POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Efficient Hybrid Rendering of Large-scale Data for Simulations on Cloud-based Platform for a Fluid User Experience

Politecnico di Torino

Supervisors
Dr. Stefano Scanzio
Dr. Edoardo Lombardi
Dr. Katherine May

Candidate
Sebastian Gutierrez Zambrano

December 2023

Contents

List of Figures

List of Tables

Acknowledgements

I want to dedicate this thesis to all the people who assisted me on my journey through my academic career, not only those who helped me academically and professionally but also those who provided emotional support during both the good and bad times. Without any of them, I wouldn't be here, and I wouldn't have come this far.

I am especially grateful to my family, who were always there for me and served as my driving force. I also want to express my gratitude to my friends and former university colleagues in Colombia with whom I shared the dream that I am finally fulfilling. To my girlfriend and my friends here in Italy who helped me adapt and cope with stepping out of my comfort zone, without them, I would have felt more alone than ever in this foreign experience.

Finally, I would like to thank Optimad and my colleagues for not only providing me with the opportunity to develop this thesis but also for opening the doors for me to start my professional career as a computer engineer.

A special dedication to my pets Cannella and Tomillo, whom I think of and miss dearly.

Summary

The rendering of large 3D models within a web-based application necessitates a considerable allocation of computational resources. The limitations within a user's web browser environment make it impractical for executing such resource-intensive tasks, therefore resulting in suboptimal user experiences due to perceptibly slow processing speeds. Consequently, different techniques have emerged to address this problem, offering the means to visualize large data efficiently within a cloud-based application.

This thesis work, in collaboration with Optimad Srl, searches for alleviating this computational load by migrating a portion of it to the cloud. This effort culminated in the development of a hybrid rendering approach, integrated into a corporate Computational Fluid Dynamics (CFD) Software-as-a-Service (SaaS) platform, dedicated to simulating Conjugate Heat Transfer (CHT) phenomena in electronic devices. This kind of application requires fast rendering capabilities for visualizations.

In a comparative analysis between local, remote and hybrid rendering, the results revealed the superiority of the hybrid approach across many aspects, including frame rate, data transfer, performance, and cost efficiency. Consequently, this thesis established the hybrid rendering approach as the optimal choice, conclusively demonstrating that it is a suitable option to deliver a fluid user experience visualization in cloud-based applications.

Chapter 1

Introduction

Throughout the years the significance of 3D visualizations in software applications has been progressively intensifying for different purposes such as entertainment, architecture, engineering medicine and education. Due to the evolution of the Internet and the massive creation of data centers, it is now possible to deliver in lighter, faster, and easy way those applications to user, so that they can use and interact without having the necessity to install and configure them, and simply access through the World Wide Web and use them as service. 3D visualization on web-based applications back to the nineties when Virtual Reality Modeling Language (VRLM) was created, supporting the creation of 3D models and interactive scenes using text-based markup and scripting language to define geometries, appearance and behavior of 3D objects and held by a browser that understands the language and performs the rendering [1].

Since then, the concept of 3D web-based applications has been exploited and many technologies like X3D, 3DMLW, gITF, O3D, COLLADA were born to increasingly enhance the computing performance, the latency and ease to develop applications and many research groups and companies adopted them to create their solutions. An example is a project from High Performance Computing Center Stuttgart (HLRS) whose objective was by means of their framework COVISE generates as a result of CFD simulations using HPC resources, geometries objects that consist of polygons, lines, triangles and textures that in client side are rendered using WebGL capabilities generating a scene from a vertex buffer array. They found out that although rendering with WebGL is generally fast even with high number of triangles, the large amount of vertex transferred from server and the using of vertex buffer objects are demanding resources which produces a perceptible slowdown in rendering [2]. All those technologies perform rendering using the GPU capabilities of the machine that runs the browser, and even though the evolution of those technologies allowed the execution of efficient algorithms to accomplish faster rendering, eventually the efficiency depends on client site machines and their

hardware limitations; as a result, multiple and complex geometries to be visualized might not yield good renderings or could take more time to be processed. For instance, 3D visualization of large models like geographic scene is not only memory expensive for all the buildings, textures and coordinates that have to be allocated but also for the huge data transfer that implies and also the GPU effort to trying to render it in real-time [3], so prefetching strategy based on prediction strategies and occlusion culling were implemented to rendering only what in the scene is and only transfer the potentially 3D models that would be showed in the scene, avoiding a high resources consuming on client-side [4]. In the same way, for situations whose objective is to represent cultural heritages where it is desired to have an accurate and detailed rendering for large 3D models of builds, multi-resolution approaches are implemented to rapidly load to the client a low-resolution version of the large models and progressively send from server the rest of the data to be rendered so while user can see and interact with the models those are getting more visually appealable without having long waiting time [5].

On the other hand, the concept of server rendering has been adopted to reduce the computational charge on client-side and leverage all the visualization processing to the server whose hardware requirements and specifications are known to the developers and guarantees that large models can be rendered hastily and avoid transferring the raw models to the user. Server-side rendering is the streaming of images or video of 3D models generated and rendered by the server and emitted to the client so this one can only display them by using JavaScript and WebGL. Many projects adopted this approach to create 3D visualizations of large data-sets using appropriate tools like ParaViewWeb that performs rendering running over a cluster with MPI in a distant server and uses a message broker to stream the result [6]. Similarly, server-rendering was also implemented in the early era of mobile devices where powerful GPUs in small devices were still narrow, so techniques to leverage heavy rendering processes to proxy with strong GPU capacity was a good option [7]. Furthermore, to exploit even more this approach, hardware-accelerated techniques were developed to improve the server rendering and achieve better performance [8]. Nonetheless, no matter how powerful a server could be to perform an efficient rendering and offer notable advantages with respect to pure client rendering, the overall performance also depends on client bandwidth; if the client does not have a seamless connectivity [9, 10], that would lead to latency issues, and since server rendering constantly delivers plenty of images this approach as a result is not ideal for interactive visualizations.

In the wake of the mentioned disadvantages of both approaches when they render lots of data, many projects adopted hybrid solutions to take fruitfully the best features of them to counteract their drawbacks and show better results. The combination of interactive rendering from client-side and the efficient computational

capabilities of the server-side offers even more benefits. One good example is a project of Augmented Reality (AR) for a mobile application where the concept of *hybrid* rendering is adopted as *split* rendering and its scope is to generate in the mobile terminal a low quality rendering and leverages the reflections, shadows and luminescence to the server since those are the difficult part to render, so at the end both parts cooperate together to generate a high quality image without using all the computational resources and reducing latency in both sides. The result made using split rendering was compared with a full high-quality rendering on server-side and showed that their quality is almost identical [11]. Similarly, Cloud Baking adopted the term of using both renderings as *collaborative rendering* and due to the growth of cloud-based applications server rendering evolved into cloud rendering, allowing to the remote rendering to be performed in servers that are far from client device. This project reduced the latency and let the more expensive operations of rendering to the cloud, so client-side just render a simpler 3D scene with a basic ambient light using WebGL and the server-side renders global illumination, diffuse lighting, specular lighting, and shadows of the scene for better photo-realistic appearances, improving the quality of the result by streaming the result of the scene processed using OpenGL with a WebSocket. The limitations were how to show an interactive scene where the server rendering always delivers the part of the scene that users wanted to move [12]. In addition, another advantage of collaborative rendering is the capacity to improve the frame per second (FPS) on video-games by distributing the GPU workload when high demand of game calculation is needed; experiments showed that the smoothness in the game rate FPS is highly improved when GPU client helps to render components of a remote rendering in a server that has a better GPU processing time [13].

Optimad S.r.l with its associate ESTECO SpA are developing a Computational Fluid Dynamics (CFD) application as Software-as-a-Service (SaaS) based on a micro-services architecture in which it seeks to enable non-experts in this field to perform engineering simulations and heat-sink analyses of electronic devices for Conjugated Heat Transfer (CHT) problems. One of the desired functionalities of this application is the fluid visualization and interaction of the 3D models that the potential user can load within the application. Since the modality of the application is based on a client-server paradigm, the visualization of the model will be through the client's web browser. The problem is that the rendering process should not be exclusively done by the client's browser, since the size of the model to be analyzed and the computational power of their devices are unknown; on the other hand, rendering on the server side can be a viable and effective solution, however, since the amount of data that will be transmitted through client-server communication is unknown, a constant request for rendering could lead to a slow model interactive visualization. Nevertheless, visualization rendering must be fluid and user-friendly regardless of the size of the model. In this thesis, the main goal is to implement a

service of the main application that allows the user to interact with large 3D models in a fluid, effective and computationally efficient manner, both client and server side, in such a way that the server performs the entire rendering of the model when the user is not actively interacting with it, while the client's browser can perform a local rendering of a lighter version of the model sent by the server when the user is interacting with it; minimizing both latency and size of data transfers without compromising the user experience.

The thesis is organized as follows:

- Chapter 2 presents an analysis of the frameworks that can offer the capabilities to develop the proposed hybrid rendering into the SaaS.
- Chapter 3 presents the methodology, technologies and workflows used for the development of the application.
- Chapter 4 introduces the current project context, the problematic and the proposed solution.
- Chapter 5 presents all the software engineering process on the construction of the solution proposed for the application.
- Chapter 6 shows the tests and the results obtained.
- Chapter 7 contains the conclusions.

Chapter 2

Analysis of frameworks for local and remote rendering in the web

2.1 Introduction

This section will describe some suitable software tools for 3D visualization that offer state-of-the-art rendering and are designed for usability in research fields such as geology, astrophysics, medicine, and engineering. For the purposes of this thesis the tools that will be mentioned are compared to decide the best choice to obtain an efficient and usable implementation of the hybrid rendering that is intended to be performed in this case of study, considering that those tools are appropriate for the context of CFD application in a cloud-based platform, and the compatibility and usability with the established frameworks that are used in the development of SaaS must be guaranteed.
To evaluate each tool will be considered criteria such as performance, usability, flexibility, scalability, compatibility, support from vendor and costs, also mentioning some advantages and limitations. The decisive criteria will be the ease of communication and synchronization between both local and rendering tools to produce the desired output.

2.2 Remote rendering tools

2.2.1 VTK

The Visualization Toolkit (VTK) is an open-source collection of software tools system for 3D computer graphics and image processing that allows the development of custom visualization applications starting from implemented visualization pipeline. It provides a set of C++ libraries for creating and rendering graphics, as well as Python and Tcl Interfaces for easy scripting [14].

It is one of the most popular toolkits adopted in various commercial applications because of its capabilities that provide a range of visualization techniques including volume rendering and surface rendering. It is intended to be used for a wide range of developments for end-user applications such as medical imaging and scientific visualization; since it runs on multiple operating systems including Windows, Mac OS, and Linux. Moreover, VTK has a large and active user community that contributes to its development and support, which makes it easy to extend through its modular architecture and support for plugins. VTK relies on Pyhton wrapping to expose services to the web using WebSockets for communication.
Nevertheless, VTK requires a steep learning curve which do not make it easy to use, especially for beginners, considering that despite of its wide community, some parts of the library are poorly documented or lack examples. In terms of performance, VTK is not capable of rendering seamlessly large data sets and may not provide advanced graphic features like real-time ray tracing. However, integration with MPI, allows VTK to have an excellent support for scalable distributed-memory parallel processing.

2.2.2 Mayavi

Mayavi is a free open-source scientific data visualization tool for creating 3D plots and visualizations that is closely integrated with the abundant ecosystem of Python packages for interactive manipulation of the data and its automation and reproduction. It is built on top of VTK library with the objective of being highly customizable, providing tools for developing scientific applications for interactive visualizations. Further to being just a visualization library, Mayavi offers plugins, dialogs and widgets for ease scientific workflow [15].
Since Manyavi is a higher-level interface of VTK, it is capable of creating complex 3D visualizations, also compatible with a wide range of operating systems and it inherits the support of rendering engines like OpenGL, Mesa and X3D. Unlike VTK, its learning curve is lower because it does not require technical expertise and high coding skills to use effectively, however, it can be steep for those who are not familiar with Python. The tight integration with Python can provide benefits such as the usage of package like mpi4py for enabling parallel processing to distribute computation across multiple nodes and processors. Although Mayavi is designed to be fast and efficient there are some performance issues associated with Python overhead, where VTK directly in C++ is more efficient to use. Moreover, Mayavi is designed to be run on a stand-alone machine, so it does not support remote rendering, but it is possible to extend VTK's capabilities which allow rendering to be performed on a server and the results to be streamed to a client machine in real-time.

2.2.3 Open3D

Open3D is an open-source library for rapid software development in 3D data processing. It uses efficiently powerful data structures in both C++ and Python and provides from them useful algorithms for point cloud, mesh processing and 3D visualizations. It is a user-friendly API, which simplifies the development of 3D applications having a low learning curve. Open3D was engineered for high performance so can handle large-scale point clouds and meshes using algorithms that are optimized for parallel computing. It supports multi-threading, distributed computing and GPU acceleration [16].
Open3D has limited platform support, it was mainly designed for Linux and MacOS, so features are not supported on Windows and building the framework in this operating system has another procedure. In addition, since it is an emerging library, its community is still small and supporting is limited although its documentation is growing but it needs to be polish for beginners.
A key feature of Open3D is the possibility of performing remote rendering in a C++ or Python server by means of WebRTC as protocol communication. This method is interesting since WebRTC was initially designed to be a peer-to-peer protocol to communicate between browsers and transfer mainly video, images and audio without using a server in the middle. However, Open3D achieves to implement in a minimalist way this protocol in a standalone application that exposes a port in which any browser can reach to get directly the information transmitted in real-time [17].

2.2.4 ParaView (ParaViewWeb server-side)

ParaView is built on top VTK, it is created by Kitware the same company that created VTK. It is a general data analysis and visualization application that supports a variety of data formats, including structured and unstructured grids, point clouds, and polygonal data. As VTK, ParaView is an open-source application compatible with many platforms making it accessible to a wide range of users. It is used in research fields such as engineering, geology, astrophysics, etc. It shares the same design of being an application for developers as well as end-user software. A notable difference between VTK and ParaView is that the latter has better performance since it can support larger data by working across multiple machines simultaneously in a cluster (High Perfomance Computing), making highly scalable [18].
On the other hand, although for end-user ParaView has an user-friendly interface to navigate and create visualizations, for developers it has a steep learning curve and limited documentation in some areas.

Regarding the web communication, ParaViewWeb is a web-based interface for ParaView, it is not a re-implementation of it, it is just the infrastructure to communicate ParaView capabilities and its visualizations using a client-server architecture. Similar to VTK, as web server, they share the same core of communication using a Python wrapper, however ParaViewWeb has compatible capabilities in the setup of the rendering, reducing the amount of code needed to transmit remotely the rendering results [19].

2.2.5 Datoviz

Datoviz is a high-performance open-source Python library for creating interactive scientific data visualizations supporting graphical primitives such as meshes and volumes. Datoviz provides a C API that allows not only to be wrapped in Python but also in other languages such as R, MATLAB, Julia or Rust. It also provides an integrated graphics stack for 2D, 3D, graphical user interfaces and natively supports efficient interactions between rendering and general-purpose GPU computing. The key strength of Datoviz is the performance because of its state-of-the-art rendering engine Vulkan which is a low-level graphic API created by Khronos consortium, to be the successor of OpenGL. Vulkan provides scalable, low-overhead, fine-grained control of GPU and GPU-CPU and removes abstractions in previous generation graphics API in order to deliver the highest performance, which makes it suitable for Datoviz for handling especially huge data-sets [20, 21].

Be that as it may, Datoviz is currently in an early development phase, so the library is not stabilized and some of its use-cases are not implemented yet. Hence, it has limited documentation and lacks community support which makes it hard to learn. Moreover, it is only compatible with Linux and MacOS platforms, which can be a limitation for developers who need to work on other platforms.

Finally, since Datoviz is an emerging library, it is currently a stand-alone library and although in the future it is planned for Dataviz to have a real-time video streaming with GPU-powered visualization server for remote rendering, nowadays that feature is still in development and the implementation for streaming rendering images shall be leveraged to developers who use the library. Furthermore, it is also planned to have a web integration using a state-of-the-art web standard WebGPU, successor to WebGL, which aims to be the evolution JavaScript API for accelerated graphics computation to modern 3D capabilities based on APIs provided by Vulkan, Metal and Direct3D 12, so high-performance local rendering in web browsers can be achieved since WebGPU does not depend on a port API (i.e OpenGL) to the local machine but it uses directly GPU capabilities [22].

2.3 Local rendering tools

2.3.1 Three.js

Three.js is a JavaScript library and API for creation and displaying of 3D graphics on the web. It offers several tools and functionalities to create complex 3D scenes, animations and interactive manipulation of the models in the scene. It uses WebGL so it has all the capabilities of this API and a wide range of compatibility with several platforms including desktop and mobile browsers. It was created to adopt all the functionalities that WebGL can offer without the complexity of it, so it is intuitive and easy for developers as they do not need to know low-level graphics programming. It is open-source and has a large community that contributes to the documentation and provides plenty of examples and resources to help with the development. It provides 3D features such as texture, lighting and shading, and it supports different 3D models from meshes to 3D file format that can be easily imported to the scene [23].
However, Three.js can be resource-intensive especially when rendering complex scenes with high levels of detail that leads to performance issues on devices who does not have enough computational requirements. Furthermore, Three.js does not support by default remote rendering, it is possible however to implement an ad-hoc proxy for receiving and displaying images from a remote server by using HTTP requests or custom web sockets and Three.js canvas capabilities, which potentially leads to a low transfer of data and may limit the seamless interaction with the scene.

2.3.2 X3DOM

X3DOM is an open-source framework for creating and displaying 3D graphics on the web using x3D language. It uses X3D standard, which is similar to WebGL but with a declarative syntax. Its goal is to be embedded into HTML5 DOM so interactive web applications can rapidly integrate and synchronize 3D models with the DOM using HTML tags, without the need for plugins or specialized software. It is accessible to a wide range of users since uses HTML and it is compatible with the most common technologies on the web and it can be used in any browser that supports HTML5, making it versatile and suitable for any device. It is extremely easy to use knowing the basics of web development and no need of advanced knowledge in computer graphics [24].
Unlike Three.js, setting a scene for displaying a simple 3D model is easier, since no lines of JavaScript code are needed to create the scene, adjust the camera and show the object. This simplicity is a strong feature as well as a weak one because it limits the complexity of 3D scenes and as a result performance issues arise. In the same way, it is not too easy to customize scene and objects programmatically.

Besides, it does not have a large community so, compared to other technologies, it is difficult to find resources that help to learn advanced features of the framework. Originally, X3DOM was not thought to be a real-time communication for 3D graphics, however, as mentioned in section 2.2.3, WebRTC is suitable to stream content in HTML in a remote peer connection and its protocol allows fast communication in the web. The integration of these two technologies has shown prominent results [25]. It is still necessary to create a custom way to synchronize a bilateral communication if local and remote rendering are planned to be used together using WebRTC.

2.3.3 ParaView (ParaViewWeb client-side)

As mentioned in section 2.2.4, ParaViewWeb is a framework that leverages the power of rendering either to VTK or ParaView and displays the results in the web. The JavaScript library grants the possibility of performing 3D visualizations, analyzing and manipulating data in web server by relying on a remote processing server. ParaViewWeb offers an attractive and user-friendly interface for the user to interact with their visualizations and it is compatible with most of the browsers today. A crucial property is the communication core for real-time visualization, thus interaction with data visualization can be performed and changes in the scene can be seamless even though is a remote environment, this however depends also on the network latency.
A notable disadvantage for the client library, besides the mentioned previously in its server-side version, is the low flexibility for local rendering in the user's machine; it is possible to perform local rendering in particular cases where small 3D geometries are displayed in the scene, so the free manipulation of the WebGL canvas in the client side is not possible. This feature is not actively supported since ParaViewWeb focus is remote rendering [26].

2.3.4 VTK.js

VTK.js is a JavaScript library for 3D graphic visualizations, volumetric rendering and data processing based on VTK C++ library. It includes several rendering techniques and accelerated volume rendering using ray casting developed initially in WebGL 1.0 but then adapted in version 2.0. It is not intended for end-users but for developers who can develop a web-based application using the VTK original framework. Unlike ParaViewWeb, VTK.js has high-performance visualization uses WebGL for local rendering and it is designed to be modular so it is extensible, and developers can easily add new functionalities and features to their applications. In fact, VTK.js aims to be the replacement of ParaViewWeb in client-side because it has not only the same capabilities including remote rendering but also modern code, and compatibility with popular frameworks such as React and Angular. Moreover,

it has an active community of developers who continuously improve the library and make it efficient including also in their source code the possibility of using state-of-art rendering engine WebGPU [27].
For the remote rendering, VTK.js shares core communication with VTK server and ParaViewWeb server, so it is versatile and can fit perfectly with any of the two. No additional coding for transmission of stream images is needed, it is enough to select a port of communication.
Nonetheless, VTK.js, as its C++ version, requires a significant amount of learning and its documentation is not as comprehensive as other libraries.

2.4 Selection

After describing the potential frameworks to be selected with their strengths and points to be improved, some criteria were established to compare them and choose the more suitable tool that could be adopted in the on-course developing SaaS for the main scope of the thesis. Table 2.1 and Table 2.2 summarize the characteristics based on the criteria of each tool for rendering and local rendering respectively.

First of all, from Table 2.1 it can be appreciated that Datoviz is a modern framework that was born in recent years and in future it can be even more powerful and wide-spread, but despite its state-of-the-art technologies, it is still growing a need to be improved especially in compatibility and remote communication. In the same way, Mayavi is a good option, it has good documentation, a good interface, and it is easier to use than VTK; however, it does not have as good performance as the other tools and it does not have by default a core communication for remote rendering. VTK, Open3D and ParaView are the remaining libraries to be selected.

Second of all, regarding browser rendering as illustrated in Table 2.2, X3DOM is a good tool for showing in a fast and easy way 3D models without a high degree of knowledge in computer graphics. However, those benefits are also limitations on its performance, restricting the possibility to customize rendering pipelines and, nowadays it does not have modern rendering engine. Using WebRTC is a good way to communicate with a server for the exchange of data in visualization, but it is necessary to implement it in a proper way. Three.js has a steeper learning curve than X3DOM but it is still a fast option to display 3D models on the web, it has good documentation and a large community. The main problem is that is a general-purpose library, so additional features like remote rendering are not implemented and it is leveraged for developers. ParaViewWeb, on the contrary, was developed mainly for remote rendering and browser rendering performance is limited. As a result for VTK.js was selected for local rendering on the web, it is constantly updated using state-of-the-art rendering engine, it can handle in a good

manner local and remote rendering and it has a high performance.

Finally, as VTK.js was selected in local rendering, Open3D had to be discarded. Although Open3D is modern and it has a good documentation, low learning curve and interesting way to perform remote visualization in web; it was discarded because VTK.js offers the possibility to alternate between local rendering and remote rendering to another server that executes either VTK or ParaViewWeb, providing a high degree of usability in terms of communication. Furthermore, VTK and ParaView are tools that, despite their absence of good documentation, have been used for Optimad in previous projects as stand-alone libraries, thus, a high level of knowledge is already acquired. ParaViewWeb server-side has been selected because of its performance, it was designed to support larger data than VTK and to work with parallel computing systems.

Criteria	VTK	Mayavi	Open3D	ParaView	Datoviz
Performance	Medium	Medium	High	High	Very High
Usability	Low	Medium	High	Low	Low
Rendering engine	OpenGL	OpenGL	OpenGL	OpenGL	Vulkan
Scalable	✓	✓	✓	✓	✓
Compatibility	Cross-platform	Cross-platform	Windows limitations	Cross-platform	No Windows
Documentation	Limited	Good	Good	Limited	Limited
Open-source	✓	✓	✓	✓	✓
Remote communication	VTK server	Not included	WebRTC	ParaViewWeb server	Not included

Table 2.1: Comparison between remote rendering tools.

Criteria	Three.js	X3DOM	ParaViewWeb	VTK.js
Performance	Medium	Low	High	High
Usability	Medium	High	Medium	Medium
Rendering engine	WebGL	X3D	WebGL	WebGL/WebGPU
Compatibility	Cross-platform	Cross-platform	Cross-platform	Cross-platform
Documentation	Good	Good	Limited	Limited
Open-source	✓	✓	✓	✓
Remote communication	Not included	WebRTC	VTK/ParaViewWeb server	VTK/ParaViewWeb server

Table 2.2: Comparison between local rendering tools.

Chapter 3

Development context

3.1 Introduction

This section will describe the methodology, technologies and workflows for the development of the application. Although the scope of this thesis is to show only the development of the 3D models visualization and interaction, the context also exposes the whole picture of the entire development for the application and the pipelines of working that Optimad and ESTECO implement to achieve their goal. Most of the technologies and frameworks were adopted based on previous experiences in other projects that both enterprises have worked on. The selection of the tool in charge of the rendering in Chapter 2 had to fulfill the technologies constraints described in this section.

3.2 Methodology

In order to ensure a flexible, iterative, incremental, and optimal workflow for the development process, *agile methodology* was implemented in this project. The methodology was used to plan, describe, assign, execute, and review tasks required to complete the SaaS platform.
Agile methodology is known for its flexibility in the development stages, promoting a higher degree of collaboration between teams, and facilitating continuous improvements. In contrast to methodologies like Waterfall and V model, which follow a linear approach and often lead to significant time delays and rigid modifications during execution, agile methodology allows for iterative and incremental development. The project is divided into smaller interactions called sprints, enabling faster evaluation, testing, and refinement of requirements without impeding the overall workflow.
Another key aspect of agile methodology, considered in the context of this project, is the recognition that some requirements may be volatile, and therefore the core

functionalities are prioritized throughout the development process. This approach allows for value delivery based on customer feedback and active involvement in the development. The customer can assess the ongoing results and verify if they meet their needs, making it possible to modify requirements for a more accurate solution. For the same reason, agile methodology relies on empirical observations to make decisions, rather than relying solely on the initial plan. This approach provides better predictability of the final outcome and allows for adjustments during the sprints. As a result, teams working on different features need to have transparent communication to bring constant reviewing and testing between them to gradually improve the product in each sprint [28, 29, 30]. Fig. 3.1 shows a diagram of Agile methodology.

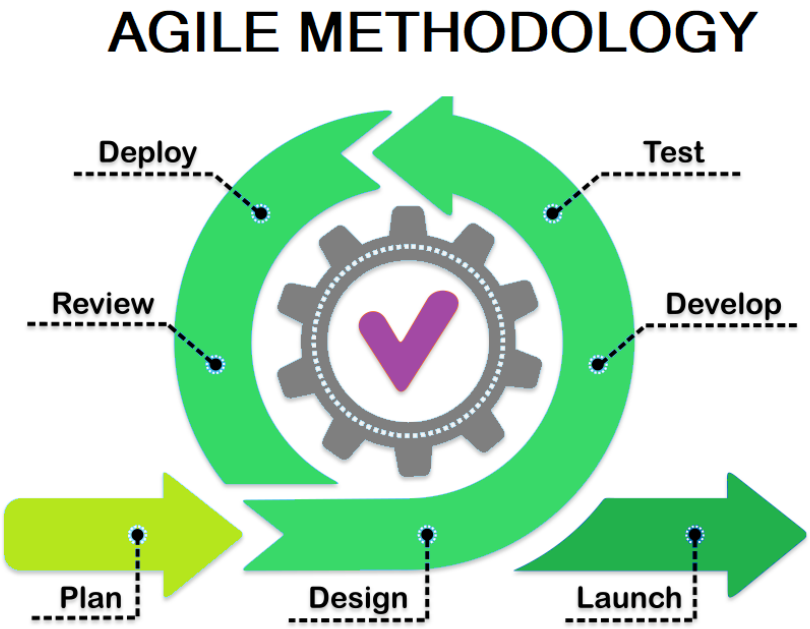


Figure 3.1: Agile methodology.

Agile methodology has several models that derive from it, like Scrum and Kanban. Both models work with the same philosophy of being iterative and incremental with a high degree of flexibility. A notable difference is how the improvements and progress are discussed and the existence of roles. Scrum presents specific roles and hierarchies to perform certain activities and the progress is shown as a discussion of the opportunities, difficulties, and experiences to improve [31]. On the other hand, Kanban in project management context, proposes a visualization paradigm where tools like boards and cards are used to represent the state of the current tasks to see the overall progress of the project and simultaneously the progress of each singular task. In Fig. 3.2 is shown the board proposed by Kanban for visualizing

the progress and organization of a project.

Figure 3.2: Kanban Board.

The main purpose of the Kanban board is to represent the task as a card to allow the team members to track the progress of the project in a visual way. Each card could have a brief description of a user story that can be grouped in subdivision of a bigger feature to be implemented. They move along the columns to represent their current state. **Backlog** refers to the user stories to be implemented with a given description, priority, category and assignee, then in the **Doing** column, are shown those tasks that are being developed by one or more members. Once a task is finished, it goes to **Review** column so the other members of the team and customers can give feedback, allowing the flexibility of modification; and finally when a task is successfully finished and reviewed, goes to **Done** column to keep track of what has been done and what remains. All the cards can move forward and backward if it is necessary thanks to the flexibility of Agile [32].
Optimad and ESTECO team has chosen Kanban model as a methodology for the development of the SaaS platform because of the previously mentioned advantages of Agile methodology, the usability that Kanban offers as it can be used not only for software engineering tasks but also for tasks of other nature related to the project and the positive results found in the post-mortem of previous projects.

3.3 Technologies

In a micro-services approach, individual components of a system operate autonomously, dynamically adjusting their resource allocation based on demand. They collaborate

seamlessly with one another to collectively deliver a wide range of functionalities within the overarching system. To follow this design pattern, a series of technologies were adopted in this project to develop the cloud-native application with a certain level of knowledge and experience for optimizing the efficiency on the developing time, taking control of versioning, monitoring the progress and testing current implemented requirements.

3.3.1 Development

Java is one of the most suitable programming languages for developing micro-services architectures due to its readability and reliability. Using Java to develop this kind of architecture is advantageous since its annotation syntax is simple to use and plenty of frameworks are built on top of it. MicroProfile, SpringBoot, Eclipse Vert.x and Quarkus are some of the most notable frameworks used nowadays [33]. Quarkus is used in the development of this project for being a framework that incorporates and implements features from MicroProfile and Vert.x to provide extended capabilities for micro-services such as authentication, REST services, security and mailing. In addition, Quarkus is designed to be efficient and lightweight in the creation of applications for containerized environments. It supports containerization technologies such as Docker and Kubernetes with a focus on optimizing memory consumption while providing a developer friendly environment [34].
Those characteristics are fundamental for this project where different sub-systems are in charge of running simulations, assessing and displaying geometries and handling user data need to work together in an efficient and scalable way in order to bring to the user a seamless and transparent experience leveraging the bigger procedures to the pieces running in HPC machines while the user can interact with other features.
Given the flexibility of Quarkus for integrating other frameworks with different scopes to complement an application, several database engines could be integrated easily and configured for development. In this particular case a non-relational database like MongoDB was considered because of the context of the methodology mentioned in Section 3.2 where requirements can change along the life cycle of the project, so a flexible database that allows to change the initial structure of the data without affecting the current data is suitable and appropriate. Furthermore, developing with containerized MongoDB can bestow an easy way to implement, test and interact with the database while allowing the transition to a production guaranteeing the same behavior [35].
Vert.x serves as the main interface for creating and managing various asynchronous components and services within an application. It is used in this project as an access point to storage in a development and production environment, files regarding the eventual manipulation of geometries that the user can upload and simulation

Figure 3.3: Frameworks used for the project development.

results. This is useful since no manual modification of the path where the information will be stored is needed as Vert.x works as an interface for any final cloud object storage [36].
Angular is used as a front-end framework for building the interface application delivered to the user that consumes the micro-services. It provides a robust set of features and tools for developing a rich interactive user interface. It offers RxJS library, which embraces the reactive programming need in a micro-services architecture since it has to handle asynchronous data streams such as HTTP requests and responses. Moreover, Angular follows the Model-View-Controller (MVC) architectural pattern using Typescript, which is a typed super-set of JavaScript, improving code organization, components re-utilization and enhancing the productivity [37].

3.3.2 Tooling

For the automation building and organization of build script tasks is used Gradle, which grants key features like supporting wide range of project structures and handling projects of different sizes. For the construction of the platform is fundamental to support multi-module builds, because separated projects need to build and work all together to represent each service in the architecture. It is also helpful since simplifies the dependency management providing a dependency resolution mechanism and extracts external dependencies from remote repositories like Maven Central [38].
In addition, IntelliJ IDEA, the IDE currently used for having a good integration development environment for Java, has also an excellent integration with Gradle,

supporting Gradle-based projects seamlessly, easing the build scripts and performing of various tasks. IntelliJ automatically detects the presence of a Gradle build script and configures the IDE accordingly.
Moreover, IntelliJ offers robust support for static type checking, allowing developers to leverage its code assistant. Continuously evolving, it provides numerous dynamic suggestions through IntelliSense. As an instance, the editor can propose available methods along with their expected parameters, a valuable functionality for expediting development [39].

3.3.3 Testing

On the back-end side, Quarkus delivers a module for testing, which is designed to facilitate the development, integrating the benefits of Mockito's mocking capabilities with the Quarkus infrastructure to simplify the process of writing unit tests and promoting good testing practices by decoupling resources and dependencies from the test and making work in an isolated way. Similarly, Quarkus makes use of JUnit annotation to organize and modulate the setup of integration and unit testing in order to execute more than one test concurrently.
The testing framework on the front-end side used for testing Angular components, REST calls and interface test is Jasmine, which offers complete capabilities to perform tests, mocking and spying the components needed to assess and providing a well-organized hierarchical structure that helps in maintaining a large suite of tests. Jasmine works along with Karma to generate Continuous Integration (CI) and Continuous Development (CD) pipelines, executing live test and generating test reports to keep track of workflow.

3.3.4 Deployment

During the development phase Docker and Kubernetes files and scripts are used locally to setup how the micro-services are going to be connected and configured in terms of CPU, memory and network performance, with the purpose of having an easy and reliable transition of what is being developed and what is going to be produced at the end. Amazon Web Service (AWS) is going to be the responsible for deploying the final micro-service architecture and it will display Kubernetes over it, so the orchestration of multiple containers is going to be simplified since the infrastructure is handled by the services that AWS offers. Furthermore, services of monitoring, hosting, storage and security are going to be provided by AWS guaranteeing that the whole application will work efficiently and optimally. Since in the future this application will be maintained and new features are going to be released, AWS allows to continuously update the application and infrastructure without cutting functionalities and performances [40].

3.3.5 Control versioning and progress

Github is being used as the git repository to handle the control versioning during the development. The possibility of collaborating seamlessly among team members on the same project, allows a higher degree of distribution of tasks without worrying on the conflicts between versions. Working remotely between teams facilitates code reviewing, discussion and documentation. Additionally, in the project is being used Github Actions, which creates CI/CD pipelines in which the building and testing of the application is triggered every time a new feature is being implemented to the main workflow of development to ensure that any feature does not cause any problem to the current state of the project.
To keep track of the user stories and the tasks to be resolved, Jira is the tool used in the platform that allows to visualize as the methodology explained in Section 3.2, the Kanban board where the project management gets easier to understand and extends the planning of future features to be implemented. One of its characteristics is that any card of the board can be seen as an issue that then is categorized into *bugs*, *tasks*, *user stories* or *spikes*; and then those are encapsulated on bigger events called *epics* where represent a big functionality to be developed. Communication and collaboration through descriptions, comments, mentions and file attaching that Jira provides, represent a huge advantage to reporting and analyzing that the progress is being going fruitfully [41].
For communicating technical issues, development progress or questions, Mattermost is used, which is the main channel of communication between the team members. Here, all the members share ideas, make calls, discuss relevant topics and keep updated on all the progress that has been made.

3.4 Development workflow

Prior to entering the development phase, it is imperative to provide an introduction to the internal software development process carried out within the company. This process is succinctly outlined in Figure 3.4
This is an iterative process that typically spans one to two weeks per defined task, which is essential for developing project features. Its purpose is to maintain a coherent and accurate project progression, fostering ongoing software construction enhancement. Comprising various phases aligned with software development best practices, prioritizing phases beyond raw coding. These stages contribute significantly to achieving an optimal end result while minimizing the expense of rectifying issues during the early development stages.
Exploring deeply into the process, the first step involves taking a task that has been initially placed in the *Backlog* on Jira's Kanban board. This task is then assigned to a developer and transitioned to the *Doing* column, signifying that a dedicated team member is actively working on it. Each task is structured as a *user story*,

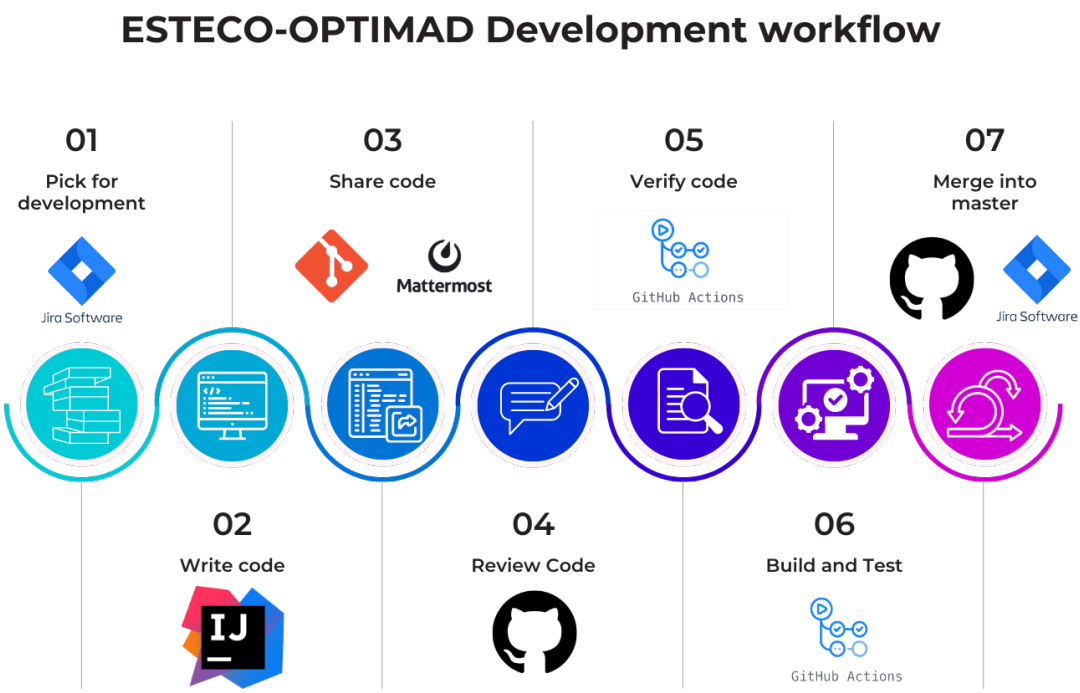


Figure 3.4: ESTECO-OPTIMAD Development workflow.

including a clear description of the features' requirements, accompanied by specific acceptance criteria. These criteria play a crucial role in the subsequent code review process, ensuring that all necessary aspects are thoroughly examined.
During the coding phase, each developer ensures not only the creation of functional code that addresses the required features but also incorporates good programming practices throughout the development process. These practices encompass the establishment of unit and integration tests that comprehensively cover each individual component and the interaction between them, respectively; this ensures that all elements function effectively both in isolation and when integrated to work seamlessly together. To facilitate streamlined code reviews for peers, lint checks are implemented, enhancing code readability and ensuring a clear presentation of new implementations. As outlined in 3.3, IntelliJ in conjunction with Gradle and Quarkus aids in generating a project framework. This empowers developers to easily incorporate new modules, libraries, or components as necessary, thereby guaranteeing the compatibility of each new service with its predecessors.
Once a developer deems a feature as complete, it is shared using Git, and a new pull request is initiated to merge the feature into the main project. The implementation of the said feature is then communicated through Mattermost, where peers are asked for their review and feedback.

During the review phase, team members collaborate by sharing their feedback on the code. They provide suggestions for improvements and, if necessary, point out areas that require changes. This ensures that any issues or deviations from the acceptance criteria are promptly identified. Additionally, the team focuses on maintaining a clean coding style. This effort goes beyond passing lint tests, as they aim to enhance readability. Once the team members have thoroughly reviewed and tested the code as users, the working branch becomes almost ready for merging.

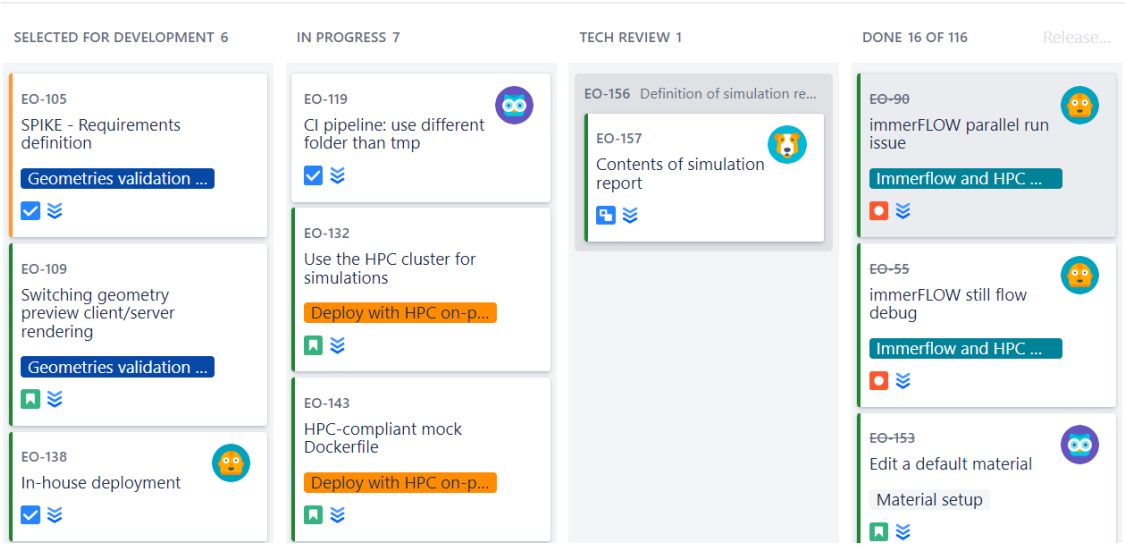


Figure 3.5: Jira Kanban's board.

Simultaneously, alongside the code review process, the CI/CD pipeline is initiated using a script. This script builds the application with the latest changes and performs various checks, such as verifying coding standards, identifying conflicts with libraries, and validating versions. Once the build is complete, the same CI/CD pipeline runs all integration and unit tests. These tests ensure that there are no conflicts with the current branch, thus guaranteeing a seamless integration of the new feature when it is merged.
Finally, in the absence of conflicts, Github provides the choice to squash all commits into a single one before merging into the master branch. Following this, the corresponding task is marked as complete in Jira and transitioned to the *Done* column. This approach ensures comprehensive issue tracking, enabling project monitoring and progress assessment. The Kanban board is consistently updated throughout each workflow phase, exemplifying its role in task management within Jira, as illustrated in Figure 3.5.

Chapter 4

Project goal

4.1 Context and current state of the project

Optimad S.r.l, in collaboration with its affiliate, ESTECO SpA, is currently in the process of developing a Software-as-a-Service (SaaS) Computational Fluid Dynamics (CFD) application. This application is structured upon a micro-services architecture designed to empower users who may not possess extensive expertise in the field of fluid dynamics to facilitate engineering simulations and heat-sink analyses for electronic devices, specifically addressing Conjugated Heat Transfer (CHT) problems.
In order to achieve their goal, users have the capability to upload their own geometries, facilitating the execution of simulations with customizable inputs. This process enables users to assess the suitability of these geometries for their intended purposes. Users have the possibility to initiate numerous simulations, employing a variety of geometries and inputs, and subsequently receiving detailed reports summarizing the outcomes for each individual simulation.
It is important for users to be provided with the capability to verify and ensure that the geometries they upload align precisely with their intended selections. In pursuit of this objective, the project has incorporated a feature that facilitates the retrieval and presentation of a static preview of the uploaded geometry from an isometric camera perspective. Figure 4.1 illustrates the appearance of this functionality as it is presented to the user.

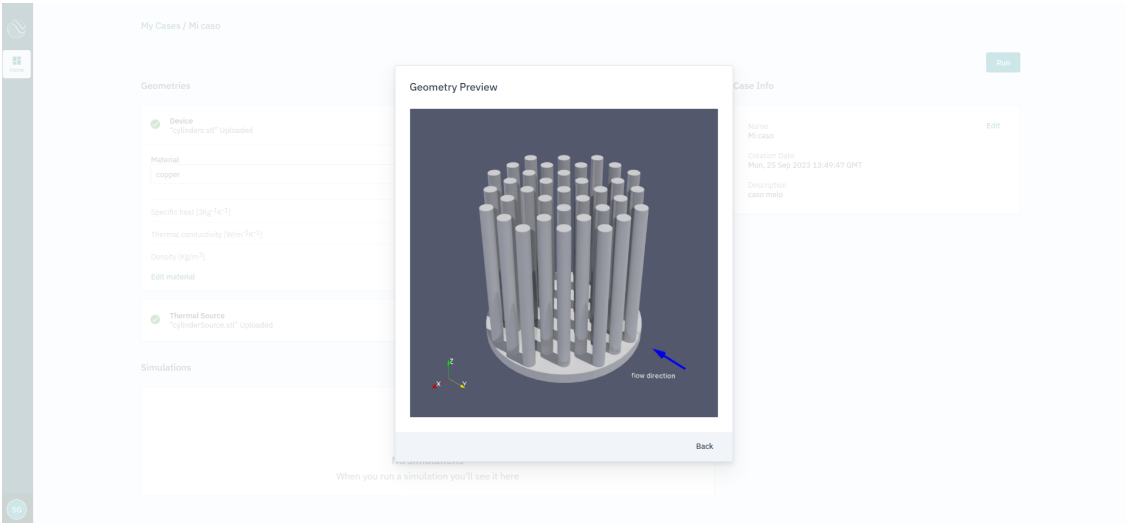


Figure 4.1: Static preview of geometry

While this feature offers the capability to visualize an uploaded geometry to confirm its alignment with the user's intended upload, it is important to note that it provides only an isometric view, offering a limited level of detail. Optimad is concerned about the possibility of users uploading geometries that closely resemble one another, with subtle variations that may only become discernible upon closer examination through zooming and rotation.
For the mentioned reasons, this project aims to incorporate a new functionality to replace the existing one. This new feature will enable users to visualize and interact with the uploaded geometries, offering a comprehensive examination of the entire 3D model. Crucially, this functionality must ensure seamless visualization of the geometry, regardless of its size or complexity. This optimization will efficiently manage server system resources, alleviating the need for users to use their computational capabilities. Moreover, it is imperative that these enhancements do not lead to a substantial increase in application costs.

4.2 Problematic

Displaying a 3D model visualization within a web-based application has become increasingly feasible, owing to the technological advancements mentioned in Chapter 1. Modern web browsers now feature robust graphics engines that harness the computational resources available on client machines to efficiently render geometries. This rendering process is commonly referred to as *Local rendering* since it is processed entirely on the client-side. Figure 4.2 provides a visual representation of the local rendering process.
Nonetheless, a key implication of this approach is that the 3D model must reside

within the browser's memory. Consequently, the geometries comprising the model must either be retrieved from a server or re-uploaded by users each time they access the application.

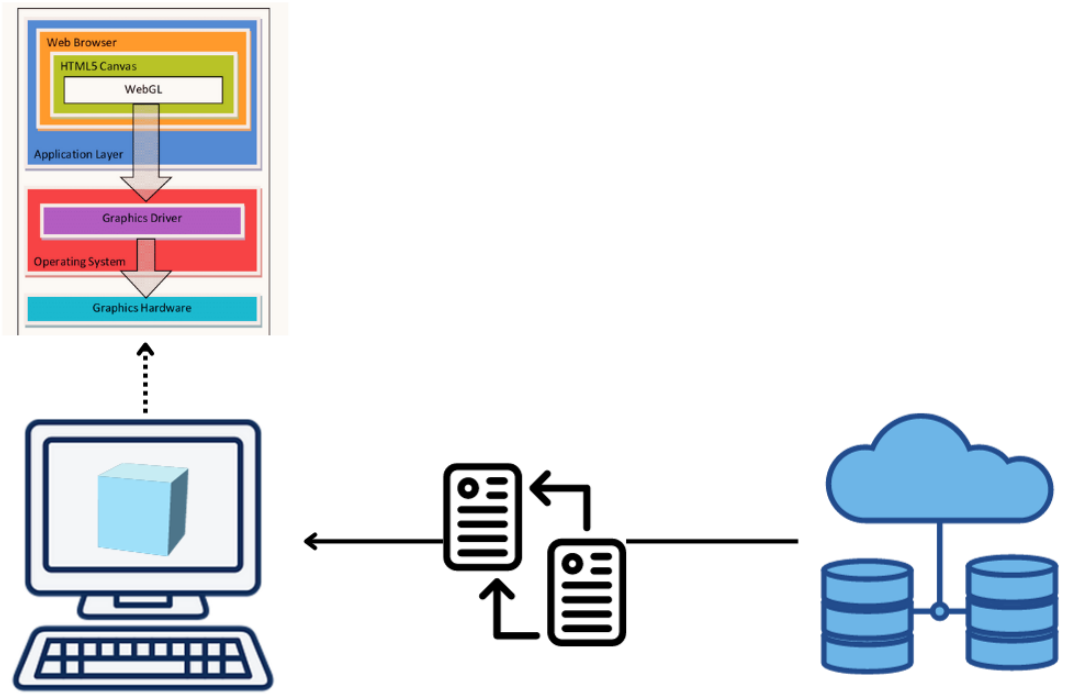


Figure 4.2: Local rendering representation

A potential remedy for this issue involves storing the geometries in the browser's local storage when they are initially uploaded and maintaining them there until a session concludes. However, a significant challenge arises when dealing with the size of these geometries. For applications that deal with CFD simulations and demand a high level of detail, users often upload geometries of considerable complexity, resulting in files that can weigh in the order of megabytes, or even gigabytes. This implies a substantial constraint as browser storage capacities are limited.
Moreover, retrieving these extensive geometries each time they need to be visualized incurs a considerable computational and economical cost. This is due to the substantial volume of data transferred between the server and client, especially when multiplied by the potential number of concurrent clients. Additionally, many web browsers struggle with the rendering of large 3D models due to the substantial memory requirements and the overhead involved in utilizing the user's graphic

hardware. Consequently, the user experience may be compromised, leading to reduced frame rates and slow performance on the user's machines.

On the contrary, given the impracticality of burdening the user with the entire computational load, an alternative approach has been contemplated. This approach involves streaming a continuous series of rendered images through a server endowed with superior computational capabilities. This technique is commonly referred to as *Remote Rendering*, as it relocates all processing activities away from the client's computer. Consequently, the client side role is reduced to merely displaying the stream of images within a dedicated canvas. In effect, this eliminates the necessity for rendering processing on the client side and prevents the need to retain large data in browser memory. Figure 4.3 illustrates a representation of a remote rendering process.

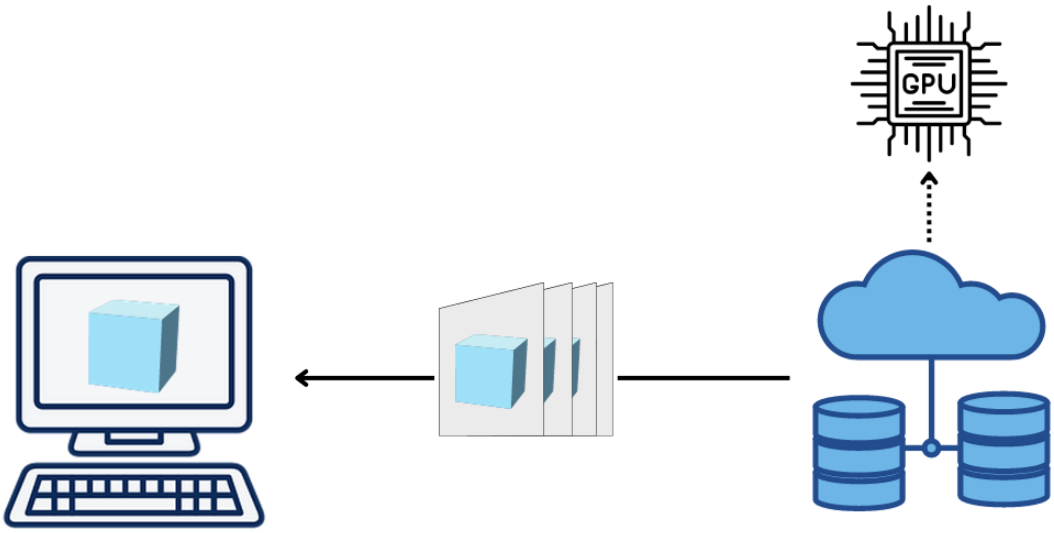


Figure 4.3: Remote rendering representation

This approach effectively addresses the majority of issues associated with local rendering. By offloading the computational workload onto a dedicated server equipped with higher capabilities, it is ensured optimal rendering of geometries of any size. Furthermore, since the user would have already provided the geometry, the expense of transferring it is a one-time occurrence. Nonetheless, it is crucial to consider that the primary purpose of this application is to conduct simulations and heat-sink analyses for CHT problems. Therefore, the cost associated with implementing an interactive preview feature should not surpass that of the core functionality of the software. This is particularly relevant when considering the

transmission of high-fidelity images of a geometry rendered on a server located remotely from the user's machine, as it involves substantial data transfer, which can vary depending on the duration and level of user interaction with the interactive preview.

Optimad has concerns regarding this approach, primarily due to two key factors. First, the frame rate depends on the smoothness of the image stream, a factor inherently reliant on user network capabilities and the variability in image sizes, it is possible that this approach may potentially undermine the overall user experience. Furthermore, the second concern pertains to the economic implications of data transfer in a multi-user environment. Optimad is interested in avoiding allocating a substantial portion of the project budget to this functionality, as it does not constitute one of the primary core aspects of the project.

In summary, there is a need to develop an interactive preview system for geometries of large size that offers seamless user-friendliness while concurrently mitigating the escalation of both economic and computational costs.

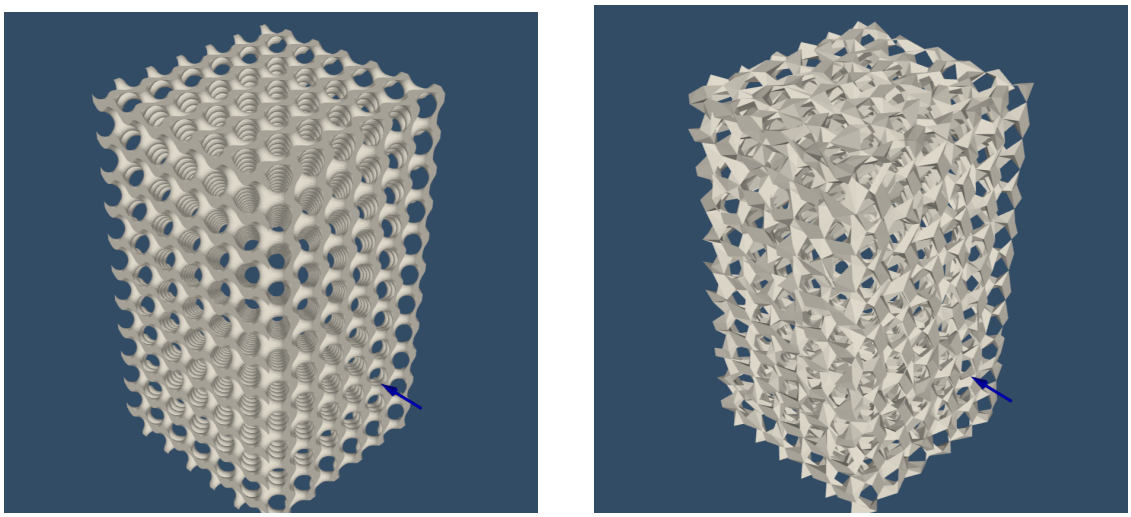
4.3 Proposed solution

Considering the advantages and disadvantages outlined earlier, an alternative solution is being proposed within the context of this thesis. The objective is to evaluate the efficacy of this proposed solution in addressing the challenges delineated by the two conventional approaches. This new approach seeks to combine the merits of the preceding solutions, thereby leveraging the strengths of both while minimizing their respective weaknesses.

This approach for this thesis will be called *Hybrid rendering* since both client and server will participate in the geometry rendering. This approach consists of rendering a lighter version of large geometry with a process called *Decimation*. Decimation is the process of reducing the number of points in a data-set while trying to preserve its overall shape and important features. It is particularly useful when dealing with large data-set, in this case, large geometries, as it can significantly reduce the memory and computational resources required for visualization [42].
This version of the geometry is generated by the server after the completion of the initial upload. Subsequently, it is retrieved by the client, enabling the browser to render it efficiently, thereby saving substantial computing resources. It is assured that this optimized version will consistently maintain a constant file size of 1 MB, achieved by minimizing data points while preserving the original geometric integrity.

However, this new version may not show a high-fidelity depiction of the geometry,

as it has undergone significant point reduction. Figure 4.4 illustrates the difference between the original geometry and its decimated form. The primary function of a decimated version is to alleviate operations regarding interaction, such as rotation and zooming on the client side. Subsequently, when a user interacts with the model and navigates to the specific portion of the geometry they wish to inspect, a high-fidelity rendering process is initiated on the server side. This mechanism closely resembles remote rendering, but with the distinction that instead of streaming a multitude of images, a single image is transferred each time a user ceases their interaction with the geometry. Figure 4.5 provides a visual representation of the hybrid rendering process.



(a) Original geometry (b) Decimated geometry

Figure 4.4: Decimated geometry example

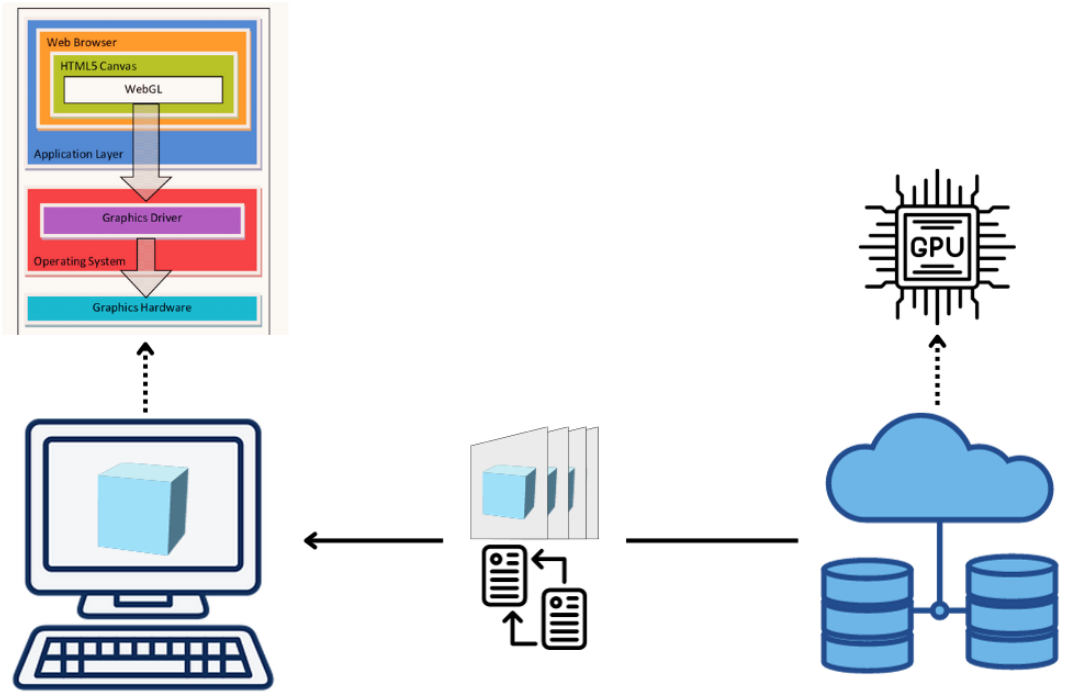


Figure 4.5: Hybrid rendering representation

In this way, it is expected that the primary limitations inherent in the first two approaches can be effectively mitigated. This is owing to the fact that there would be a notable reduction in the computational load imposed on the client side, all the while ensuring that user experience remains unaffected. Simultaneously, the volume of data transmitted from the server to the client would be substantially diminished, as only a single image would be streamed each time a user ceases interaction, as opposed to during interaction. Consequently, this approach would yield a reduction in data transfer costs, thereby aligning with the significance of this functionality and its corresponding budget constraints.

Chapter 5

Application development

5.1 Introduction

This section will comprehensively present the software engineering processes necessary for building the application. These processes encompass requirement elicitation, analysis, actor and use case design, as well as application architecture. Certain parts of this chapter will present the current overall system described, especially in sections 5.3, 5.4, 5.5, 5.6 and 5.7, however, the focus of this chapter lies principally on how the proposed solution is developed and integrated with the current system.

5.2 Requirements elicitation

The development initiative was initiated with the formulation of mockups and conceptual ideas. Optimad, a specialist in numerical analysis and CFD simulations, sought to transform one of their existing applications into a web-based interface. This transformation aimed to enhance user-friendliness by incorporating an appealing and intuitive graphical user interface.
ESTECO, closely affiliated with Optimad, joined the development efforts due to their wealth of experience and expertise in similar undertakings. As a result, the primary source of the project's requirements stems from Optimad itself. As highlighted in the preceding chapter, the initial phase involved the creation, discussion, and organization of requirements into user stories using an iterative approach. This dynamic process led to the emergence of new user stories during meetings, while implementation of *Epics* was still under development.
Within the scope of this thesis, the following tables will exclusively outline the descriptions of requirements tied to the functionalities of the proposed solution (hybrid rendering). Additionally, requirements pertaining to the conventional solution (involving local and remote rendering) will be included for comparative analysis.
In Tables 5.1 to 5.7 are described the functional requirements.

Requirement	FR01	**Requirement type**	Functional
Description	Upload a geometry file that can be used as a heat-sink or a source body.		
Reason	Based on the geometries uploaded, the user can know which geometries are going to be used for simulations.		
Acceptance criteria	The uploaded file is only STL extension. The uploading should show the progress. The geometry will be lost if the server goes down during the uploading. A geometry identifier must be generated after being uploaded.		
Priority	Medium		

Table 5.1: Functional requirement 01.

Requirement	FR02	**Requirement type**	Functional
Description	Enable client-side rendering of uploaded geometries for interacting preview		
Reason	Visualize geometry to make sure that the geometry corresponds to the file that was uploaded previously by the user		
Acceptance criteria	Client-side rendering processed by user's machine. The front-end must give the possibility to rotate the camera view and zoom in/out. The geometry visualized must be the expected that was uploaded.		
Priority	Medium		

Table 5.2: Functional requirement 02.

Requirement	FR03	**Requirement type**	Functional
Description	Enable server-side rendering of uploaded geometries for interacting preview		
Reason	Visualize geometry to make sure that the geometry corresponds to the file that was uploaded previously by the user. Visualize geometries whose file size is too big to be processed by user's machine		
Acceptance criteria	Server-side rendering processed by a remote server. The front-end must give the possibility to rotate the camera view and zoom in/out. The geometry visualized must be the expected that was uploaded.		
Priority	Medium		

Table 5.3: Functional requirement 03.

Requirement	FR04	**Requirement type**	Functional
Description	Generate a decimated/lighter version from an uploaded geometry.		
Reason	A lighter version of big geometries is needed to be displayed on the user's machine.		
Acceptance criteria	The size of the decimated geometry must have an overall size of 1MB. Should be generated on the back-end with an existing geometry ID.		
Priority	Medium		

Table 5.4: Functional requirement 04.

Requirement	FR05	**Requirement type**	Functional
Description	Display geometry decimation status when is being generating even when the user navigates to another component.		
Reason	The visualization of the geometry will not block the user from using other functionalities. If the generation takes long, the user should see the status at any moment.		
Acceptance criteria	The status must be tracked. The status should be displayed once the geometry has finished to be uploaded. Changing the page must retrieve the status of the decimation generation		
Priority	Medium		

Table 5.5: Functional requirement 05.

Requirement	FR06	**Requirement type**	Functional
Description	Retrieve to user the decimated geometry when is generated.		
Reason	An existing decimated geometry needs to be retrieved to the user in order to display it locally.		
Acceptance criteria	The resulting decimated geometry is delivered as a structure of vertices or triangles persisted in user's memory. The decimated geometry should be delivered when a decimated has finished to be created.		
Priority	Medium		

Table 5.6: Functional requirement 06.

Requirement	FR07	**Requirement type**	Functional
Description	Enable automatic switch client-side and server-side rendering of uploaded geometries for interacting preview.		
Reason	Visualize interactively a geometry to make sure that the geometry corresponds to the file that was uploaded previously by using a lighter version of geometry for the client side and the delivery of the full geometry processed by a server		
Acceptance criteria	Client-side rendering of a decimated geometry when the user interacts with it (zoom in/out, camera rotation). Server-side rendering of the complete geometry when the user stops interacting with it. The transition between rendering should be automatic.		
Priority	Medium		

Table 5.7: Functional requirement 07.

All functional requirements must be implemented while also satisfying non-functional requirements, ensuring the application's performance, usability, scalability, availability, security and licensing, and compatibility. In Table 5.8 the non-functional requirements are described.

Requirement	Type	Description
NFR01	Compatibility	The application should be compatible with any device equipped with a modern web browser, including both desktop PCs and laptops.
NFR02	Performance	The platform should be able to handle visualizations for large and complex geometries efficiently, with minimal degradation in performance.
NFR03	Performance	The application must implement efficient storage management to accommodate the storage requirements of geometries and intermediate data generated during the visualization process.
NFR04	Scalability	The system should scale seamlessly to accommodate a growing number of users requests without compromising response times.
NFR05	Security	User-uploaded geometries and simulation data must be securely stored and transmitted to prevent unauthorized access or data breaches.
NFR06	Cost Efficiency	The application should be designed to efficiently utilize and optimize resources, to avoid incurring excessive infrastructure costs.

Table 5.8: Non-functional requirements.

Requirement	Type	Description
NFR07	Security	Ensure compliance with relevant data protection regulations, such as GDPR.
NFR08	Availability	The platform should maintain high availability, ensuring that visualizations can be run at any time without extended downtime.
NFR09	Usability	The user interface should be intuitive and user-friendly, allowing users to easily upload geometries, configure simulation parameters, and interpret results.
NFR10	Licensing	Ensure compliance with software licensing agreements for any third-party libraries or tools utilized in the simulation process.
NFR11	Performance	95% of application requests must be served within 2 seconds to meet user experience expectations

Table 5.8: Non-functional requirements.

5.3 Actors

After establishing and collecting both the functional and non-functional requirements, it becomes crucial to address additional facets of the project needed for its development. These facets pertain to the entities engaged within the application, their interfaces, and the broader system context. Specifically, the actors and their descriptions are outlined below, providing insight into the key entities that will interact with the application. The system's actors are delineated in Table 5.9.
The intended users comprise individuals with expertise limited to 3D graphic design, lacking familiarity with CFD. They seek to assess and simulate uploaded geometries to determine the successful completion of CFD tests for their models.

Actor	Description
User	Uses the application to upload geometries and make CFD simulations.
Administrator	Manages and oversees various aspects of the application to ensure its proper functioning, security, and efficient use.

Table 5.9: Actors

Figure 5.1 represents the context of the system that reports how the actors are supposed to interact with the system.

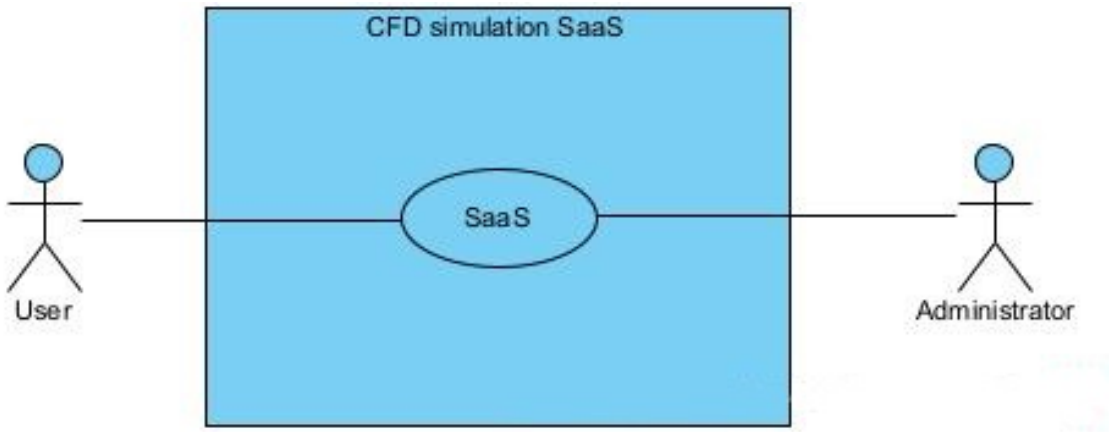


Figure 5.1: Context diagram

The actor in the role of *Administrator*, within this context, serves as a maintainer for ESTECO-Optimad. They possess an enhanced authorization to manipulate system features, enabling a quicker response in case of any internal malfunctions. For normal scenarios, this role will not participate actively with the functionalities of the application, so *User* will not need directly *Administrator* to interact with the system.

5.4 Interfaces

After mentioning the system actors, the subsequent step involves establishing the mechanisms through which these actors can interact with the application. These mechanisms can be categorized into two distinct types: the logical interface and the physical interface. The logical interface delineates the conceptual interaction between the entity and the application, including various technologies and interfaces. On the other hand, the physical interface explains the tangible means by which this interaction occurs, encompassing any pertinent hardware components. Table 5.10 provides an overview of the essential interfaces for the entities to interact with the application.

Actor	Logical Interface	Physical Interface
User	GUI	Network, monitor, mouse, keyboard
Administrator	GUI, company VPN	Network, monitor, mouse, keyboard

Table 5.10: Interfaces

Both actors will interact with the application through a graphical user interface. The distinction lies in the fact that only the *Administrator* can get access through a VPN in order to use privileged functionalities; Considering that the application operates as a web-based system, a computer with Internet access is necessary for its use. Furthermore, the application has been optimized for utilization on smartphone browsers.

5.5 Use cases

Figure 5.2 presents a detailed diagram illustrating the interactions between actors and the system, showcasing how their various requirements are mapped to specific use cases and the associated behaviors. Each use case provides insights into certain pre-conditions, post-conditions, and scenarios. It is important to note that this diagram encompasses all the use cases that will be present in the final application, but only those pertinent to the thesis will be elaborated upon.

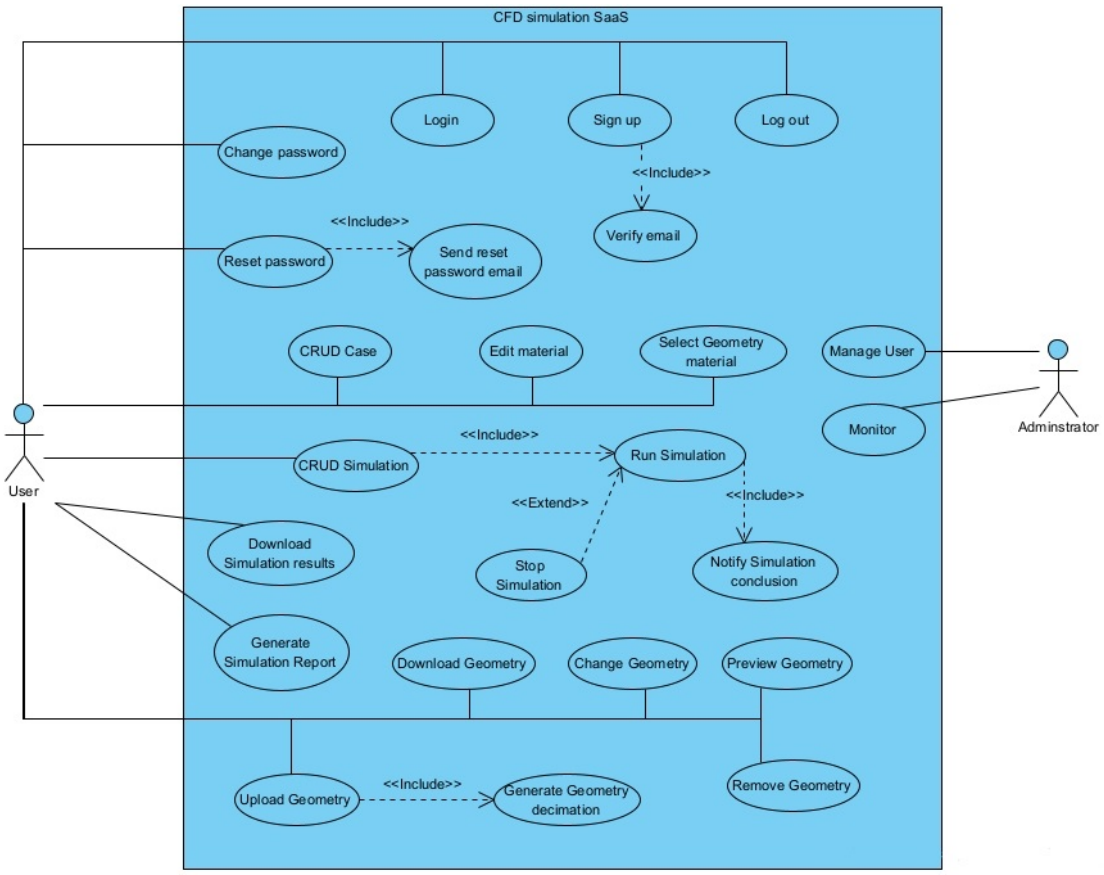


Figure 5.2: Use Case Diagram

In Tables 5.11 to 5.13 are described in detail only the use cases relevant to the

thesis. The use case *Preview Geometry* will show a scenario for each rendering mode used in the thesis.

Name	Upload Geometry
ID	UC1
Pre-conditions	User must be logged in and have a Case created.
Post-conditions	The geometry that the user uploaded is persisted with a new ID.
Nominal scenario	1. User clicks on button *Add device*. 2. System opens file chooser. 3. User chooses the STL file of the desired geometry. 4. System divides the file into chunks and sends it to the server. 5. System shows a progress bar until is complete 6. System builds the file from the chunks, persists it and generates a unique identifier. 7. System initiates Use Case 2.
Non-nominal scenario	1. User clicks on button *Add device*. 2. System opens file chooser. 3. User chooses the STL file of the desired geometry. 4. System divides the file into chunks and sends it to the server, but the server is overloaded or there are connection problems. 5. System shows a failure message and displays a *Retry* button.

Table 5.11: Use Case 1.

Name	Generate Geometry decimation
ID	UC2
Pre-conditions	User must be logged, have a Case created and a geometry uploaded successfully.
Post-conditions	The geometry that the user uploaded has now a decimated version that is sent back to the browser.
Nominal scenario	1. System checks if the geometry file has a size higher than 1MB. 2. System decimates the number of triangles of the STL file into a number that fits with a file size of 1MB. 3. System shows a progress bar while decimating until is complete 4. System retrieves the decimated file to the user. 5. System enables *View* button.
Non-nominal scenario	1. System checks if the geometry file has size higher than 1MB. 2. System decimates the number of triangles of the STL file into a number that fits with a file size of 1MB. 3. System takes more than it should or there are connection problems. 4. System shows an error message. 5. System keeps disable *View* button.

Table 5.12: Use Case 2.

Name	Preview Geometry
ID	UC3
Pre-conditions	User must be logged, have a Case created and a geometry uploaded successfully with a decimated version in memory if needed
Post-conditions	The geometry is displayed to the user this one can interact with it.
Nominal scenario - Local rendering	1. System clicks on *View* button. 2. System shows in a canvas the full geometry uploaded in memory. 3. User interacts with it (zoom in/out, move). 4. System renders with user resources the geometry.
Nominal scenario - Remote rendering	1. System clicks on *View* button. 2. System sends from the browser the geometry ID to be rendered. 3. System retrieves the geometry file from ID and displays it into a canvas remotely. 4. User interacts with it (zoom in/out, move). 5. System retrieves camera position from the browser and from these, renders and sends images of the geometry back to it.

Table 5.13: Use Case 3.

Name	Preview Geometry
ID	UC3
Nominal scenario - Hybrid rendering	1. System clicks on *View* button. 2. System sends from the browser the geometry ID to be rendered. 3. System retrieves the geometry file from ID and displays it into a canvas remotely. 4. User interacts with it (zoom in/out, move). 5. System renders with user resources the geometry decimated. 6. User stops interacting with the geometry. 7. System retrieves camera position from the browser and from these, renders and sends images of the geometry back to it.
Non-nominal scenario	1. System clicks on *View* button. 2. System is overloaded or is not able to display the geometry. 3. System displays an error message.

Table 5.13: Use Case 3.

To take an insight into their interconnection, now is presented a mapping between functional requirements and use cases in Table 5.14.

	UC1	UC2	UC3
FR01	✓		
FR02			✓
FR03			✓
FR04		✓	
FR05		✓	
FR06		✓	
FR07			✓

Table 5.14: Functional requirements mapped into use cases.

5.6 Solution design

It is now possible to start designing the key components required for building the application, incorporating its domain model and the execution of various scenarios, based on established use cases and their respective functional requirements.
In Figure 5.3, it is presented the class diagram of the application, illustrating the entities that abstractly represent the attributes and relationships of objects needed within the program. It is important to note that, given the nature of the project and methodology, these entities are only a subset of the potential entities that may emerge during its development. As of the writing of this thesis, these are the entities identified and that are actively developing during the sprints.
The core of the business logic lies in allowing *Users* to create portfolios of simulations, referred to as *Cases*. Within these Cases, users can upload two types of *Geometry*: *Device* and *Source*. These components are essential for conducting CFD analyses within the context of a Compact Heat Exchanger (CHT) used for heat sinks. The *Source* represents the electrical or electronic equipment generating heat, and its characteristics vary depending on the *Material* used and its attributes. On the other hand, the *Device* signifies the contact point with the surrounding environment where heat is released or dissipated [43].
These geometries enable the possibility to run multiple simulations with different configurations. Since some simulations can be time-consuming, it is provided a progress status indicator to keep users informed about the progress of each simulation.

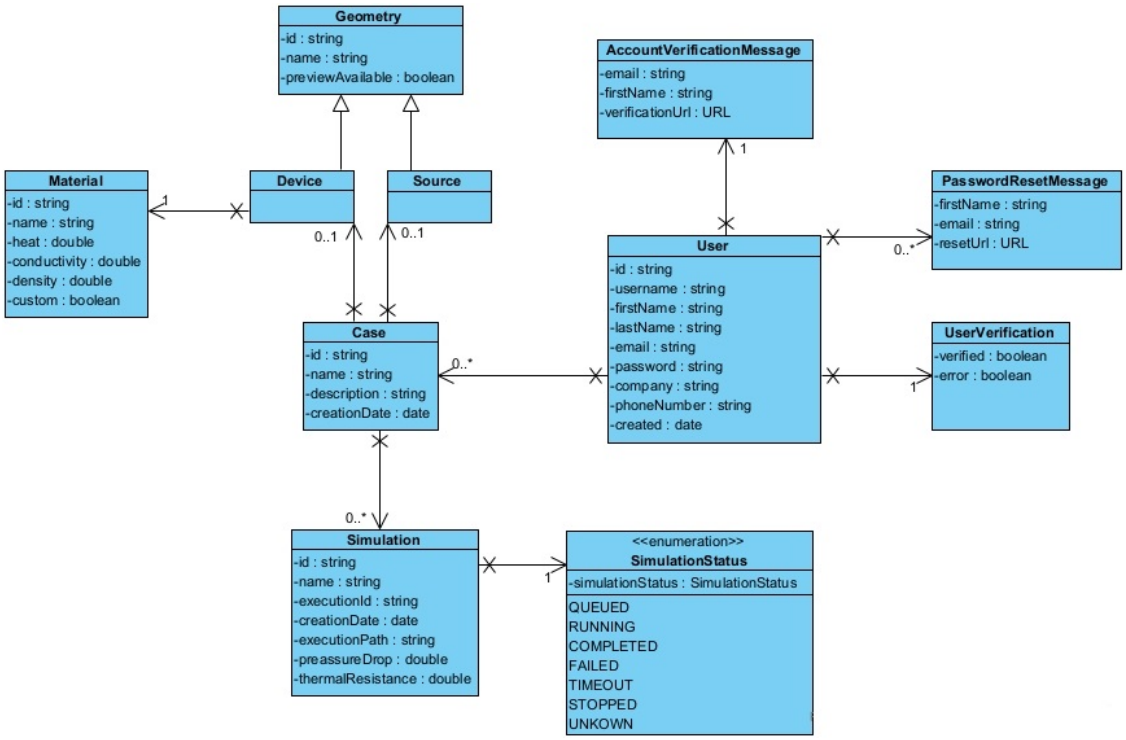


Figure 5.3: Class diagram.

Subsequently, the upcoming subsections will provide an in-depth exploration of how entities are interconnected and how they interact to fulfill the use cases outlined in Section 5.5. It employs Entity-Boundary-Controller (EBC) diagrams to illustrate the flow and connections between users and the responsible components for executing the desired functionalities. The sequence diagrams delineate the steps and timeline governing the communication among components, ultimately leading to the successful execution of desired functionalities.

5.6.1 Use Case Upload Geometry

Figure 5.4 shows the interaction between the boundaries and controllers to effectuate the uploading of a geometry using the software design pattern Model-View-Controller (MVC). The geometry is saved in memory during the uploading on the front-end side, however, once it is uploaded, the binary file is saved in an Amazon Elastic File System (EFS) and its information is persisted in a database.

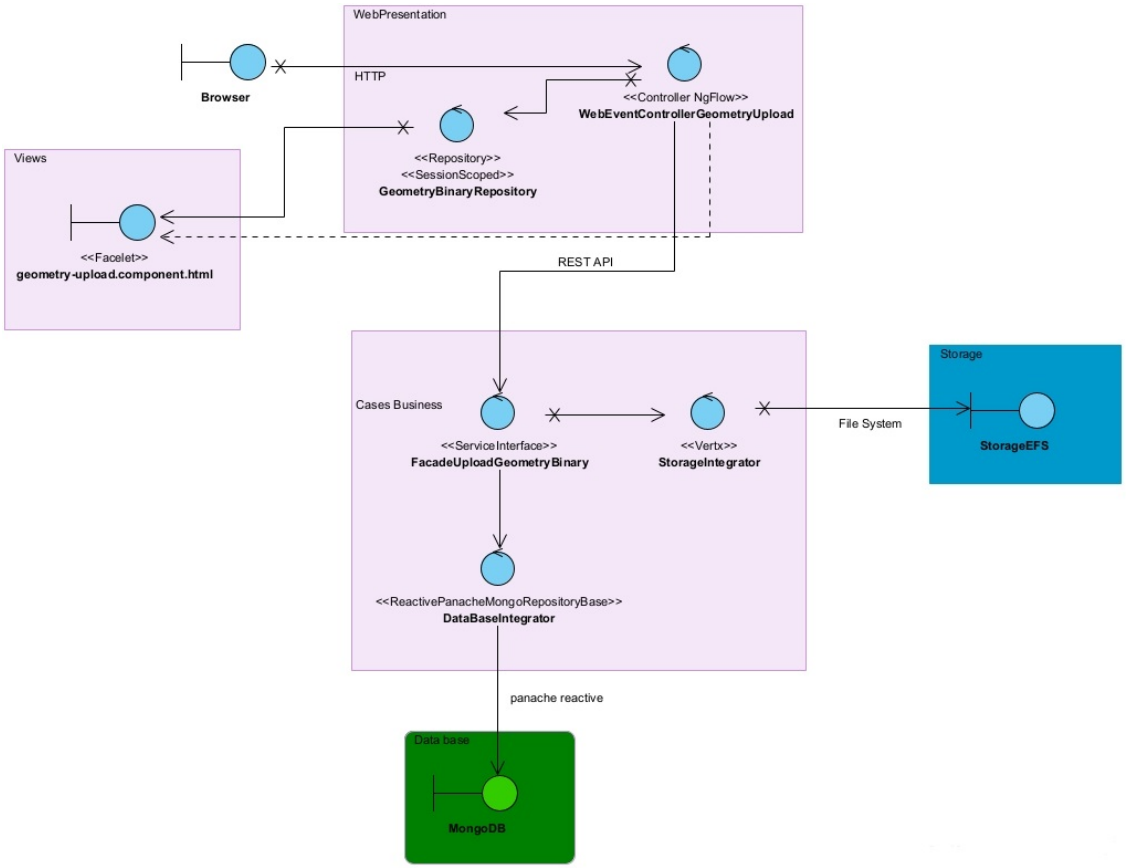


Figure 5.4: UC1 Upload Geometry EBC diagram

In Figures 5.5 and 5.6 are presented the sequence diagram of the front and back-end side respectively. The file is divided in chunks in order to facilitate the transfer and data load towards the back-end and once is totally uploaded decides whether to begin with the decimation or not. Whenever an error is presented is informed directly to the user. In the same way, the back-end reconstructs each chunk, generates a unique identifier and saves the information properly.

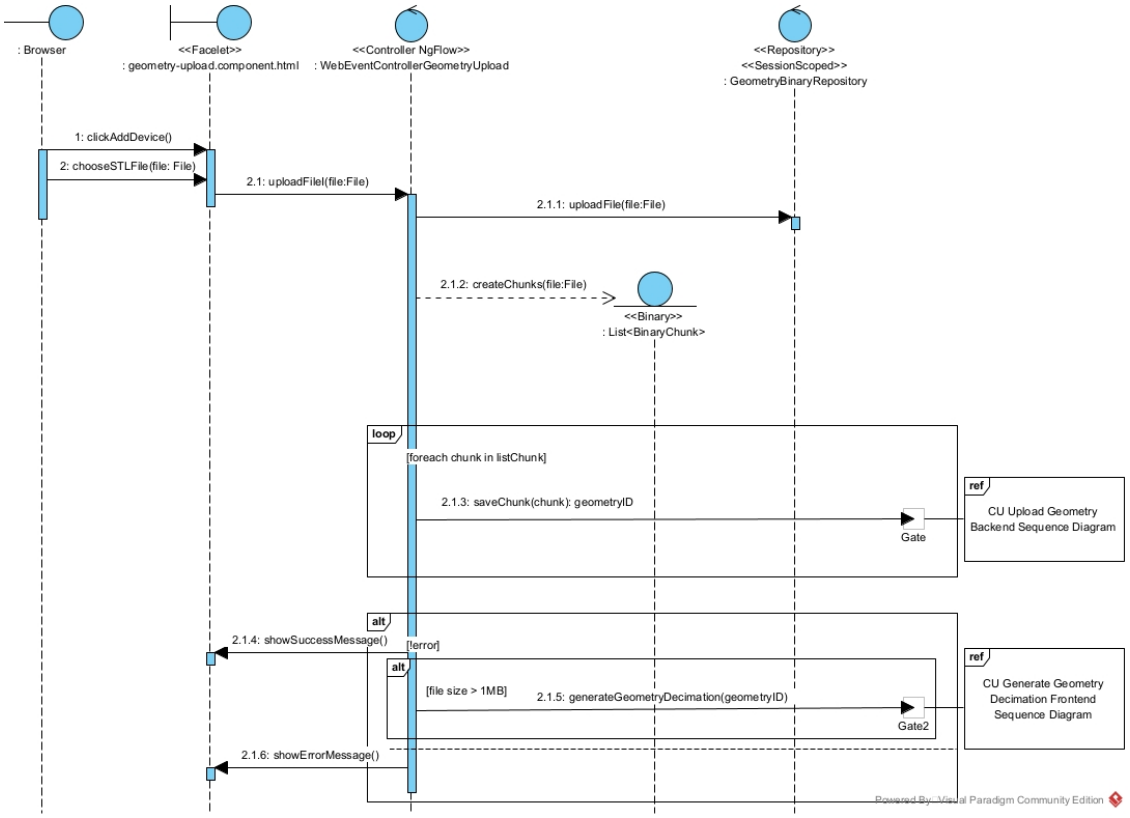


Figure 5.5: UC1 Upload Geometry Front-end sequence diagram

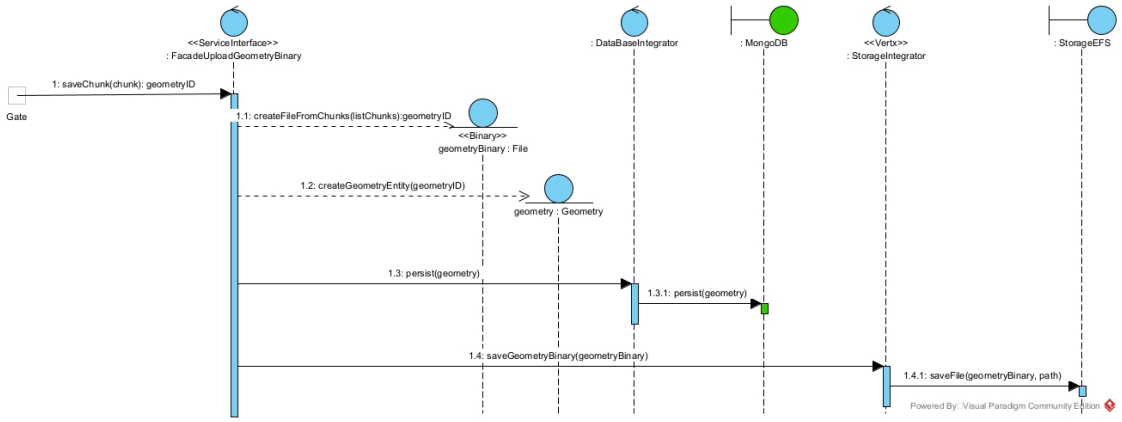


Figure 5.6: UC1 Upload Geometry Back-end sequence diagram

5.6.2 Use Case Generate Geometry Decimation

As explained in Section 4.3, the decimation process is necessary to deliver to the user a lighter version of the geometry, keeping the most important points to preserve its original shape.

In Figure 5.7 shows how the boundaries and controllers interact to generate the decimation.

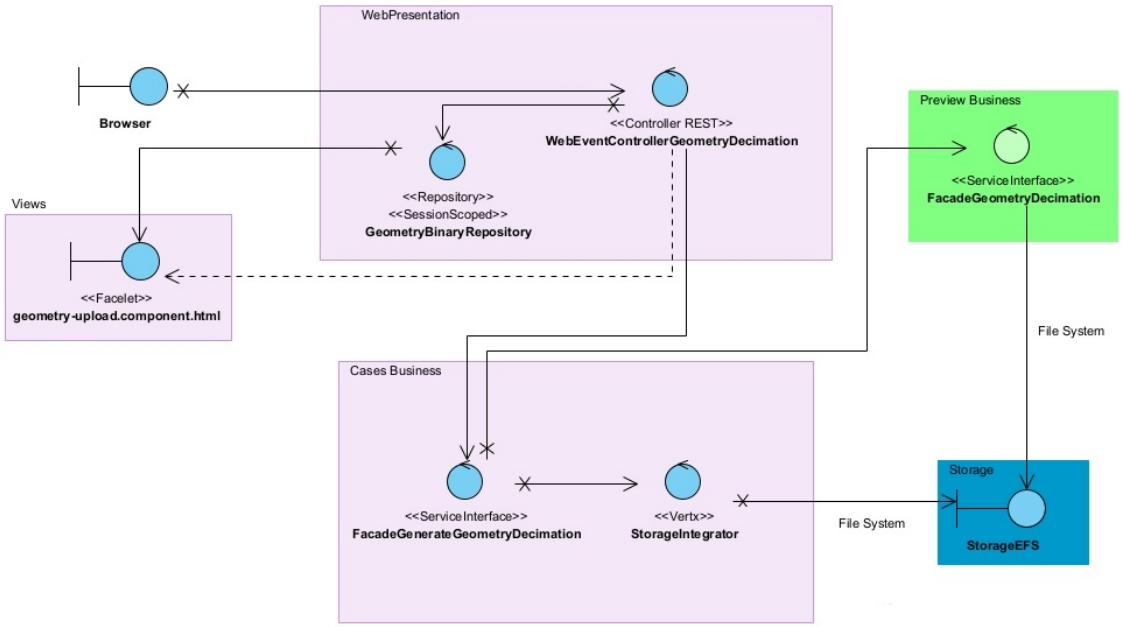


Figure 5.7: UC2 Generate Geometry Decimation EBC diagram

In Figures 5.8 and 5.9 are presented the communication between elements for decimating. The process starts when a large geometry is encountered, since the geometry is already saved in storage, it is enough to ask for the decimation using a unique identifier generated in the previous use case. Since decimating is a time-consuming process, the front-end uses a polling approach to verify if the decimated version of the geometry has finished to be created, in a positive case, it is asked for the binary file and saved in the user's browser memory. In the back-end, a service is in charge of asking for the decimation and continuously checks if the decimated version is located in a particular path in the shared file system.

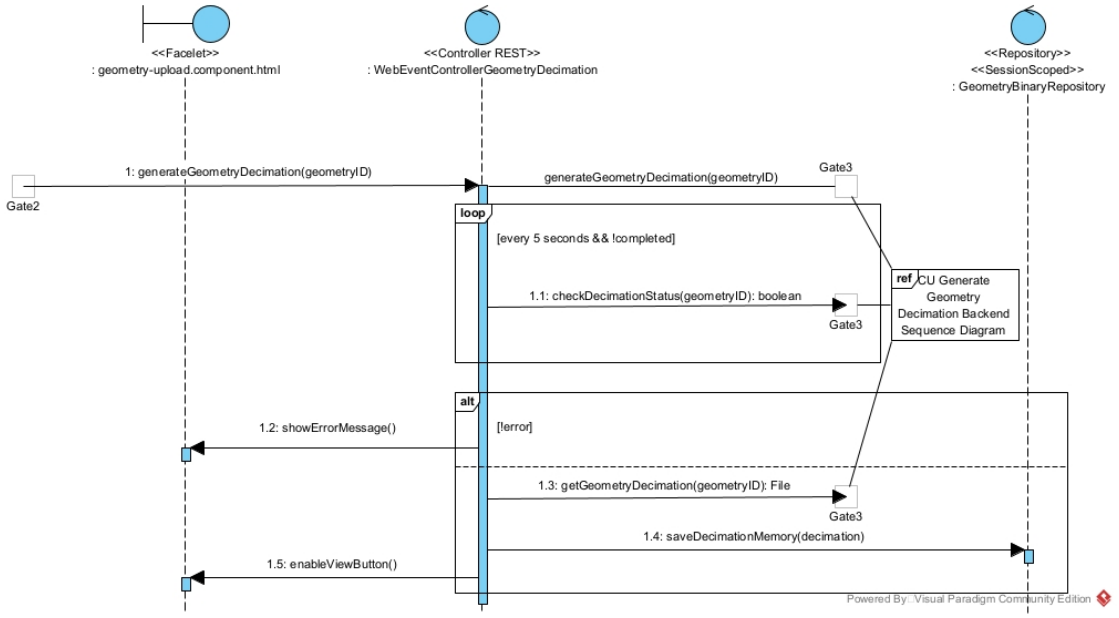


Figure 5.8: UC2 Generate Geometry Decimation Front-end sequence diagram

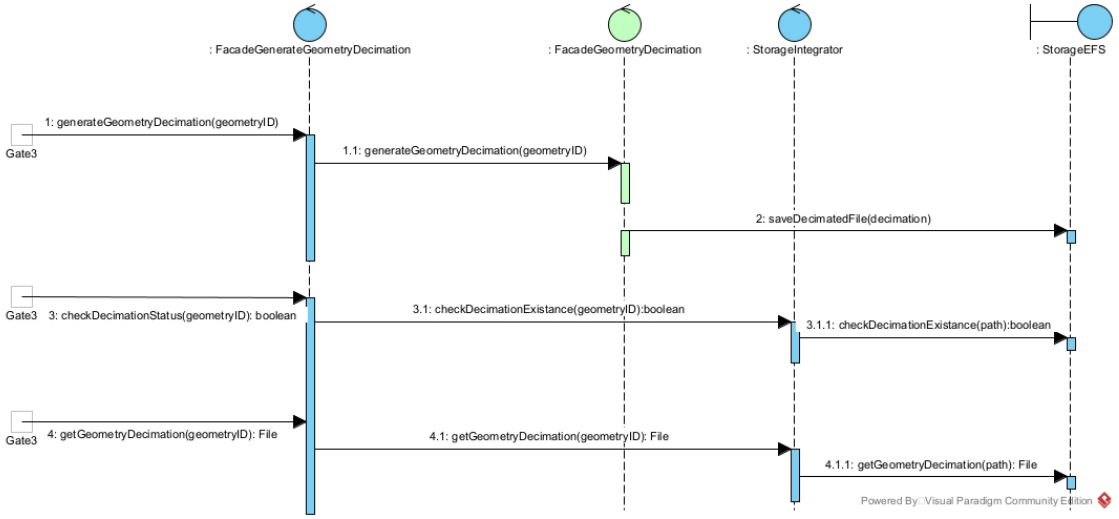


Figure 5.9: UC2 Generate Geometry Decimation Back-end sequence diagram

5.6.3 Use Case Preview Geometry

In Figure 5.10 is presented the elements involved for displaying the hybrid rendered interactive preview. This scheme shows the necessity of facades used to retrieve the location of the path, since *Cases Business* has the logic to provide the path of a geometry from its ID. In full remote rendering the scheme will be the same but the sequence will be different, and in a full local rendering is not needed those facades,

since everything is done in the user's browser.

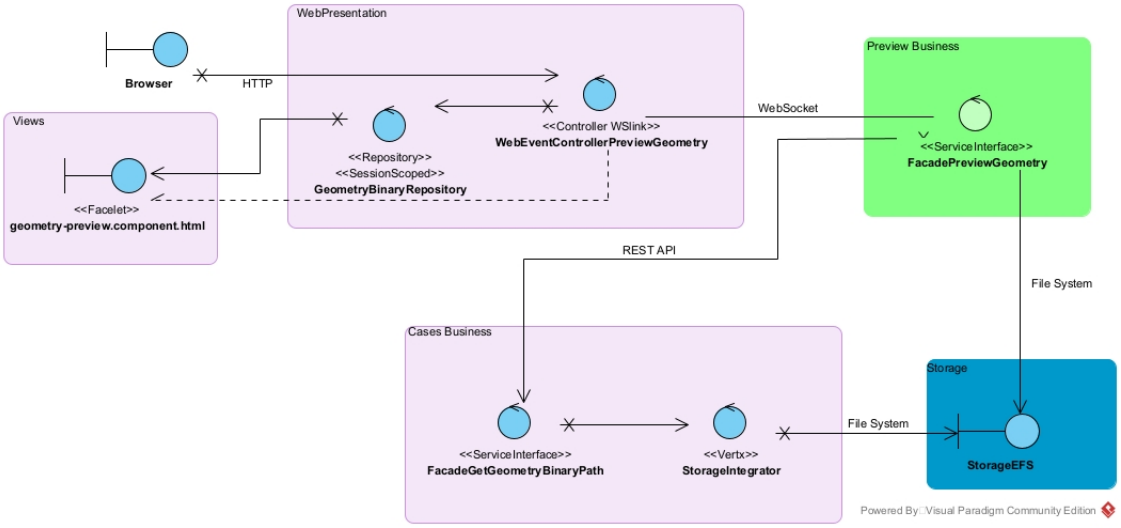


Figure 5.10: UC2 Preview Geometry EBC diagram

In Figures 5.8 and 5.9 are shown only the sequence of procedures and communications to perform a hybrid rendering. The process starts by asking for WebSocket connection between the front-end with the service in charge of doing the rendering remotely and sending it the geometry ID that wants to visualize. Once the connection is established, the user can start interacting with the canvas, visualizing the decimated version that was retrieved in the previous use case and once the user stops interacting, the front-end sends the camera position of the last frame and it gets back a high fidelity image of the geometry rendered in the back-end. In the same way, the back-end asks for the path of the geometry and loads it in memory to render it. Once it is uploaded in memory, it listens through the WebSocket for camera position coordinates in order to synchronize both views and deliver a seamless interaction.

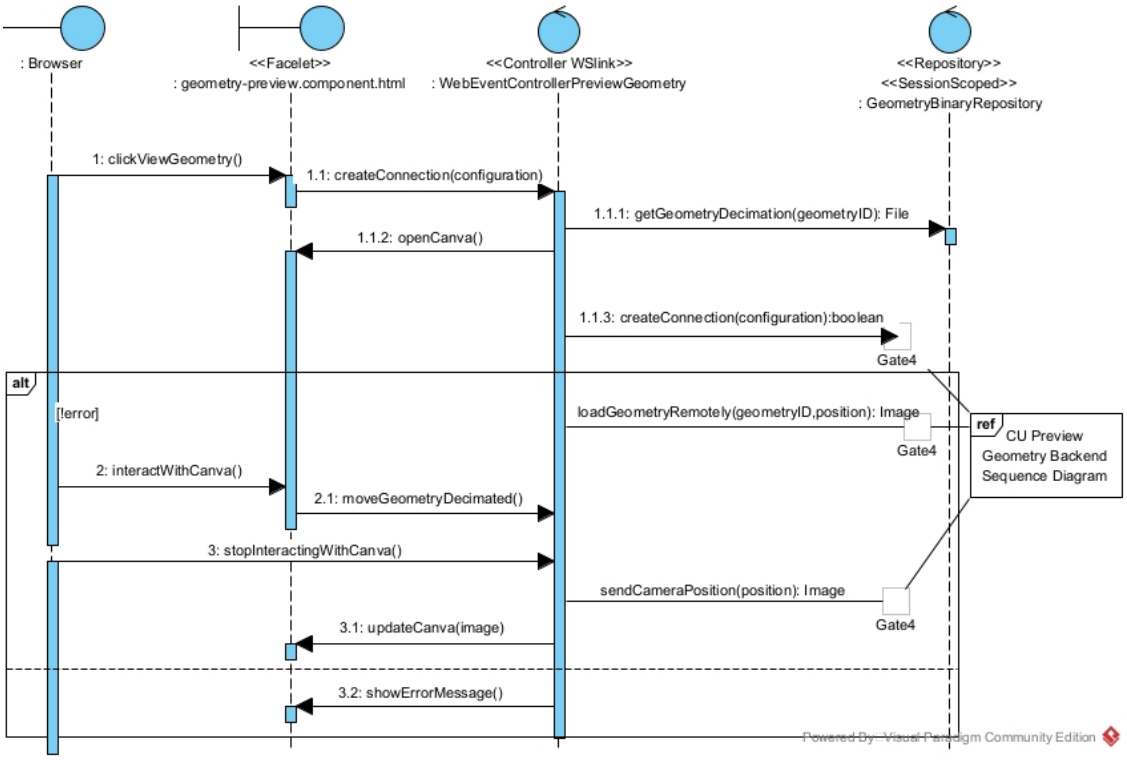


Figure 5.11: UC3 Preview Geometry Front-end sequence diagram

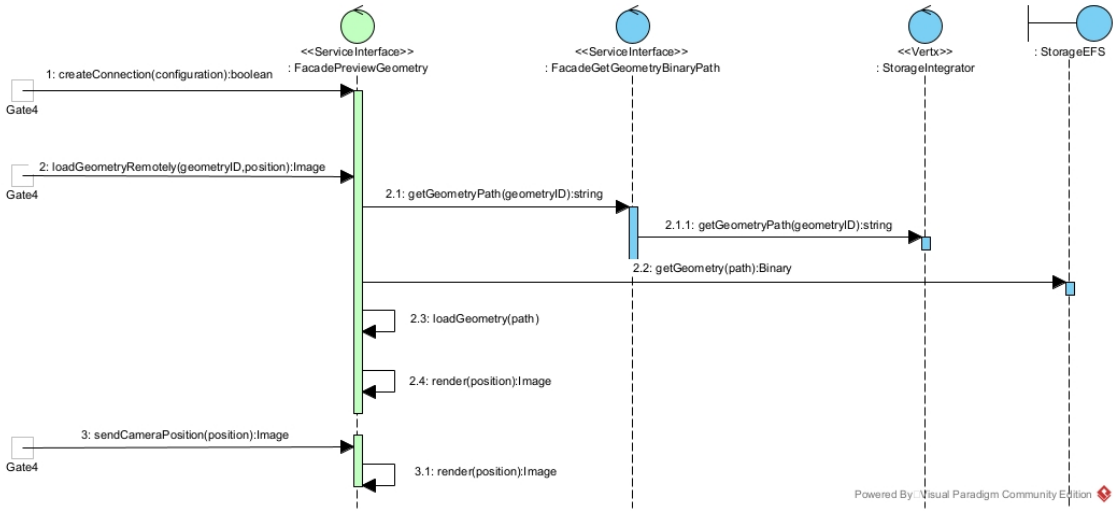


Figure 5.12: UC3 Preview Geometry Back-end sequence diagram

5.7 Application Architecture

In this section is presented the whole architecture necessary to build and deploy the entire system. Figure 5.13 shows the application deployment diagram which complements the technologies and frameworks mentioned in Section 3.3 and shows how the software components are distributed in physical nodes and how they interact with each other.
The front-end will be deployed using Amazon Content Delivery Network (CDN) which is useful for distributing static and dynamic content in a fast and reliable way. The *SideCar* is presented in this component but it is assumed that it will be present along with the application, so every component knows which entities they are working with. Along with *Angular components* needed for proper front-end functioning, there are external libraries such as *VTK.js* (selected in Section 2.4), a ESTECOS's custom GUI design system called SOUL, and *Flowjs* which is in charge of partitioning the STL files in chunks for delivering to back-end.
Some of the back-end services will be hosted on Elastic Kubernetes Service (EKS), an Amazon Web Service that offers Kubernetes cluster provisioning and management. In contrast, the high-resource-demanding HPC processes will be entirely managed through AWS Parallel Cluster. The deployment will be leaded by Marco Cisternino who is the architecture specialist from Optimad.

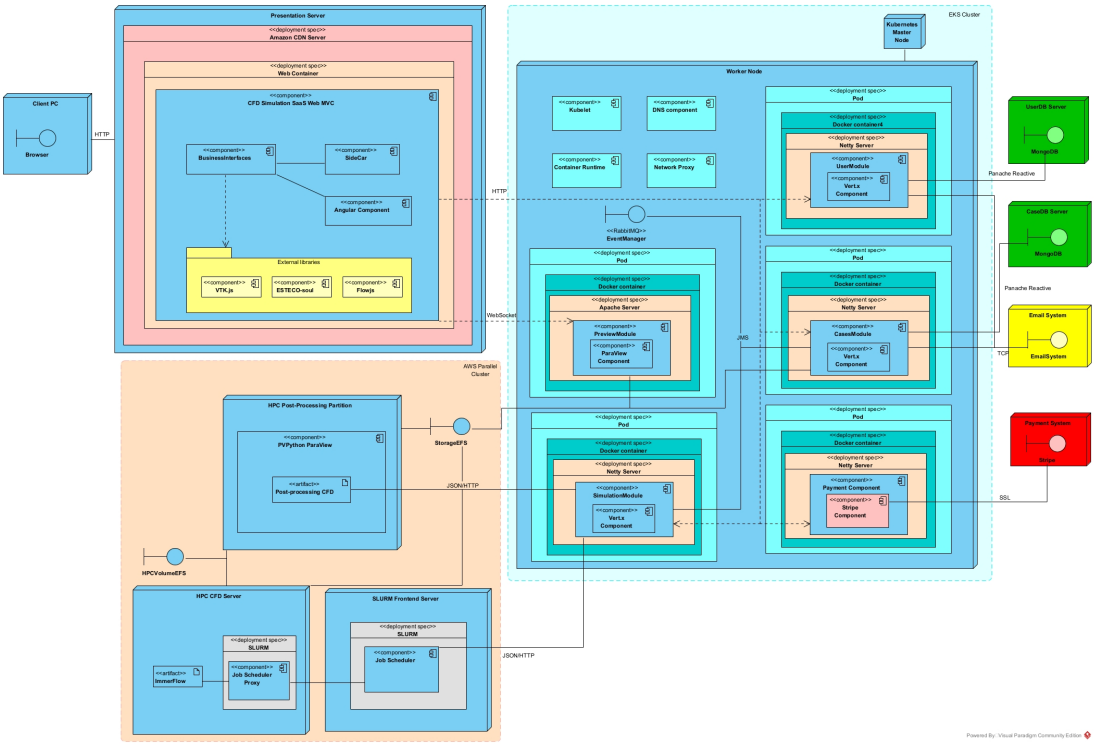


Figure 5.13: Deployment diagram

Each service will operate within Docker containers deployed in *Pods*, which will dynamically scale based on demand. Some of these services will utilize Vert.x and Netty, which integrate seamlessly with Quarkus. In the specific case of the rendering service for remote access, it will be deployed using an Apache server to facilitate multi-user visualization. This approach is aligned with the recommendation from ParaViewWeb developers and is consistent with the choice in Section 2.4.
RabbitMQ seamlessly integrates with Kubernetes and has been chosen as the event manager whenever a new simulation state update occurs. These updates are accompanied by an email service that notifies users about the simulation state. Additionally, this external system is utilized for user management. There will be two databases: one exclusively for this application and another to manage user data. The latter will not be exclusive, as future Optimad applications may share the same user information. To handle payments securely, *Stripe* will be employed as an external platform responsible for processing all user payments.
Ultimately, the simulations will be executed on an HPC cluster, requiring a job scheduler capable of managing simulation queuing and notifications. These simulations will be conducted using Optimad's software, *Immerflow*, and post-processed through a custom *PVPython* script, which is a Python interface for ParaView. These components will utilize a shared storage facility for post-processing and shared storage to establish a unified location for sharing geometry binaries, integrating with an Amazon Elastic File System.

Chapter 6

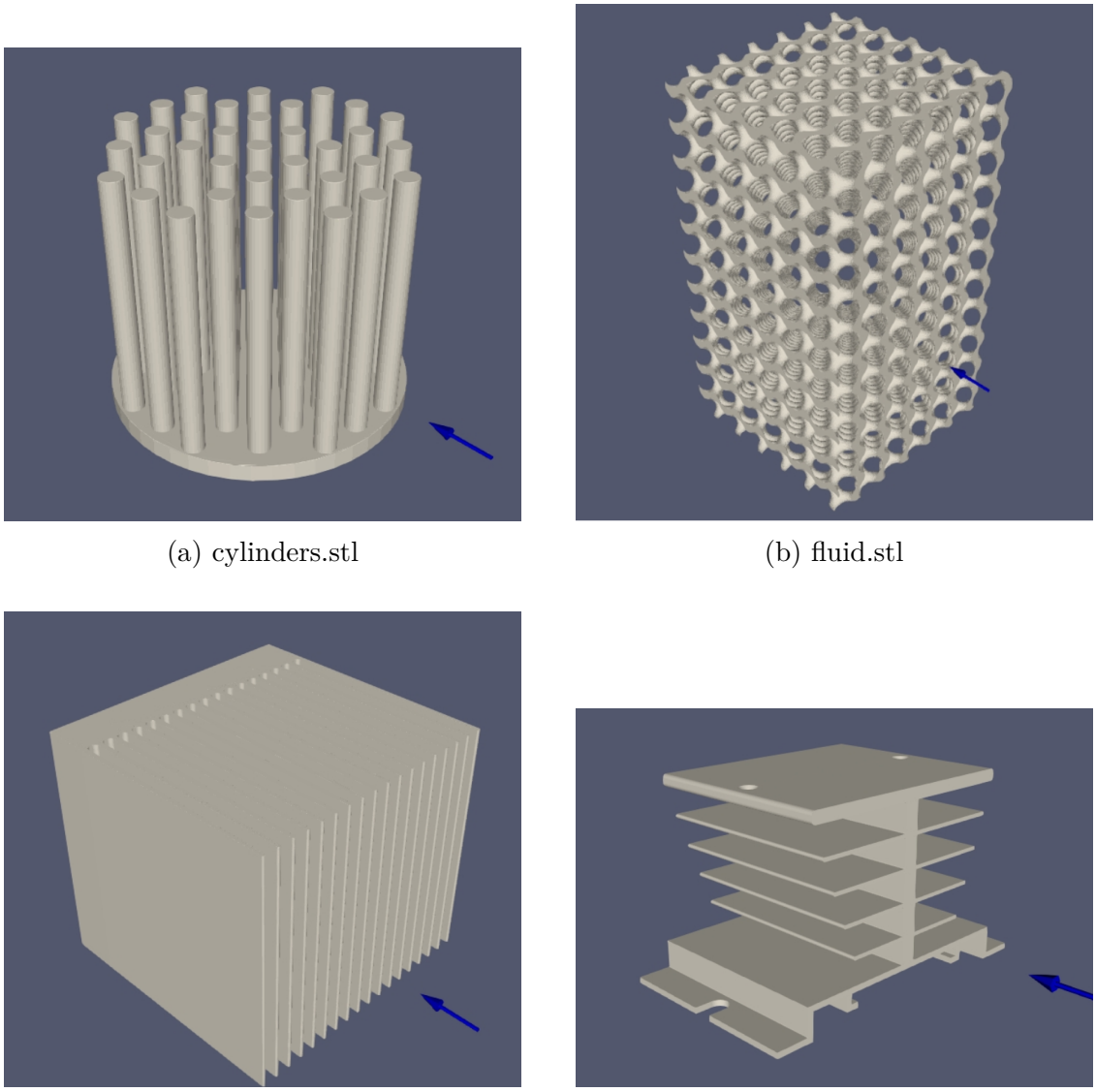
Results

6.1 Introduction

This chapter provides a wide illustration and analysis of the results obtained through a comparative evaluation of performance across various rendering approaches: Local, Remote, and Hybrid. These results have been acquired from the execution of stressful tests involving four distinct geometries each characterized by varying dimensions. These geometries are represented in STL files of considerable scale, which typically align with the types of geometries that users might employ for Computational Fluid Dynamics (CFD) simulations. Figure 6.1 showcases the geometries utilized in these tests: Figure 6.1a, measuring 365 MB in size with 7,665,694 triangles; 6.1b, with a size of 594 MB and 3,860,958 triangles; Figure 6.1b, measuring 906 MB and featuring 19,019,730 triangles; and Figure 6.1c, with a size of 1.37 GB and 29,614,110 triangles.

Each geometry was subjected to visualization sessions utilizing the aforementioned rendering approaches, each lasting 20 minutes. During each session, the geometry was interacted with a two-second intervals, employing random movements generated by an automated pointer. This two-second pause was intentionally incorporated to highlight how high-fidelity rendering was achieved after cessation of interactions with the geometries.

On the client side, the visualization sessions were run using a computer equipped with Windows 10 Pro 64-bit with an 11th Gen Intel Core i5-1135G7 processor running at 2.40GHz (using 8 CPUs), 16GB of RAM, and an Intel(R) Iris(R) Xe Graphics card with 8GB of GPU RAM. The web browser employed was Firefox version 118.0.2. On server-side, an Amazon EC2 instance (g4dn.xlarge type) was selected. This server has 16GB of RAM, a 25 Gb/s bandwidth, 4 vCPUs based on Intel Cascade Lake architecture, and was enhanced with an NVIDIA T4 GPU. These instances are good at delivering high-performance capabilities for graphic



(a) cylinders.stl

(b) fluid.stl

(c) heatsink_cooler.stl

(d) heatsink_rele.stl

Figure 6.1: Geometries used for test results

applications.
The results comprehend the collection of performance data from both the local and server-side components, gathering metrics such as CPU load, GPU usage, frame rate (FPS), the volume of data transmitted from the server during visualization, and the latency for hybrid rendering, quantified from the moment of user interaction cessation to the rendering of a high-fidelity image of the geometry. This information was fundamental in the assessment of whether the proposed approach outperformed traditional methods, and in determining its suitability for integration

into the production version of this SaaS offered by Optimad and ESTECO.

6.2 Local rendering approach

This section presents the performance data obtained using local rendering approach. In this particular scenario, visualization sessions exclusively employed the original version of the geometries. This was done to assess the capability of client browser in handling complex and sizable geometries. As the rendering process is entirely executed on the client-side, the focus is exclusively on client-side performance, while server-side performance remains negligible throughout the visualization process.

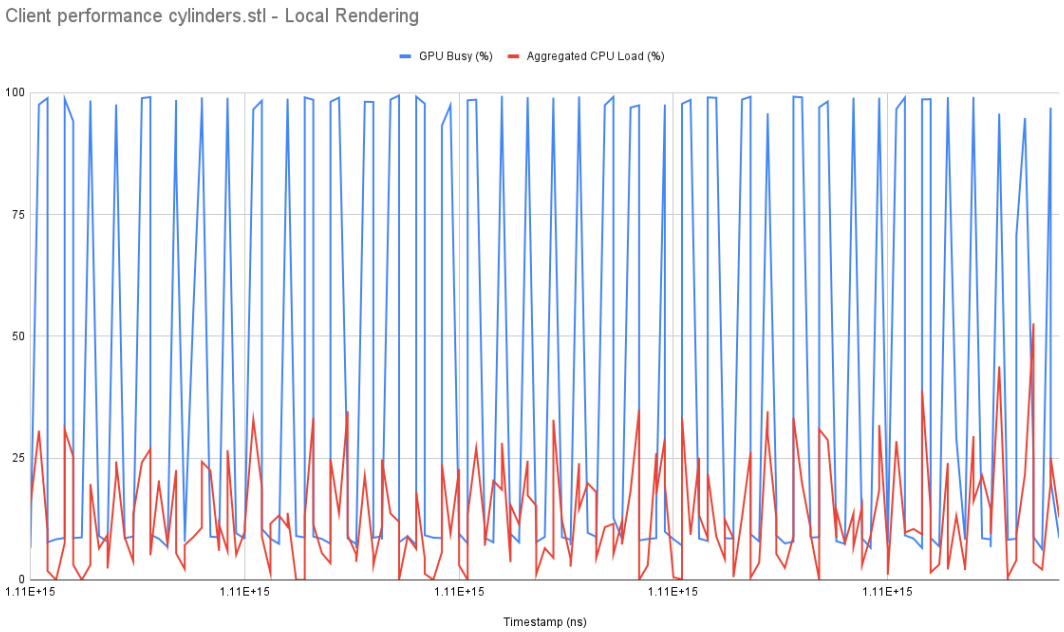
6.2.1 Client performance

In Figure 6.2 are illustrated the GPU usage and CPU load regarding the geometries cylinders.stl, fluid.stl, heatsink_cooler.stl and heatsink_rele.stl respectively during the whole visualization. The data was taken using sampling at regular intervals of approximately six seconds.

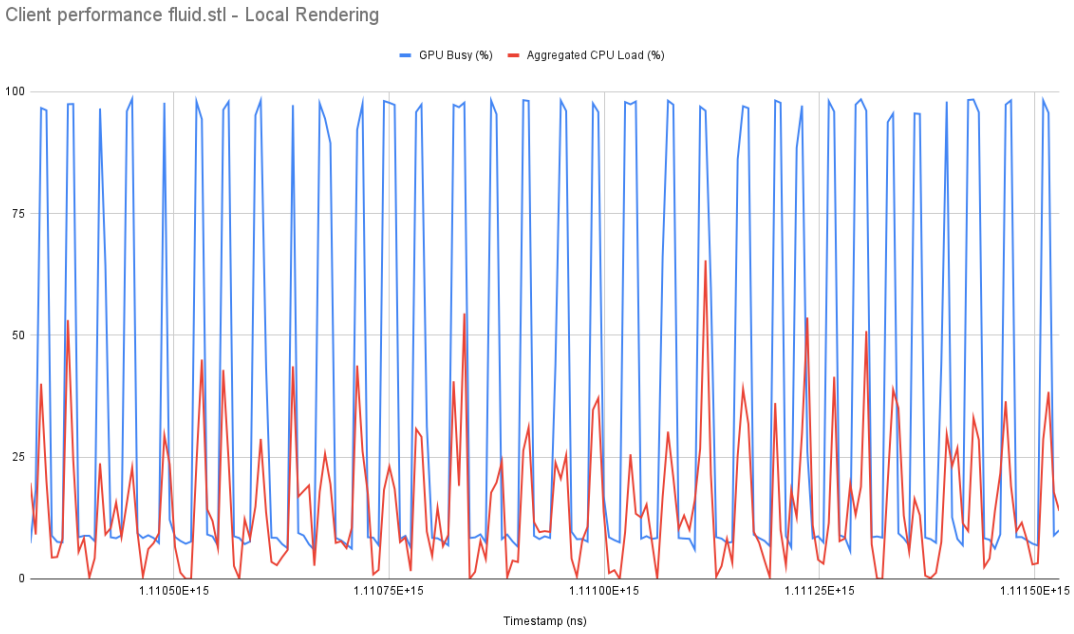
The provided graphics present a consistent trend in the extensive usage of the client machine's GPU. In all four graphics, GPU usage illustrates prominent peaks, reaching approximately 100% utilization with occasional momentary drops. This particularity is primarily attributed to the nature of rendering, which only engages the GPU when new frames are displayed. Consequently, during periods of inactivity or minimal user interaction, the GPU remains relatively idle.

In a similar way, the CPU load shows a similar pattern, marked by periodic peaks corresponding to each interaction made by the user and processed by the system. Notably, the CPU load, while presenting peaks, consistently maintains a considerably lower percentage of utilization compared to the GPU usage. This behavior remains relatively consistent, with minor variations observed when processing more complex geometries.

These graphics demonstrate that rendering large geometries has a huge impact on GPU performance. This effect for long visualizations tends to result in a general degradation of the user's computer performance, particularly due to the considerable memory space occupied by these large geometries.

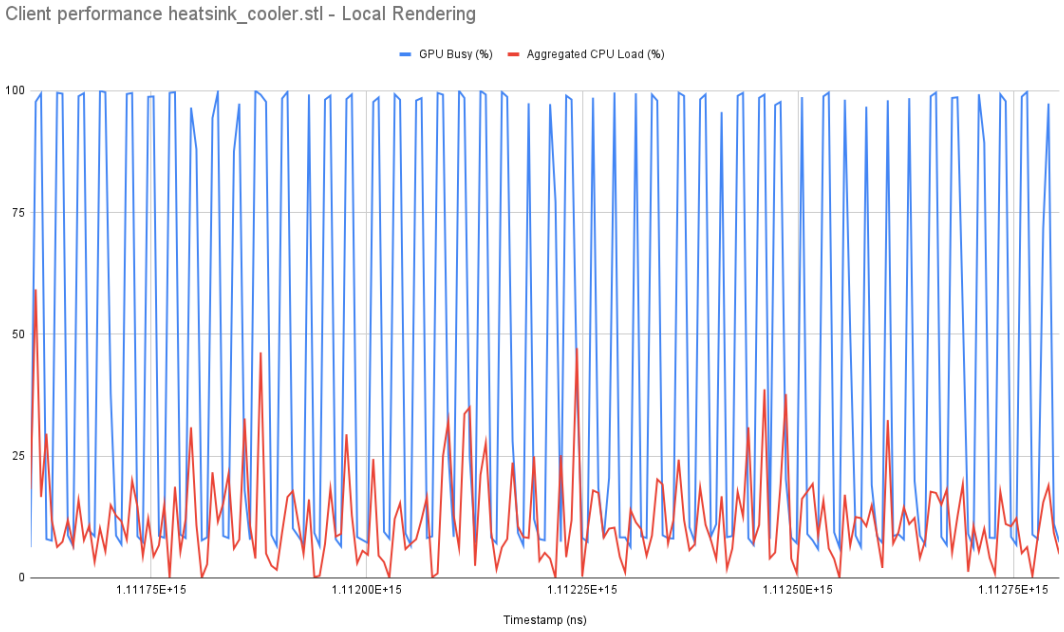


(a) cylinders.stl

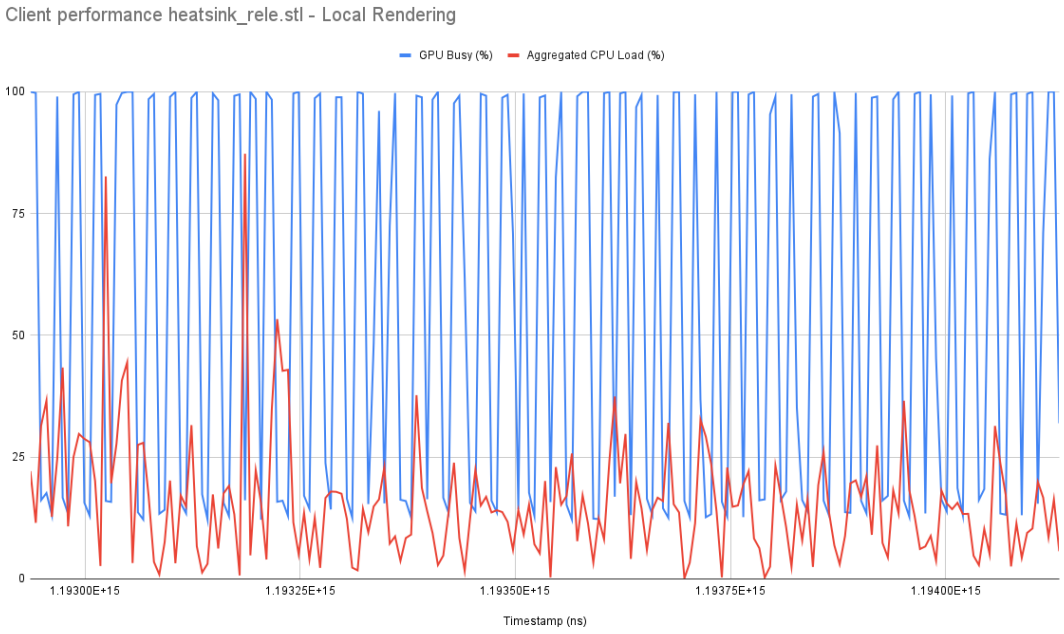


(b) fluid.stl

Figure 6.2: GPU usage and CPU load client-side with local rendering.



(c) heatsink_cooler.stl



(d) heatsink_rele.stl

Figure 6.2: GPU usage and CPU load client-side with local rendering

6.2.2 Frame rate and data transferred

Table 6.1 illustrates the frame rate and data transferred during each geometry's visualization session. Peculiarly, a considerable decrease in frames per second is observed as the geometry's size increases, with the exception of *fluid.stl* which, despite being larger than *cylinders.stl* exhibits a superior frame rate. This situation can be potentially attributed to the fact that, although *fluid.stl* possesses a larger file size, it contains fewer triangles than *cylinders.stl*. This difference in triangle count is an important factor, as it results in a reduced processing load between frames in *fluid.stl* displaying.

Since all visualization sessions used local rendering, the only data transmission involved was the transfer of the initial geometries from the server to the browser for display. This process incurs a significant data transfer expense for each visualization.

Geometry	Frame rate	Amount data transferred
cylinders.stl	21 fps	365 MB
fluid.stl	38 fps	594 MB
heatsink_cooler.stl	10 fps	906 MB
heatsink_rele.stl	3 fps	1.37 GB

Table 6.1: Frame rate and data transfer with local rendering

6.3 Remote rendering approach

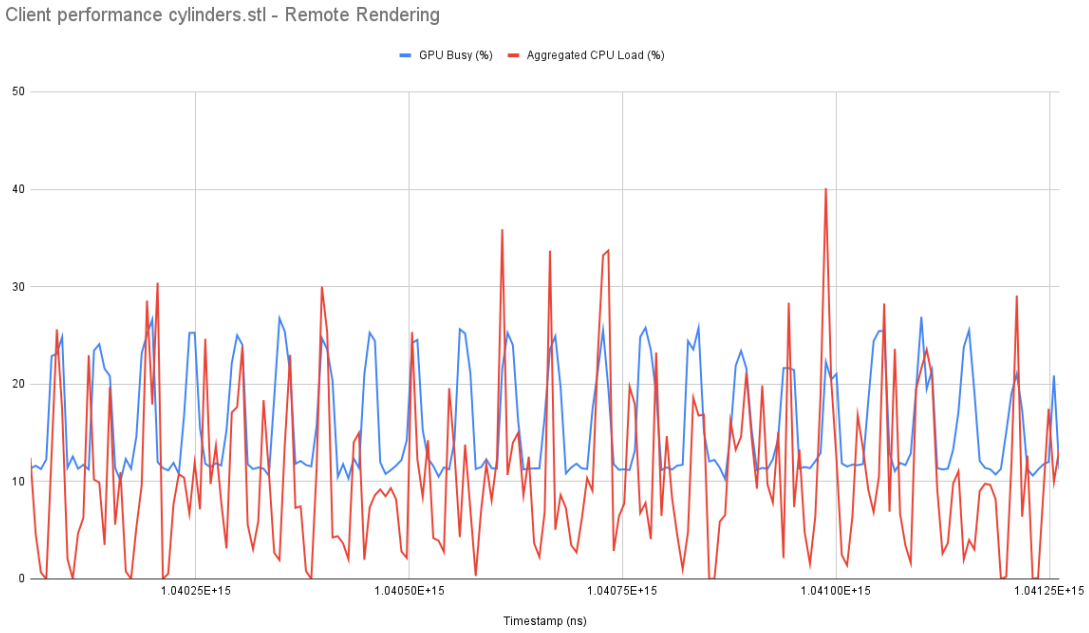
This section describes the performance data acquired through the remote rendering approach. In this case, visualization sessions made use of the server's rendering capabilities to process completely all the geometries. Consequently, information regarding to the server's GPU and CPU, together with the client's performance information, was collected to evaluate the amount of load to which the computational resources was offloaded from the client and transferred to the server.

6.3.1 Client performance

In Figure 6.3 are illustrated the GPU usage and CPU load regarding the aforementioned geometries in the same order as the previous section. The data was taken using sampling at regular intervals of approximately six seconds.

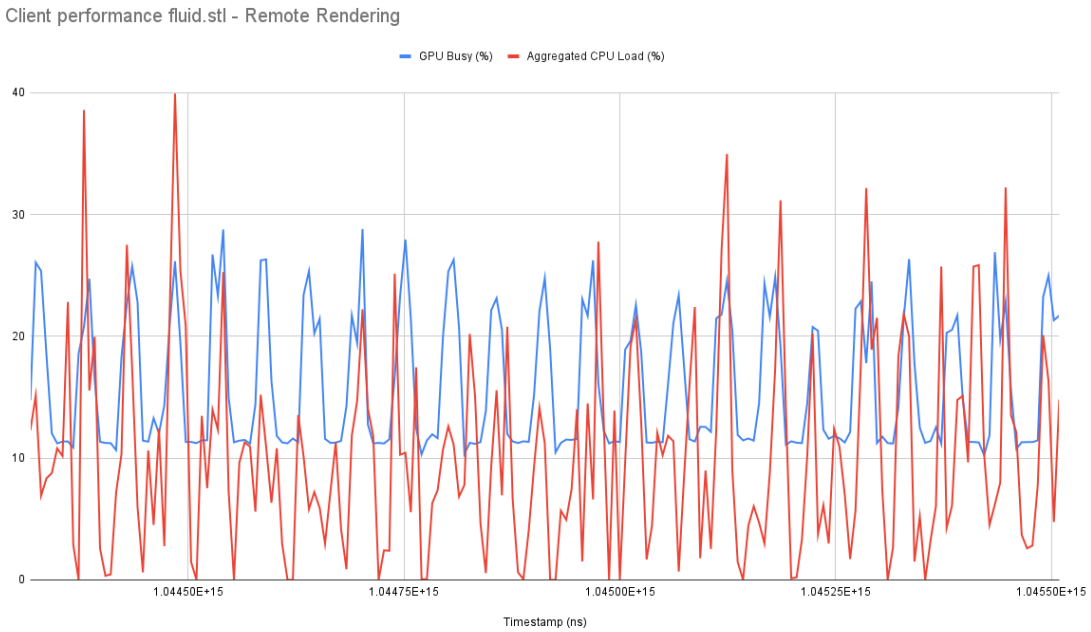
The four graphics show similar patterns in both GPU utilization and CPU load

throughout the visualization sessions. Such behaviors align with the expected outcome, given that the web browser primarily handled the processing and rendering of a sequence of images. These images, despite being generated at the highest possible quality, do not impose a processing load comparable to processing real geometries data. Remarkably, the GPU and CPU data present fluctuations within the range of 20% to 30%, occasionally showing marginal deviations. These metrics generally indicate a standard utilization of the client machine's resources for this kind of operation, with no issues of resource overhead.

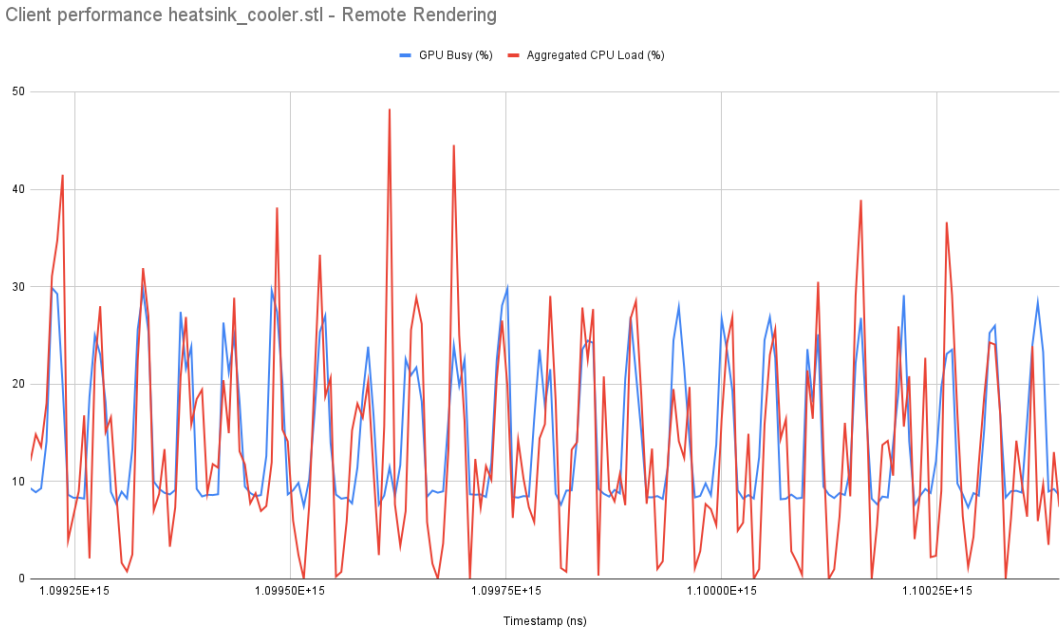


(a) cylinders.stl

Figure 6.3: GPU usage and CPU load client-side with remote rendering

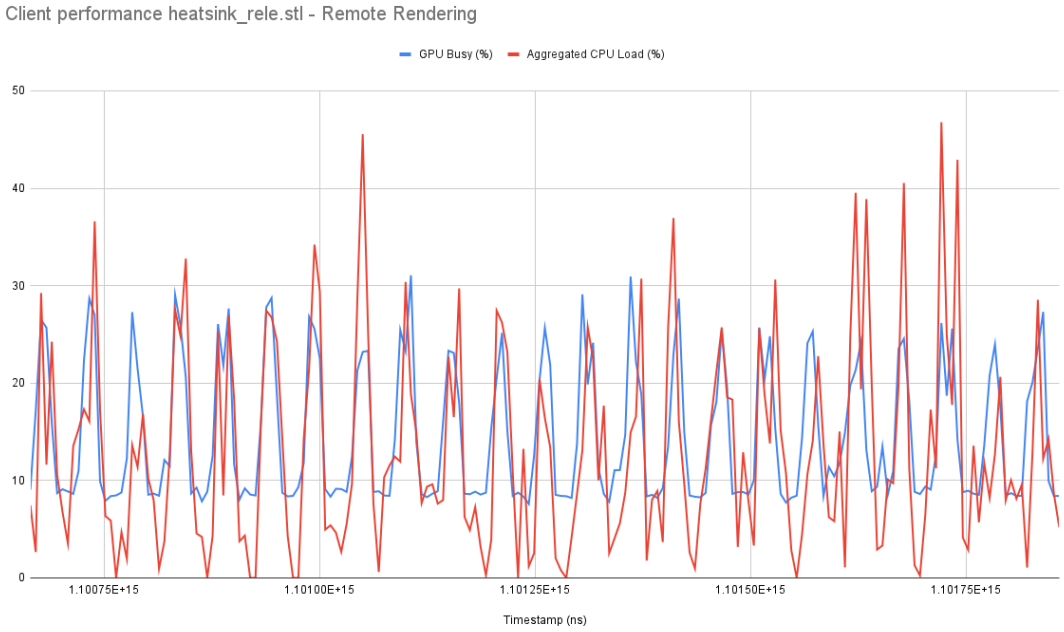


(b) fluid.stl



(c) heatsink_cooler.stl

Figure 6.3: GPU usage and CPU load client-side with remote rendering



(d) heatsink_rele.stl

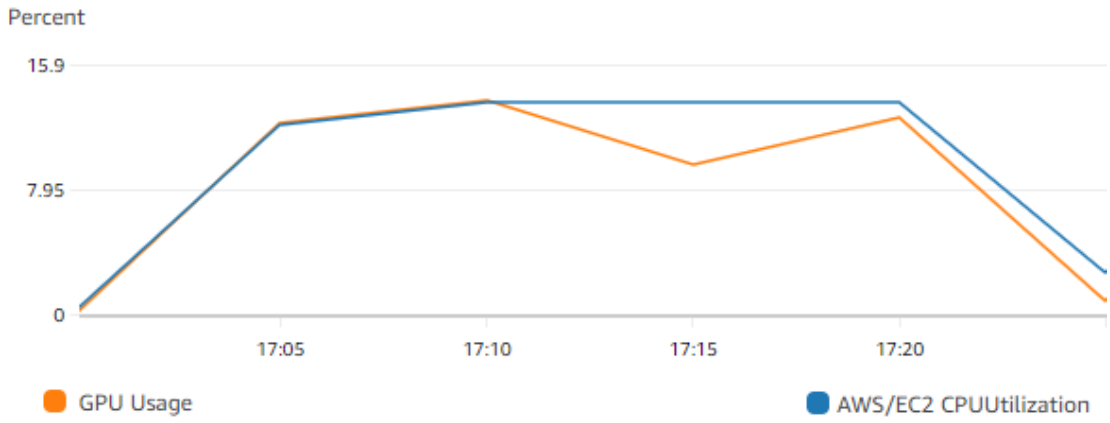
Figure 6.3: GPU usage and CPU load client-side with remote rendering

6.3.2 Server performance

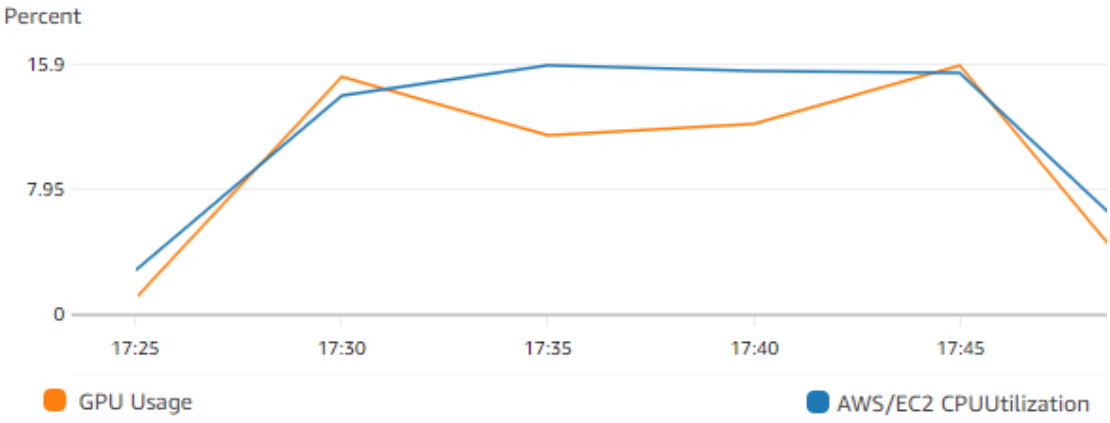
Figure 6.4 illustrates the resulting graphics generated by Monitor Amazon EC2 taking GPU usage and CPU utilization for the four visualizations with different geometries. Amazon used a five-minute interval sampling in which each sampling took the overall average of GPU/CPU utilization between intervals.

In all four graphics, the performance presented a considerable degree of efficiency, with figures consistently ranging between 12% and 17%, despite variations in the geometrical sizes. This outcome remarks the server's exceptional capacity to seamlessly handle extensive visualizations involving huge geometrical complexities, while avoiding any significant performance overhead.

A good example of the server capabilities is the comparable behavior displayed by cylinders.stl and heatsink_rele.stl. Despite *heatsink_rele.stl* being nearly four times larger in size and having approximately 22 million additional triangles compared to *cylinders.stl*, both geometries presented performance levels well below 16% of the server's overall capacity. This observation demonstrates that server could manage even larger geometries.

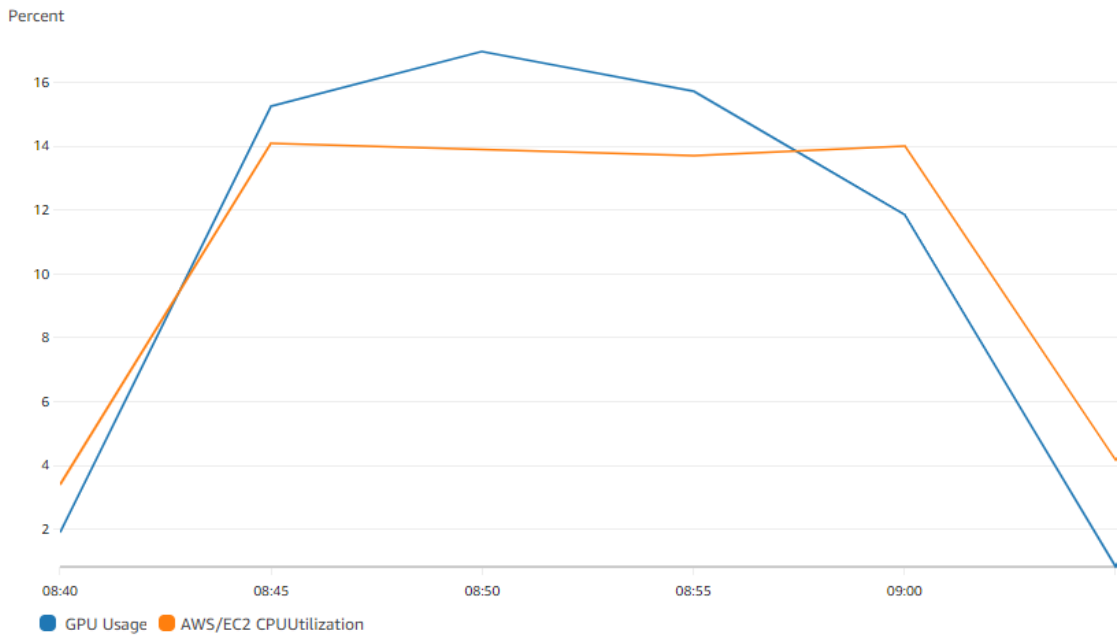


(a) cylinders.stl

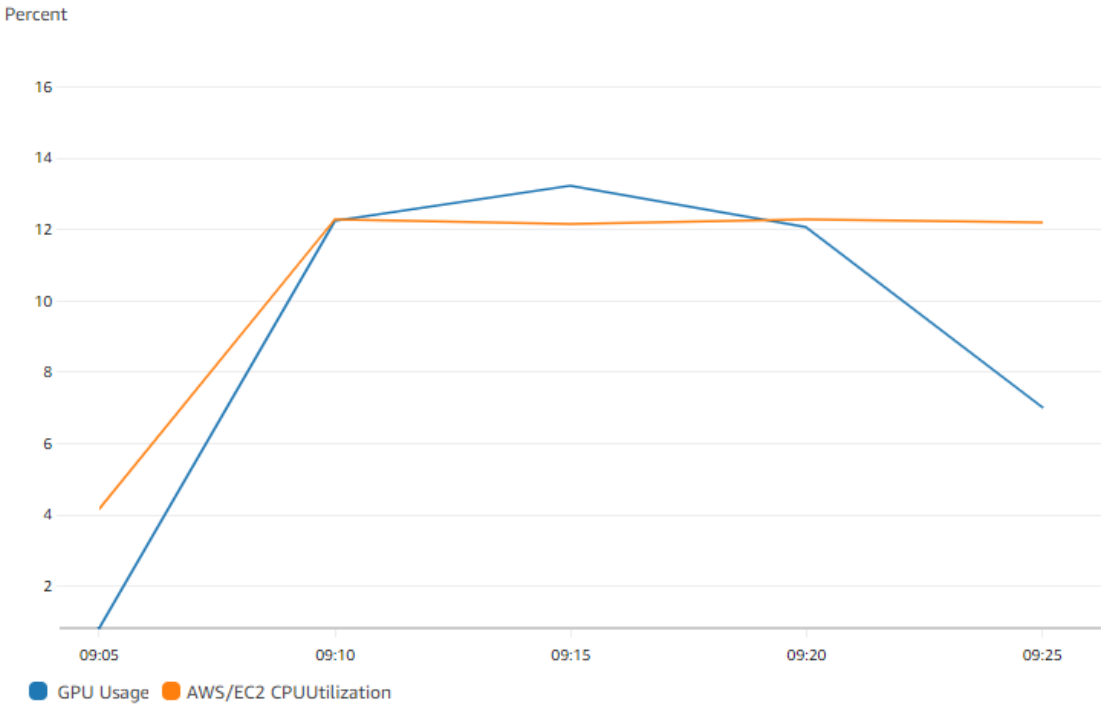


(b) fluid.stl

Figure 6.4: GPU usage and CPU load server-side with remote rendering



(c) heatsink_cooler.stl



(d) heatsink_rele.stl

Figure 6.4: GPU usage and CPU load server-side with remote rendering

6.3.3 Frame rate and data transferred

Table 6.2 represents the frame rate and data transmission associated with remote rendering visualizations. The frame rate was calculated by tallying the images received by the server and displayed by the browser per second. The volume of data transmitted from the server to the client was determined by taking the quantity of images transferred in each visualization, which were executed at the utmost possible quality.

Remote rendering presented a frame rate between 22 and 24 fps, which can fluctuate depending on client network conditions. On the other hand, the amount of data transfer is not directly proportional with respect to size of the geometries, since for example the largest geometry showed the lowest data transfer (81 MB) whereas *fluid.stl* transferred 1.42 GB, surpassing its original file size. This behavior could depend on the level of detail represented in each visualization, The more complex graphical details rendered in each frame, the larger the resultant image becomes. Besides, less frames per second sent by the server, less the overall amount of data. That could explain why *fluid.stl* had the largest amount of data transferred, since is the one that showed more details on each frame and more frames displayed per second.

Geometry	**Frame rate**	**Amount data transferred**
cylinders.stl	24 fps	740 MB
fluid.stl	24 fps	1.42 GB
heatsink_cooler.stl	23 fps	127 MB
heatsink_rele.stl	22 fps	81 MB

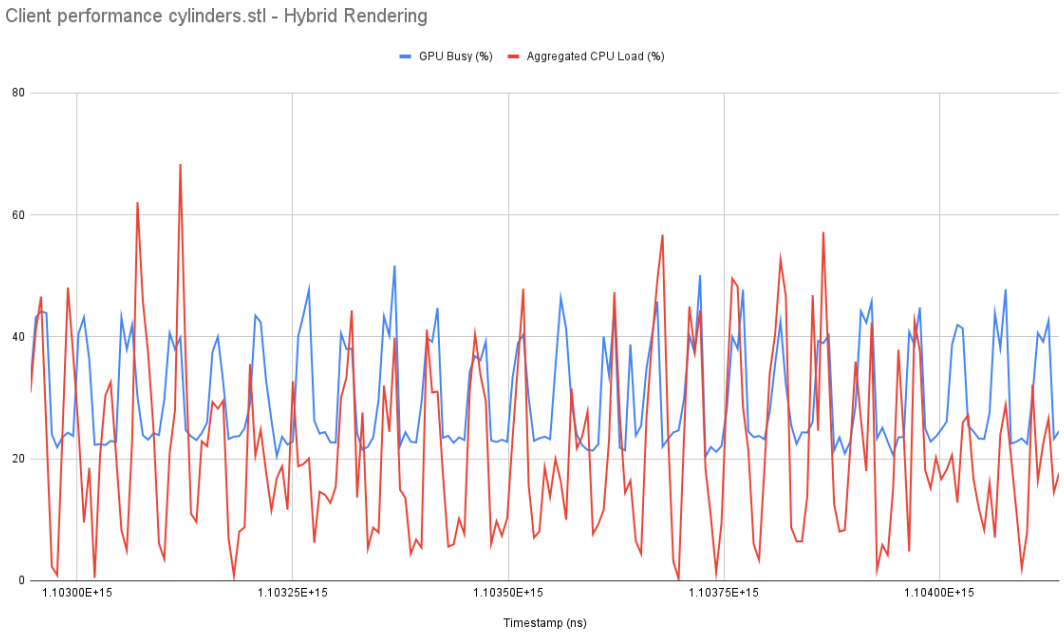
Table 6.2: Frame rate and data transfer with remote rendering

6.4 Hybrid rendering approach

This section encompasses the data results of the hybrid rendering approach. Similar to remote rendering, the data collected includes both client and server performance metrics, as well as the data transfer rate and frame rate achieved during the visualizations. Furthermore, this approach also covers an assessment of the latency that occurs between the cessation of interaction with the geometry and the generation of a high-fidelity image by the server.

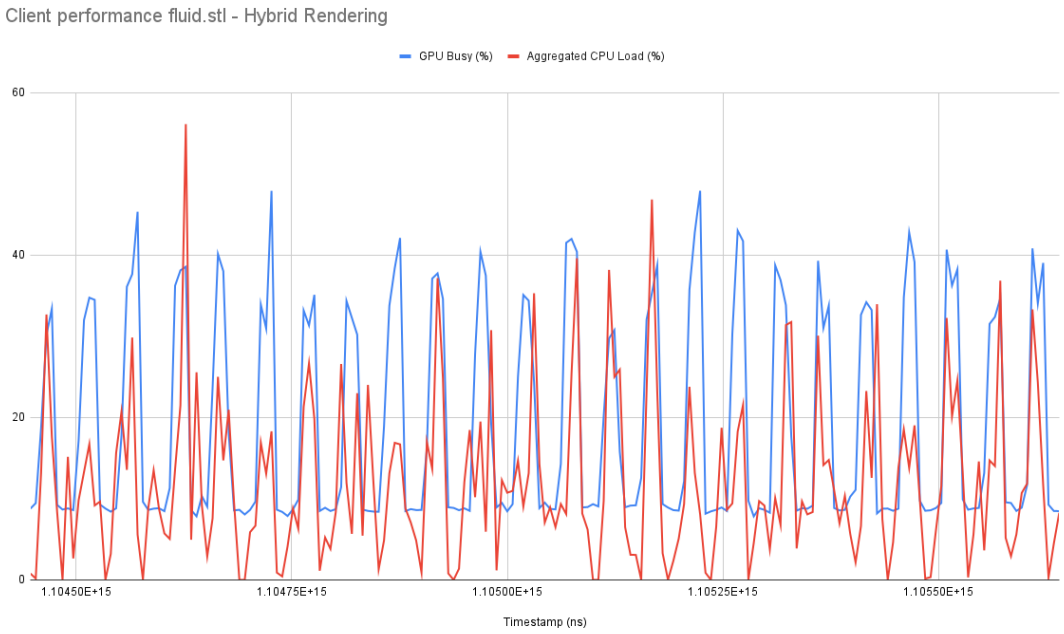
6.4.1 Client performance

Figure 6.5 shows the client performance of the four geometries visualizations. As mentioned before, the data was taken every six seconds nearly.
As expected, the graphics presented a similar behavior having a GPU usage around 40% and a CPU load around 30%. This is because, in theory, the geometries rendered on client-side have almost the same characteristics (same number of triangles and the same size). Overall, these tests presented a normal and safe usage of CPU and GPU resources on client machines.

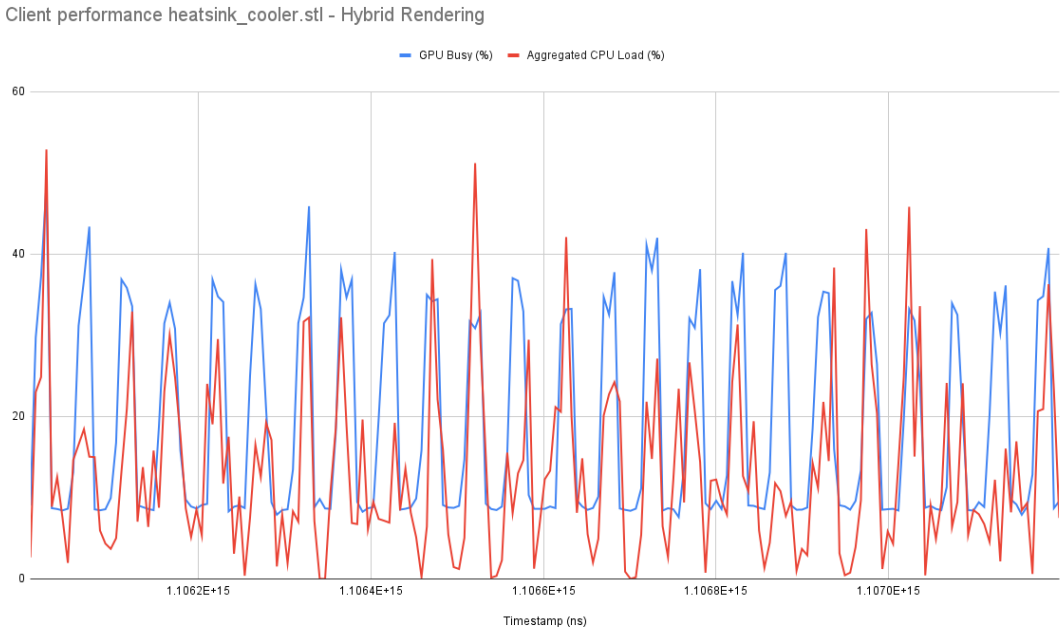


(a) cylinders.stl

Figure 6.5: GPU usage and CPU load client-side with hybrid rendering

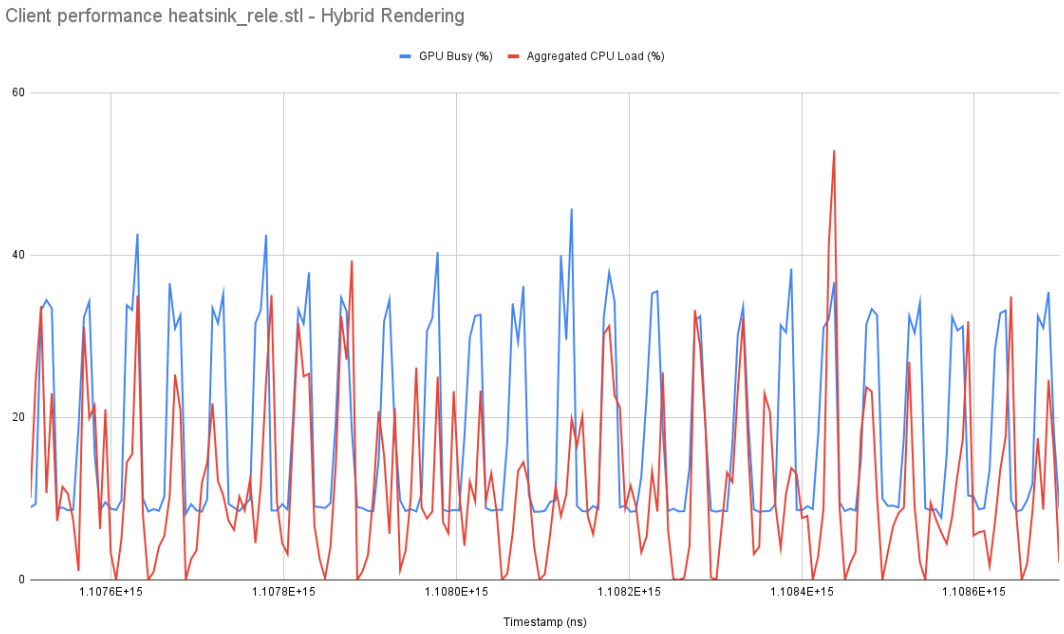


(b) fluid.stl



(c) heatsink_cooler.stl

Figure 6.5: GPU usage and CPU load client-side with hybrid rendering



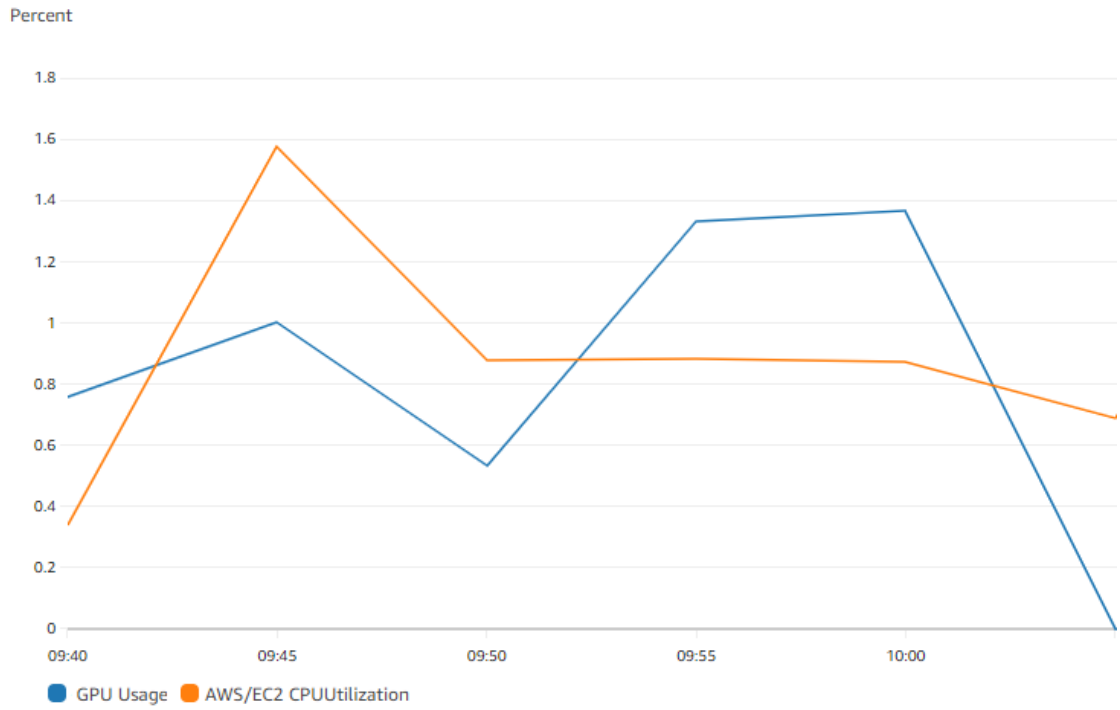
(d) heatsink_rele.stl

Figure 6.5: GPU usage and CPU load client-side with hybrid rendering

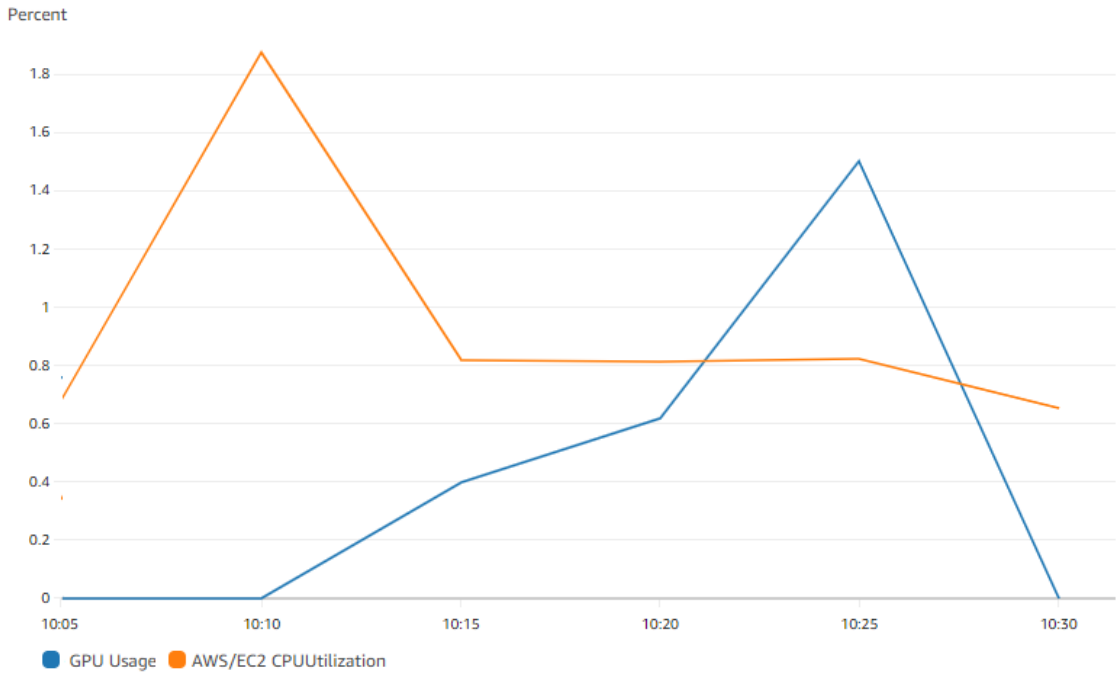
6.4.2 Server performance

Similar to the server performance on the remote rendering, Figure 6.6 shows the resulting graphics generated by Monitor Amazon EC2 using a five-minute interval sampling, taking the overall average of GPU/CPU utilization between intervals. The performances do not include decimation operations since the algorithm used saves computational costs and takes only a few seconds, therefore the number of resources needed is negligible [44].

Surprisingly, in all four graphics there was a trend of GPU and CPU utilization lower than 10%. In this approach it is visible the small fluctuations in performances as the geometry size increases having for example in the last figure a GPU usage roughly to 8% while in the first two are 2%. In general, this usage was expected since the server did not render the geometries actively but only when there was a pause every two seconds.

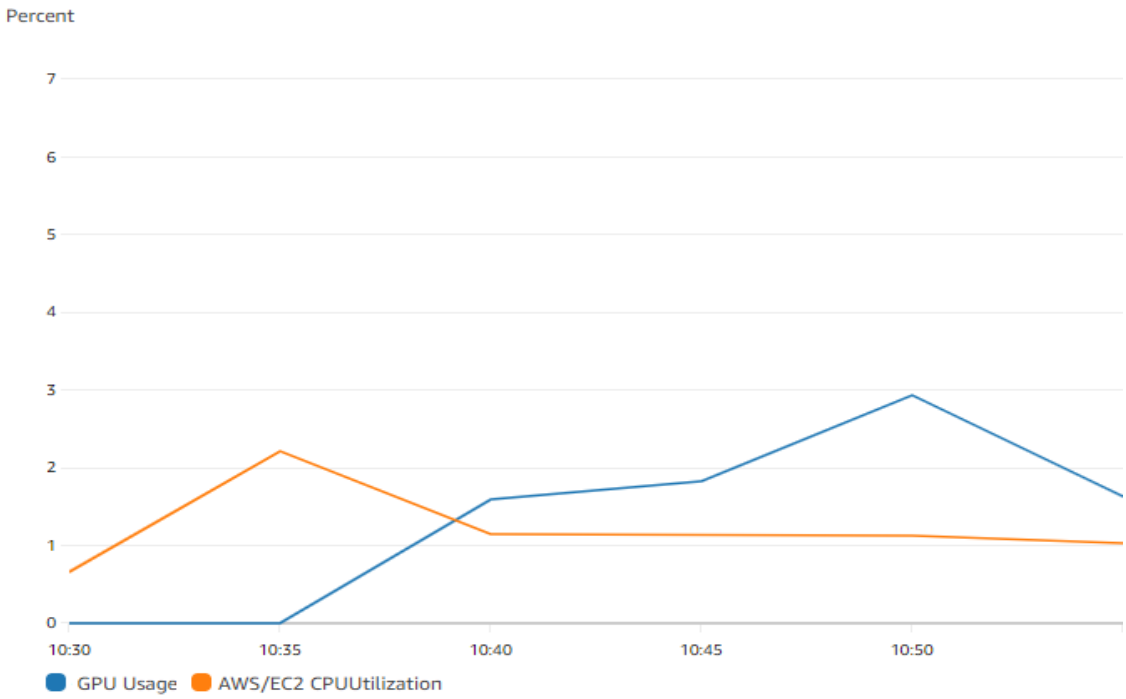


(a) cylinders.stl

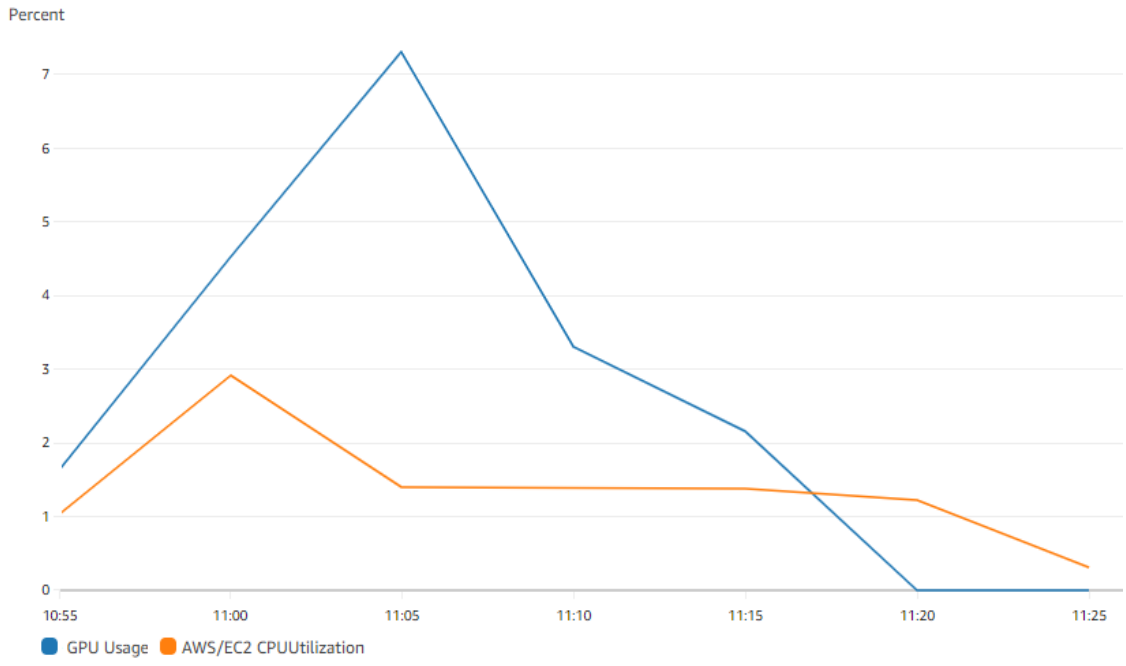


(b) fluid.stl

Figure 6.6: GPU usage and CPU load server-side with hybrid rendering



(c) heatsink_cooler.stl



(d) heatsink_rele.stl

Figure 6.6: GPU usage and CPU load server-side with hybrid rendering

6.4.3 Frame rate and data transferred

Table 6.3 illustrates the frame rate, amount of data transferred, and the average image delay during the visualizations using hybrid rendering.

Considering the decimated version of the geometries rendered on client-side, the frame rate for all the geometries was 60 fps which means that there was a seamless visualization and interaction on each geometry. In addition, the overall amount of data transferred including the transmission of the decimated geometry was incredibly low for each geometry. Similar to remote rendering, the size of the images depends on the graphical details of each frame transmitted. In fact, this pattern was also observed on a smaller scale with *fluid.stl* and *heatsink_rele.stl*, with the former showing the highest volume of data transferred and the latter being associated with a comparatively lower volume.

In general, the average image latency delivered by the server after each interaction is around 250 ms. In particular, depending on the geometry size, this delay resulted in a gradual increment. That fact occurred for the time employed by the server to process larger data.

Geometry	Frame rate	Amount data transferred	Average image latency
cylinders.stl	60 fps	3.8 MB + 1 MB	217.54 ms
fluid.stl	60 fps	6.2 MB + 1 MB	218.56 ms
heatsink_cooler.stl	60 fps	3.3 MB + 1MB	240.55 ms
heatsink_rele.stl	60 fps	2.5 MB + 1MB	274.73 ms

Table 6.3: Frame rate and data transfer with hybrid rendering

6.5 Analysis

After presenting the results of the three aforementioned approaches, it is now imperative to analyze their respective trends in order to discern the differences in performance and assess which one is more opportune to a seamless user experience while maintaining cost-effectiveness.

First of all, the local rendering approach exhibited the most inferior performance. Despite its advantage in reducing computational costs on the server side, as it offloads all the workload to the client, it disappointingly met expectations by delivering low performance. This can be attributed to the inherent computational overhead incurred when handling large datasets in the web browser, particularly

with the GPU when rendering large geometries. Even with the most modest of geometries, the graphical performance presented noticeable struggles during rendering. Additionally, the frame rate analysis revealed a direct correlation between the size of the geometry and the decreasing fluidity of visualization. Another drawback of this approach lies in the necessity to transfer the original geometry from the server for each new visualization session, resulting in an unsatisfactory user experience characterized by prolonged transfer wait times and an expensive data transfer cost.

In reference to remote and hybrid rendering, both approaches presented adequate performance with regard to user experience. In normal user usage, the client-side performance of both approaches is generally satisfactory. Notably, remote rendering evidenced greater efficiency, approximately 10%, when compared to hybrid rendering, across both CPU and GPU performance metrics. In addition, remote rendering in some cases presented an eye-human acceptable frame rate (between 24-30 fps) to perceive cinematic. Be that as it may, the results also revealed that, as the complexity of the rendered geometry increases, the frame rate tends to decline. Consequently, scenarios related to the frame rates observed in *heatsink_cooler* and *heatsink_rele* as detailed in Table 6.2 are susceptible to perceptible flicker, which can be discomforting to the human eye [45]. It is essential to clarify that the actual frame rate in real-world situations is subject to the quality of the client's network connection. A stable and robust bandwidth is imperative to ensure a seamless stream of images and consequently, a higher frame rate.

Furthermore, another crucial disadvantage of remote rendering, is the large amount of data transferred from server, being comparable to the amount of data transferred in local rendering. This approach, despite being adequate enough to have a normal visualization, represents a huge cost in terms of data transferred that, mentioned in Chapter 4.1, is something that Optimad desires to avoid.

On the contrary, the results revealed that hybrid rendering exhibited superior performance during testing, with the exception of client-side performance, as previously mentioned, where remote rendering displayed a slight advantage. From a general perspective, rendering a modestly-sized geometry on the client-side yields standard CPU and GPU utilization, thereby preventing any considerable overhead. Consequently, the frame rate consistently ranked highest for all four geometries, contrasting with remote rendering, where the number of displayed frames is contingent on network conditions and its latencies [46]. On the other hand, server-side performance was optimal. Despite remote rendering also exhibiting minimal CPU and GPU utilization, the hybrid approach required fewer interventions to process high-quality images. Consequently, data transfer volumes were significantly lower, allowing Optimad to afford amount of data similar to those detailed in Table 6.3.

The latency in delivering high-quality images fell short of ideal, as real-time interactions typically demand a latency of no more than 100 ms. However, this latency, while not optimal, is sufficiently manageable and does not significantly impede fluid user experience.

Chapter 7

Conclusions

In conclusion, this thesis explored the application of a hybrid approach for rendering large 3D geometries in a cloud-based CFD platform, focusing on the use of the ParaViewWeb with VTK.js frameworks integrated with the technologies that are currently used in Optimad. The results and analysis provided valuable insights into the efficiency, user experience, and cost-effectiveness of this approach compared to local and remote rendering. Several key points emerged from the study.

The use of ParaViewWeb with VTK.js demonstrated an easily learnable curve and simplified configuration for achieving the desired rendering scope. This accessibility was essential not only for creating the approach proposed but also the possibility to implement the other two compared approaches, in order to assess their advantages and disadvantages.

The local rendering approach proved to be inferior in terms of performance due to computational overhead on the client side and the need to transfer large geometries, leading to extended wait times and high data transfer costs. This approach is not recommended for handling large 3D geometries in web applications.

Both remote and hybrid rendering approaches demonstrated satisfactory user experiences. Remote rendering was slightly more efficient on client-side, but it suffered from decreased frame rates as the complexity of rendered geometries increased, potentially leading to perceptible flicker. A stable network connection is crucial for optimal performance in remote rendering.

Hybrid rendering outperformed the other approaches in most aspects. It provided consistent frame rates, low CPU and GPU utilization on client-side, and optimal server-side performance. The reduced data transfer volume made it an attractive choice, aligning with the goals of organizations aiming to minimize data transfer costs. While the latency in delivering high-quality images in the hybrid approach

was not ideal for real-time interactions, it was manageable and did not significantly obstruct the overall user experience.

While specific monetary data was not presented in the study, it was indicated that the hybrid rendering approach showed promise in terms of being economically efficient compared to alternatives which presented a significant data transfer from the server, as a result, in high data transfer costs. Therefore, using this approach implies potential cost savings for organizations implementing this technology.

In future work, it is suggested to explore the other tools mentioned in Chapter 2 to assess whether they can provide better performance, since some of them resulted to be more recent than the ones used in this thesis. Additionally, there is room for improvement by evaluating more efficient decimation algorithms or evaluating transfer costs and provide a larger decimation size target to prevent the loss of the original geometry's form, further enhancing the user experience. In this way, future tests in production scenarios will be performed to obtain a real user perception of visualizations.

In summary, the study underlines the advantages of the hybrid rendering approach, with ParaViewWeb and VTK.js as key enabling technologies, for efficiently rendering large 3D geometries in web applications. It offers a balanced mix of performance, user experience, and cost-effectiveness, making it a promising choice for organizations looking to leverage 3D visualization in web applications.

Bibliography

[1] "VRML97 Functional specification and VRML97 External Authoring Interface (EAI) International Standard ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002." ISO/IEC International Standard, 1997. ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002.

[2] F. Niebling, A. Kopecki, and M. Becker, "Collaborative steering and post-processing of simulations on hpc resources: Everyone, anytime, anywhere," in *Web3D Symposium Proceedings*, pp. 101–108, ACM, 2010.

[3] S. Scanzio, S. Cumani, R. Gemello, F. Mana, and P. Laface, "Parallel implementation of Artificial Neural Network training for speech recognition," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1302–1309, 2010.

[4] R. Miao, J. Song, and Y. Zhu, "3d geographic scenes visualization based on WebGL," in *2017 6th International Conference on Agro-Geoinformatics*, pp. 1–6, IEEE, 2017.

[5] A.-L. Boutsi, C. Ioannidis, and S. Soile, "An integrated approach to 3d web visualization of cultural heritage heterogeneous datasets," *Remote Sensing*, vol. 11, no. 21, 2019.

[6] J. Jomier, S. Jourdain, U. Ayachit, and C. Marion, "Remote visualization of large datasets with MIDAS and ParaViewWeb," in *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pp. 147–150, ACM, 2011.

[7] W.-K. Yoo, S. Shi, W.-J. Jeon, K. Nahrstedt, and R. H. Campbell, "Real-time parallel remote rendering for mobile devices using graphics processing units," in *2010 IEEE International Conference on Multimedia and Expo, ICME 2010*, pp. 902–907, 2010.

[8] K. Engel, O. Sommer, and T. Ertl, "A framework for interactive hardware accelerated remote 3d-visualization," in *Data Visualization 2000* (W. C. de Leeuw and R. van Liere, eds.), (Vienna), Eurographics, Springer, 2000.

[9] G. Cena, S. Scanzio, and A. Valenzano, "A Prototype Implementation of Wi-Fi Seamless Redundancy with Reactive Duplication Avoidance," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 179–186, 2018.

[10] G. Cena, S. Scanzio, and A. Valenzano, "Improving Effectiveness of Seamless Redundancy in Real Industrial Wi-Fi Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2095–2107, 2018.

[11] H. Kato, T. Kobayashi, M. Sugano, and S. Naito, "Split rendering of the transparent channel for cloud AR," in *IEEE 23rd International Workshop on Multimedia Signal Processing, MMSP 2021*, 2021.

[12] C. Liu, W. T. Ooi, J. Jia, and L. Zhao, "Cloud baking: Collaborative scene illumination for dynamic web3d scenes," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 14, no. 3s, 2018.

[13] X. Wu and G. Pei, "Collaborative graphic rendering for improving visual experience," in *Collaborative Computing: Networking, Applications and Worksharing. CollaborateCom 2008* (E. Bertino and J. B. D. Joshi, eds.), vol. 10 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, (Berlin, Heidelberg), Springer, 2009.

[14] W. Schroeder, L. S. Avila, and W. Hoffman, "Visualizing with vtk: A tutorial," *IEEE Computer Graphics and Applications*, vol. 20, no. 5, pp. 20–27, 2000.

[15] P. Ramachandran and G. Varoquaux, "Mayavi: 3d visualization of scientific data," *Computing in Science and Engineering*, vol. 13, no. 2, p. 40 – 51, 2011. Cited by: 448; All Open Access, Green Open Access.

[16] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[17] G. A. P. Eriksson and S. Håkansson, "Webrtc: Enhancing the web with real-time communication capabilities," *Ericsson Review (English Edition)*, vol. 89, no. 1, pp. 4–9, 2012.

[18] J. Ahrens, B. Geveci, and C. Law, "Paraview: an end-user tool for large-data visualization," in *Visualization Handbook* (C. D. Hansen and C. R. Johnson, eds.), pp. 717–731, Burlington: Butterworth-Heinemann, 2005.

[19] S. Jourdain, U. Ayachit, and B. Geveci, "Paraviewweb, a web framework for 3d visualization and data processing," in *Proc. of the IADIS Int. Conf. - Computer Graphics, Visualization, Computer Vision and Image Processing, CGVCVIP 2010, Visual Commun., VC 2010, Web3DW 2010, Part of the MCCSIS 2010*, pp. 502–506, 2010. Cited By :18.

[20] C. Rossant and N. Rougier, "High-performance interactive scientific visualization with datoviz via the vulkan low-level gpu api," *Computing in Science and Engineering*, vol. 23, no. 4, p. 85 – 90, 2021.

[21] M. Bailey, "Introduction to the vulkan graphics api," in *ACM SIGGRAPH 2018 Courses, SIGGRAPH 2018*, 2018.

[22] A. Dakkak, C. Pearson, and W. . Hwu, "Webgpu: A scalable online development platform for gpu programming courses," in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pp. 942–949, 2016. Cited By :10.

[23] R. Cabello, "Three.js." http://threejs.org/, 2010.

[24] J. Behr, P. Eschler, Y. Jung, and M. Zöllner, "X3dom - a dom-based html5/ x3d integration model," *Proceedings of Web3D 2009: The 14th International Conference on Web3D Technology*, pp. 127–135, 2009.

[25] H. Andrioti, A. Stamoulias, K. Kapetanakis, S. Panagiotakis, and A. G. Malamos, "Integrating webrtc and x3dom: Bridging the gap between communications and graphics," in *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, (New York, NY, USA), p. 9–15, Association for Computing Machinery, 2015.

[26] Kitware, "Paraviewweb." https://kitware.github.io/paraviewweb/docs/, Accessed 2023.

[27] Kitware Inc., "Vtk.js." https://kitware.github.io/vtk-js/, Accessed 2023.

[28] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "The agile manifesto," 2001.

[29] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *Product-Focused Software Process Improvement* (F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson, eds.), (Berlin, Heidelberg), pp. 386–400, Springer Berlin Heidelberg, 2009.

[30] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[31] K. Schwaber, "Scrum development process," in *Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas*, pp. 117–134, Springer, 1997.

[32] E. Brechner, *Agile Project Management with Kanban*. Best practices, Microsoft Press, 2015.

[33] K. Finnigan, *Enterprise Java microservices*. Simon and Schuster, 2018.

[34] T. Koleoso and T. Koleoso, "Microservices with quarkus," *Beginning Quarkus Framework: Build Cloud-Native Enterprise Java Applications and Microservices*, pp. 51–132, 2020.

[35] A. Boicea, F. Radulescu, and L. I. Agapin, "Mongodb vs oracle–database comparison," in *2012 third international conference on emerging intelligent data and web technologies*, pp. 330–335, IEEE, 2012.

[36] J. Ponge, *Vert. x in Action: Asynchronous and Reactive Java*. Manning Publications, 2020.

[37] A. Moiseev and Y. Fain, *Angular Development with TypeScript*. Simon and Schuster, 2018.

[38] A. L. Davis and A. L. Davis, "Gradle," *Learning Groovy 3: Java-Based Dynamic Scripting*, pp. 105–114, 2019.

[39] T. Hagos, *Beginning IntelliJ IDEA*. Springer, 2022.

[40] M. Stigler, *Amazon Web Services*, pp. 41–81. Berkeley, CA: Apress, 2018.

[41] P. Li, *Jira Essentials*. Packt Publishing Ltd, 2015.

[42] O. Gazi, "Understanding digital signal processing," 2018.

[43] D. Bohn, J. Ren, and K. Kusterer, "Conjugate heat transfer analysis for film cooling configurations with different hole geometries," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 36886, pp. 247–256, 2003.

[44] Kitware, "vtkBinnedDecimation." https://vtk.org/doc/nightly/html/classvtkBinnedDecimation.html. Accessed on October 22, 2023.

[45] B. Tag, J. Shimizu, C. Zhang, K. Kunze, N. Ohta, and K. Sugiura, "In the eye of the beholder: The impact of frame rate on human eye blink," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, (New York, NY, USA), p. 2321–2327, Association for Computing Machinery, 2016.

[46] S. Scanzio, F. Xia, G. Cena, and A. Valenzano, "Predicting Wi-Fi link quality through artificial neural networks," *Internet Technology Letters*, vol. 5, no. 2, p. e326, 2022.