of both dropdowns
48          dropdown_mesh_Value = "HE" + dropdown_mesh_pipe.
        options[dropdown_mesh_pipe.value].text +
49                          dropdown_mesh_baffle.options[
        dropdown_mesh_baffle.value].text;
50          Debug.Log("Dropdown mesh value updated: " +
        dropdown_mesh_Value);
51      }
52
53
54      private void OnDropdownPostprocessValueChanged(int
        index)
55      {
56          if (dropdown_postprocess.options[
        dropdown_postprocess.value].text == "Temperature")
```

```
57          {
58              dropdown_postprocessValue = "T";
59          }
60          else if (dropdown_postprocess.options[
    dropdown_postprocess.value].text == "Velocity")
61          {
62              dropdown_postprocessValue = "U";
63          }
64          Debug.Log(dropdown_postprocessValue);
65      }
66      public IEnumerator OnVelocityChanged()
67      {
68          if (velocitySliderValue != null)
69          {
70              Debug.Log(velocity_Topic);
71              mqttReceiver.GetComponent<
    MeasurementPublisher>().publishMessage(velocity_Topic
    , velocitySliderValue);
72              Debug.Log("velocityvaluepublished"+
    velocitySliderValue);
73          }
74          yield return new WaitForSeconds(1);
75      }
76
77
78      public IEnumerator OnTemperatureChanged()
79      {
80          if (temperatureSliderValue != null)
81          {
82              mqttReceiver.GetComponent<
    MeasurementPublisher>().publishMessage(
    temperature_Topic, temperatureSliderValue);
83              Debug.Log("temperaturevaluepublished"+
    temperatureSliderValue);
84          }
85          yield return new WaitForSeconds(1);
86      }
87
88      private IEnumerator CheckMeshDropdownValue()
89      {
90          string messageToSend = dropdown_mesh_Value;
91
92          // Publish the message
```

```
93          mqttReceiver.GetComponent<MeasurementPublisher>()
    .publishMessage(dropdown_mesh_Topic, messageToSend);
94          Debug.Log("Dropdown value published: " +
    messageToSend);
95
96          yield return new WaitForSeconds(1);
97      }
98      private IEnumerator CheckPostprocessDropdownValue()
99      {
100         string messageToSend = dropdown_postprocessValue;
101         // Publish the message
102         mqttReceiver.GetComponent<MeasurementPublisher>()
    .publishMessage(dropdown_postprocess_Topic,
    messageToSend);
103         Debug.Log("Dropdown value published: " +
    messageToSend);
104
105         yield return new WaitForSeconds(1);
106     }
107
108
109     public IEnumerator OnSimulateButtonPressed()
110     {
111         string topic = "sim_start";
112         string start_message = "start";
113         mqttReceiver.GetComponent<MeasurementPublisher>()
    .publishMessage(topic, start_message);
114         yield return new WaitForSeconds(1);
115     }
116
117     public void OnSimulate()
118     {
119         // Check if the temperatureSliderValue and
    velocitySliderValue are not null
120         // and the dropdown text does not start with "
    Select"
121         if (temperatureSliderValue != null &&
    velocitySliderValue != null &&
122             !dropdown_mesh_pipe.options[
    dropdown_mesh_pipe.value].text.StartsWith("Select")
    &&
123             !dropdown_mesh_pipe.options[
    dropdown_mesh_baffle.value].text.StartsWith("Select")
```

```
     &&
124             !dropdown_postprocess.options[
    dropdown_postprocess.value].text.StartsWith("Select")
    )
125         {
126             StartCoroutine(CallAllFunctionsSequentially()
    );
127             simulatButton.IsEnabled = false;
128             Debug.Log("Simulate started");
129         }
130         else
131         {
132             // Handle the error condition, log an error
    message or call an error function
133             Debug.LogError("Simulation cannot start.
    Please ensure all selections are made correctly.");
134         }
135     }
136
137     private IEnumerator CallAllFunctionsSequentially()
138     {
139         yield return StartCoroutine(OnVelocityChanged());
140         yield return StartCoroutine(OnTemperatureChanged
    ());
141         yield return StartCoroutine(
    CheckMeshDropdownValue());
142         yield return StartCoroutine(
    CheckPostprocessDropdownValue());
143         //yield return StartCoroutine (
    OnSimulateButtonPressed());
144
145     }
146
147
148 }
```

Listing 13: Button controller