Master's Degree in Computational and Mathematical Engineering

Final Master Project (FMP)

Evoker: a scalable web-GUI visualizer and mesh generator

Student: Francesc Costa Majó

Course instructor: Sergio Iserte Agut

Coordinating professor: Josep Jorba Esteve

Area of the FMP: High-Performance Computing

Date of Delivery: 06/06/2021

Francesc Costa

INDEX CARD OF THE FINAL MASTER PROJECT

Title of the FMP:	*Evoker: a scalable web-GUI visualizer and mesh generator*
Name of the author:	*Francesc Costa Majó*
Name of the TUTOR:	*Sergio Iserte Agut*
Name of the PRA:	*Josep Jorba Esteve*
Date of delivery (mm/aaaa):	06/2021
Degree:	*Master's in Computational and Mathematical Engineering*
Area of the Final Work:	*High-Performance Computing*
Language of the work:	*English*
Keywords	*3D scientific visualization, meshing, ParaView, CFD, cloud computing, distributed computing*

Abstract:

Computational fluid dynamics (CFD) is a branch of fluid mechanics that uses numerical analysis and data structure to analyze and solve problems that involve fluid dynamics. The computations rely on a mesh that discretizes the physical domain. These are used by the CFD solver to construct control volumes and to do simulations. Meshing is computer-intensive and meshing tools are not easy to use, being a lot of them interactive shells without GUI. The accuracy of the numerical solution depends on the quality of the computational mesh.

In this regard, this FMP presents Evoker, a platform as a Service (PaaS) solution, that will allow to do 3D scientific visualizations and to generate and refine complex 3D meshes from point cloud in a web-based zero-installation client-server architecture. Evoker is designed to make good use of the compute nodes of the system leveraging High-Performance Computing (HPC) techniques.

Evoker is launched as a customizable, cloud, meshing tool GUI adaptable to specific fluid-related scientific fields. Concretely, in this work Evoker demonstrates its versatility when used in waste water tanks, providing a friendly user interface and capable of leveraging underlying computational resources.

Table of contents

Index of Figures

1. Introduction

In this section, a context and justification of the FMP are presented along with the objectives and methodology followed during the two semesters. Finally, an overview of the thesis organization is detailed.

1.1. Context and justification of the final master thesis

The gateway of the Fourth Industrial Revolution is around the corner. The Fourth Industrial Revolution introduces novel key pillars [1]. The areas that this technology revolution comprises are growing as time passes and it will have a direct effect on the way we live and work. This is already happening as this thesis is being written but the expected growth is massive. This project has the aim of being interesting to the innovators and technologists of the Fourth Industrial Revolution as involves the following aspects: it has a client-server architecture, it performs 3D scientific visualization, it has cloud computing, it generates and refines 3D meshes that can be used in further CFD simulations, and it is capable of visualizing 3D objects generated by third-party devices.

In this sense, this project encompasses below pillars of the Fourth Industrial Revolution:

1. **Internet of Things**: 3D printers and scanners can upload objects to a system server.
2. **Simulation**: use of OpenFOAM to refine meshes used in CFD.
3. **Cloud Computing**: in this project, there is a client-server architecture where all the computational work is done in the cloud.
4. **Additive Manufacturing**: a 3D visualizer is key in any type of 3D printing, for instance, additive manufacturing.

On top of that, the presented project is web-based requiring zero client installation and harnesses HPC. After studying previous research, it can be followed that the implementation and conclusions reached with this thesis can be of high interest to a wide audience.

The author of the project has a personal bond to choose this project due to his professional career and background. Two years ago, he was working in a 3D printer manufacturing company, and currently, he is working in a Color Science team. In this Color team, they sometimes use tools to visualize the color gamut in 3D space (see **Figure 1**). Visualize a color gamut in 3D space is important to assess the quality of color reproduction of a device, in this case, a printer: the greater the volume the more colors it can reproduce. In **Figure 2** a screenshot of a 3D printing job is shown.

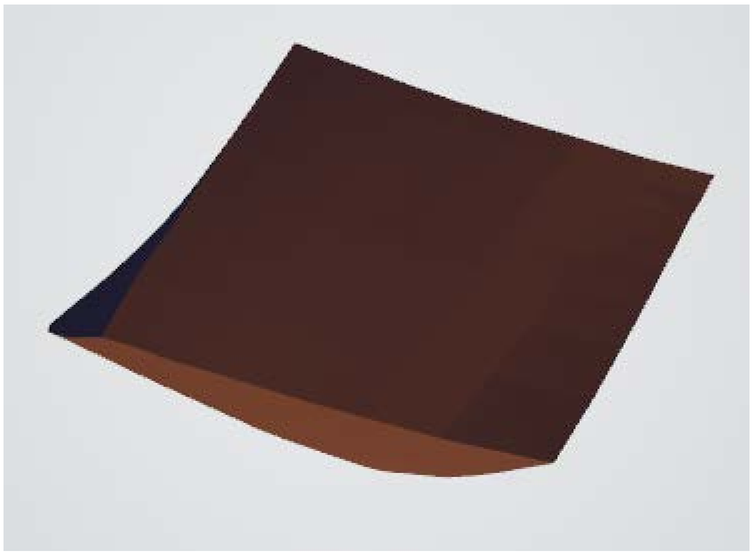


Figure 1: Gamut volume of 2D printer

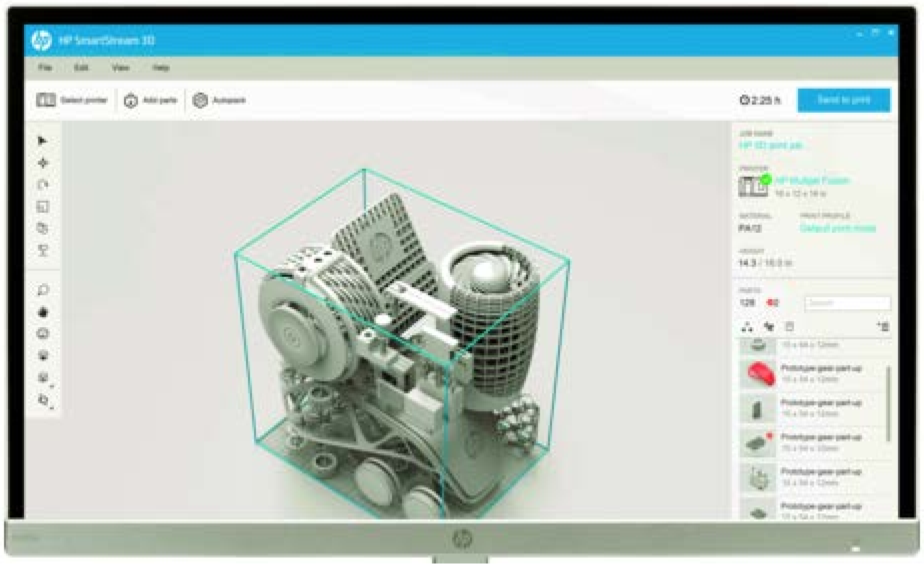


Figure 2: 3D printer job visualized in a computer

1.2. Objectives

The objectives of this project are to design, implement and evaluate a scalable client-server system for remote visualization using ParaView and also to prepare a scalable HPC cloud for mesh refinement with CFD software. The system must accept OpenFOAM projects with surfaces defining the boundary domain, where a higher level of mesh refinement is usually required. The user will be able to set the minimum and maximum refinement levels from a starting mesh and morphing the resulting split-hex mesh to the surface.

This project poses the following challenges:

1. Will the student be able to prepare a 3D visualizer with the time scheduled for an FMP (300 hours) with a consumer experience at the level of a commercial solution?
2. Will the solution be able to run in low-performance devices like tablets and mobiles?
3. Will the student be able to define, design, implement and test web front-end code for mesh refinement?
4. Will the mesh refinement be scalable in an HPC cloud?

1.3. Methodology

FMP´s tutor Sergio Iserte and the author have been meeting online for one hour once a week during the realization of this project. These meetings can be considered as a relaxed version of Agile Scrum stand-up meetings: the author updates the tutor in terms of what he did last week, what he plans to do this week, and if any roadblocks impede progress.

Sergio has been key during these meetings to guide the author in the usage of OpenFOAM. He has been giving the author very good advice and feedback regarding many aspects of the project: application front-end design, ParaView application to fork from, Azure vs AWS, or meshing knowledge.

Notes were taken during the meetings and saved in Google Docs. Those notes have been helpful as a task backlog and a progress history of this project.

1.4. Thesis organization

This thesis has seven sections. Section 1 is an introduction to the FMP and has the viewpoint of the FMP's subject without entering in describing the specific project in hand. Section 2 goes into the description of the project in terms of the scope, state-of-the-art, resources, planning, and financials. Section 2 also introduces the reader to the technologies used in this project to accomplish the goals: VTK, ParaView, Web Sockets, Vue.js and Azure. Section 3 and 4 explain the 3D visualization and the meshing functionality respectively. Section 5 shows the visualization results as well as performs a performance analysis for meshing under different process topologies. Finally, Section 6 presents the conclusions.

2. Project description

In this section, the project is described covering the scope, state-of-the-art, training, tools, resources, planning, and financials.

2.1. Overview

This project aims to offer a tool for generating and refining complex 3D meshes from a point cloud, as well as visualizing them in a client-server architecture through a web browser.

The tool will allow modifications of the meshes using refinement and filtering techniques as part of a pipeline. It will also visualize a mesh (in VTK format or OpenFOAM point clouds) already created and imported as a 3D model.

The software to be built has been named Evoker as per the definition of “to evoke”: bring (in this case an object) to the conscious mind.

This type of system can be developed for both client-side and server-side visualization architecture, to handle differing needs for performance and processing power. This is one of the most important decisions when developing a web application for visualization – where the data processing will be done. Both client and server-side processing have their advantages and drawbacks. It is important to carefully understand parameters, such as the data size, the network capability, the number of concurrent users, the type of rendering, the devices on which the application will run, and the types of computation needed [2]. However, from the inception of this project it is aimed at doing the rendering in the server for the below reasons:

1) This project is part of a Master's Thesis belonging to the HPC field. In this regard, it is crucial to provide a scalable solution that leverages all the resources in the system. For this purpose, the effort has to be made in the area of HPC in the server, leaving the client-side to be instrumental.
2) The solution will be based on a modular design that allows the offloading of meshing operations to an HPC cloud instead of overloading the webserver. That will be discussed during the project and it will depend on the number of cores in the webserver. It will be checked whether executing the meshing in the web server will overwhelm it in terms of performance freezing the client views.
3) It will allow two or more users to share the same visualization and mesh modifications of the same 3D model hosted on the server. For instance, the reader can imagine two doctors remotely located sharing a patient 3D model and discussing online the best treatment.
4) Another benefit of the distributed architecture is that it can visualize the data where it is generated, eliminating the need to move the data. It can be envisaged a set-up where someone has a 3D printer factory with dozens of devices, all of them uploading their models to a server as an ordinary procedure. In this scenario, the models are in one server and different users can visualize them without moving files around.
5) Modular design that allows offloading meshing operations to an HPC cloud (transition from interactive to batch mode). This design also addresses on and off-premises workflow. The system is not bound to be only a cloud-based solution and is open to new paradigms such as

Edge and Fog computing (leveraging the computing capabilities within a local network to carry out computation tasks that would ordinarily have been carried out in the cloud [3]).

6) If the client runs on the web (being this one option of the project and another one to run on a native desktop app) the system could be used from a mobile, touch-friendly interface without strong processing power.
7) Security: having the data processing in the backend can ensure the code is not bad intentionally modified and your data is not exposed to the client.
8) Ease of updating libraries and configurations. The client does not need to install any software if she/he uses the web client option (not in the case of using ParaView client desktop app).

So, the idea is that the client-side just handles the file browsing (hosted in the server), model viewer, and mesh creation and processing interactivity, leaving the heavy lifting to the backend. In this system, we will study the meshing functionality performance and analyze which is the best process topology configuration for a case study, i.e, problem size.

In **Figure 3** the reader can see an overview of the solution architecture. On the right side of it, the web client is shown. The client can be running on a PC, tablet, or mobile phone. It is using the **VTK.js** framework to develop the views and interactions and **wslink** to send and receive commands to the backend. In the middle of the drawing lays the **webserver** and **ParaView** backend. Its responsibilities are rendering the 3D models and serve the web pages. Finally, on the left, the HPC cloud is shown. The HPC cloud cluster is in charge of performing a distributed calculation of the meshing functionality in their nodes.

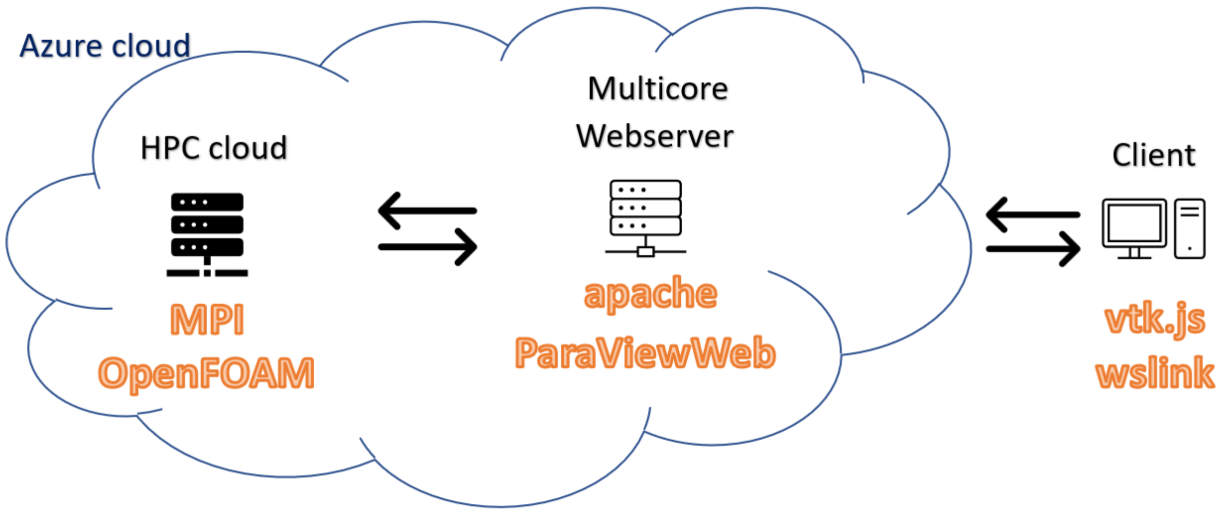


Figure 3: Architecture overview

There is a wide range of libraries for doing the different proposed tasks. During the project, active monitoring of the different options has been performed, being open-minded, and changing rails if better options are found during the lifetime of the project. The architecture is composed of the below component libraries:

1) VTK.js and wslink libraries for the web client option.
2) ParaView desktop app connected to a server for the native client option.
3) OpenFOAM on the HPC server for the meshing operations.
4) ParaViewWeb with Python interface backend in the server for the 3D model visualization.
5) Message Passage Interface (MPI) to parallelize the computation.
6) The backend is hosted in the Azure cloud provider.

2.2. Out of scope

To focus the FMP´s effort on the HPC, meshing, and scientific visualization, the system will not be multi-user. If two or more users access the web page, they will see the same in their browsers. In other words, the apache web server will not instantiate a ParaView server per user; in fact, it will not instantiate any ParaView server and it will count on that it is already running. It will not have user logins, but we will have only ParaView server running and the apache server will just connect to this one. If more than one user is connected, all of them will visualize the same.

The project won't either do fluid temporal simulations limiting it to meshing and 3D visualization.

2.3. State-of-the-art

There are a lot of 3D scientific visualization toolkits of all sorts available: desktop native application, web-browser running in the client, client-server rendering in the server, client-server rendering in the client, small dataset models or mesh processing in the client. But there are not many examples of solutions that fulfill the aim of this project, as said in the description, a customizable scalable HPC 3D scientific visualization and mesh generator with processing facilities of large datasets in a client-server architecture. What makes it unique is the mesh generator and edition part.

For an overview of the different applications, a very basic test is being realized consisting of loading two STereoLithography (STL) files of 250MB in total and an STL of 142MB (in case the app was not able to load the former). The test is checking if the application can load it, if it uses GPU, the memory consumption on the client-side, and general performance check (it can be rotated, zoomed, and spaned smoothly).

The files used for the test are:

- SerpentineMerchant_SPLIT_05.stl: 108MB (**Figure 4**)
- SerpentineMerchant_SPLIT_01_A.stl: 142MB (**Figure 5**)

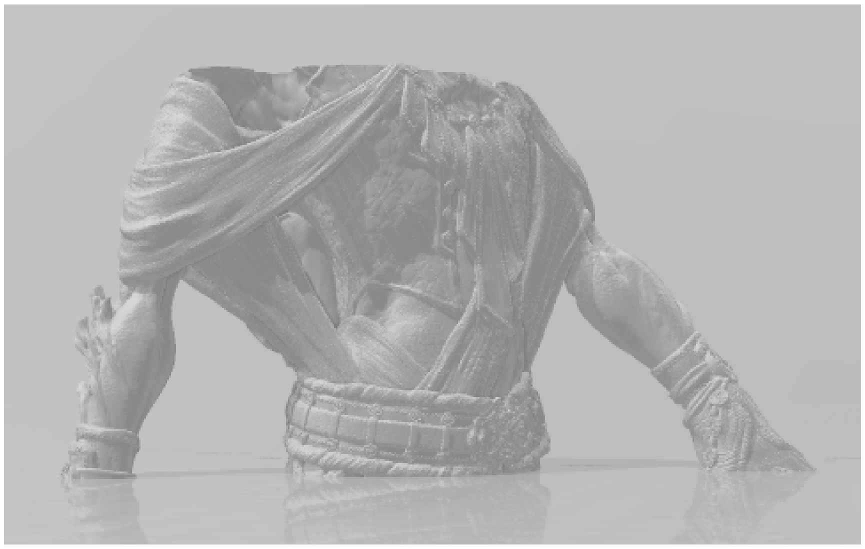


Figure 4: SerpentineMerchant_SPLIT_05.stl

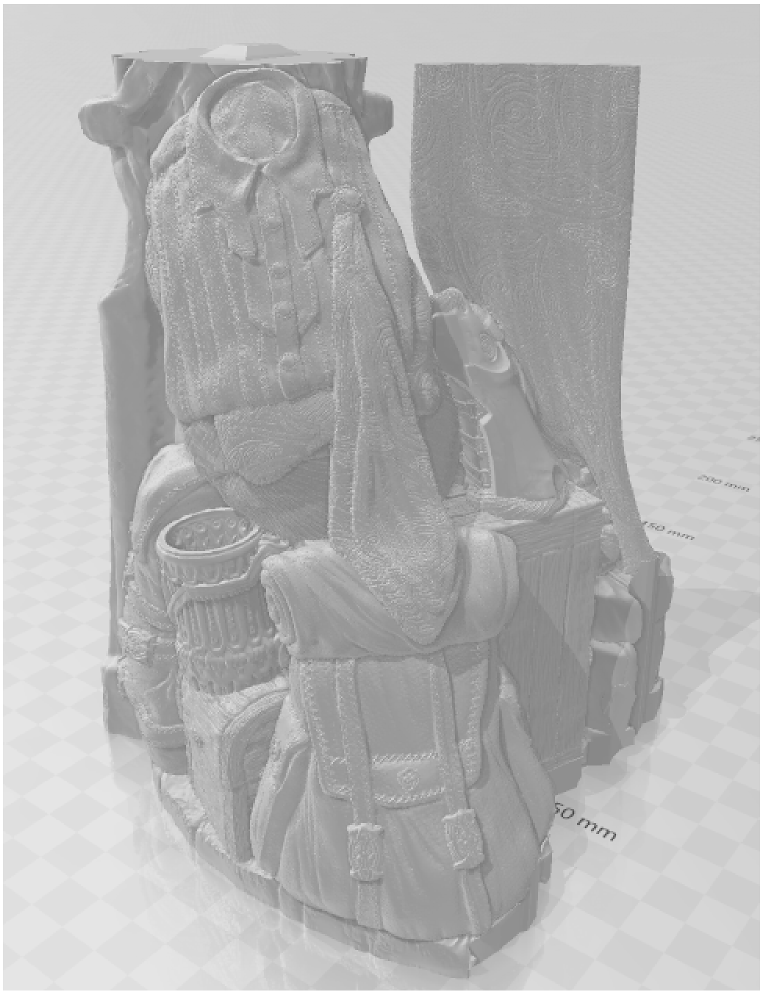


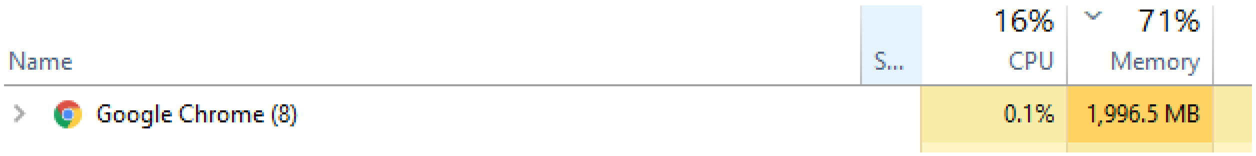
Figure 5: SerpentineMerchant_SPLIT_01_A.stl

and they have been downloaded from https://cults3d.com/es/modelo-3d/arte/serpentine-merchant-bendansie.

Find below a list of related work:

1) **MeshLabJS**: A JavaScript, client-side, mesh processing tool. MeshLabJS is built using, for all the mesh processing tasks, the C++ vcg library compiled into asm.js using emscripten and relying on three.js for rendering. MeshLabJS is developed by the Visual Computing Lab of the Institute of Information Science and Technologies - Italian National Research Council (ISTI - CNR).

MeshLabJS was tried and it resulted in memory errors when added a big dataset (a couple of STL files with 250MB in total). As said, everything is run on the client. Loading an STL of 142MB on disk turn out in consuming 1.9GB of Memory in the browser and it is using the GPU to accelerate the rendering (see **Figure 6**):

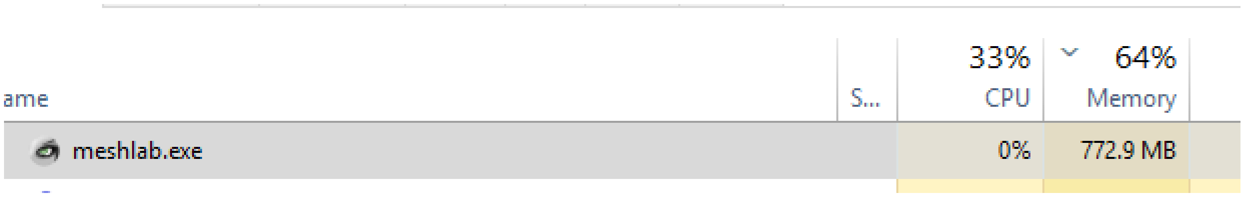


Name	S...	16% CPU	71% Memory
Google Chrome (8)		0.1%	1,996.5 MB

Figure 6: MeshLabJS performance

So MeshLabJs is not a good option for either large datasets and/or in devices with limited processing power. There were not options either to create a mesh from a point cloud. MeshLabJs has filters for modifying an existing mesh.

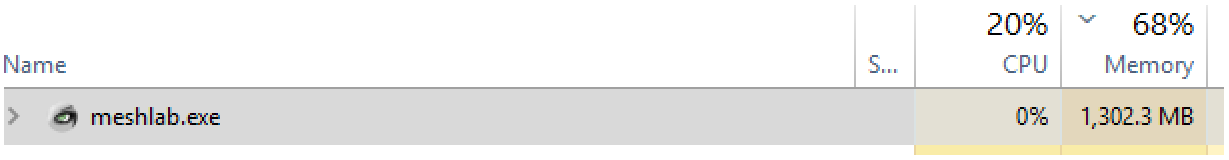
2) **Meshlab**: native desktop application version of MeshLabJS. Adding the same STL of 142MB mentioned in MeshLabJS turn out to consume 772MB in MeshLab, significantly less than MeshLabJs (see **Figure 7**):



ame	S...	33% CPU	64% Memory
meshlab.exe		0%	772.9 MB

Figure 7: MeshLab performance test 1

MeshLab was able to load the two STL files of 250 MB that were not possible with MeshLabJs consuming 1.3GB of RAM (see **Figure 8**).



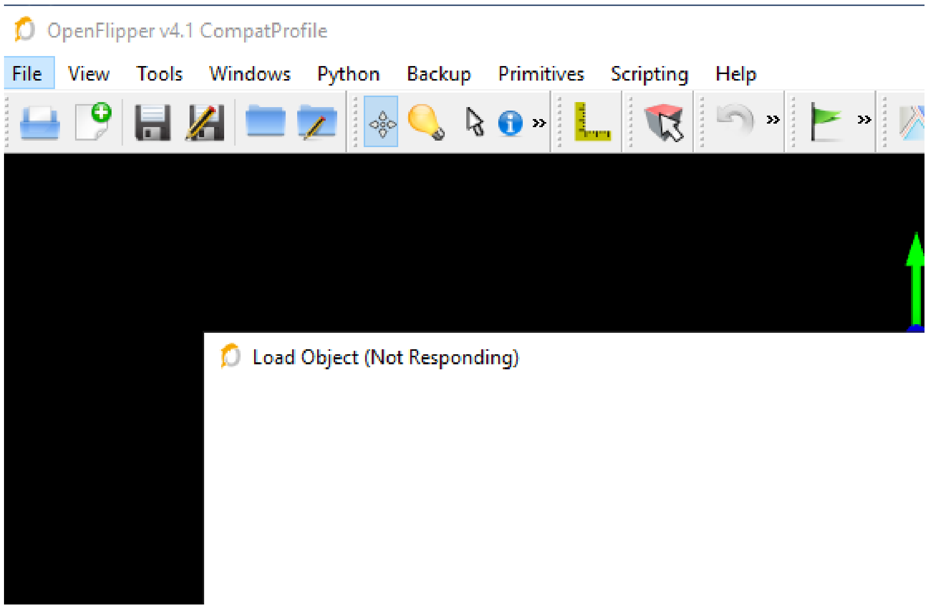
Name	S...	20% CPU	68% Memory
meshlab.exe		0%	1,302.3 MB

Figure 8: MeshLab performance test 2

Meshlab desktop performs better than MeshLabJS being the advantage of MeshLabJs the lack of installation of software in the client computer.

3) **OpenFlipper** [4] is an OpenSource multi-platform application and programming framework designed for processing, modeling, and rendering of geometric data. Currently, supported platforms are Windows, MacOS X, and Linux. Using OpenFlipper a variety of file formats are supported (off, obj, ply, ...). OpenFlipper provides a highly flexible interface for creating and testing its geometry processing algorithms. Basic functionality like rendering, selection, and a lot of user interaction is provided by the system which significantly reduces the coding effort for developing new geometry processing algorithms.

It has a Desktop interface. When trying to open a file several times with version v4.1 it consistently failed (see **Figure 9**):



Load Object (Not Responding)

Figure 9: OpenFlipper performance test 1

On the other hand, OpenFlipper works ok when creating a primitive (a natively created object) as can be seen in **Figure 10**:

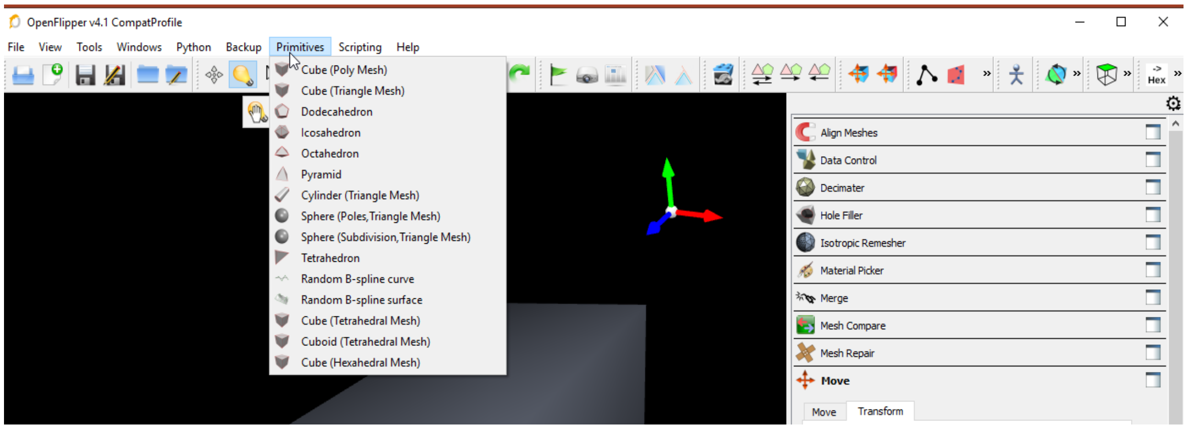
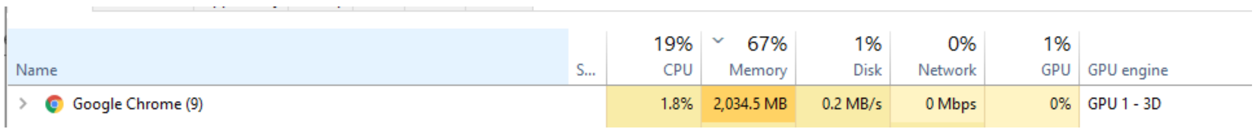


Figure 10: OpenFlipper performance test 2

4) **ParaView Glance** [5] is a web-based visualization application built on VTK.js. It's both desktop- and mobile-ready. It demonstrates the power of VTK web capabilities and can be customized to a variety of specific applications.

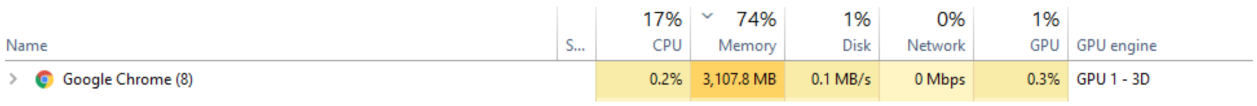
The rendering is done in the client as MeshLabJS. Adding the same STL of 142MB mentioned in MeshLabJS turns out to consume 2GB in MeshLab and it is using the GPU to accelerate the rendering (see **Figure 11**).



Name	S...	19% CPU	67% Memory	1% Disk	0% Network	1% GPU	GPU engine
Google Chrome (9)		1.8%	2,034.5 MB	0.2 MB/s	0 Mbps	0%	GPU 1 - 3D

Figure 11: ParaView Glance performance test 1

ParaView glance was able to load the two STL files of 250 MB that were not possible with MeshLabJs consuming 3GB of RAM (**Figure 12**).

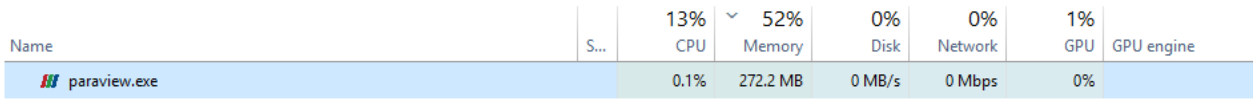


Name	S...	17% CPU	74% Memory	1% Disk	0% Network	1% GPU	GPU engine
Google Chrome (8)		0.2%	3,107.8 MB	0.1 MB/s	0 Mbps	0.3%	GPU 1 - 3D

Figure 12: ParaView Glance performance test 2

5) **ParaView desktop**: it is a native desktop application. ParaView desktop leverages ParaView's design with dynamic user interface (UI) generation based on Proxy definitions. Relying on ParaView's design has both benefits and drawbacks. The benefits are that ParaView end-users are familiar with the application use and its overall organization. However, for beginners or focused end-users, the dynamic UI may not be as useful for these end-users.

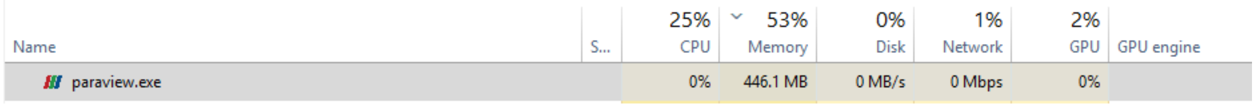
   Adding the same STL of 142MB mentioned in MeshLabJS turns out to consume 272MB in the ParaView desktop and it is not using the GPU to accelerate the rendering (**Figure 13**).



Name	S...	13% CPU	52% Memory	0% Disk	0% Network	1% GPU	GPU engine
paraview.exe		0.1%	272.2 MB	0 MB/s	0 Mbps	0%	

Figure 13: ParaView desktop test 1

ParaView desktop was able to load the two STL files of 250 MB that were not possible with MeshLabJs consuming 446MB of RAM (**Figure 14**).



Name	S...	25% CPU	53% Memory	0% Disk	1% Network	2% GPU	GPU engine
paraview.exe		0%	446.1 MB	0 MB/s	0 Mbps	0%	

Figure 14: ParaView desktop test 2

6) **ParaView Visualizer** [6] provides a web user interface to ParaView in the browser. Visualizer is a core web offering, and Kitware has been updating it throughout its evolving use of web technologies.

There is not an available URL to test ParaView Visualizer. People interested in trying have to download it and manually configure it. It uses a client-server architecture.

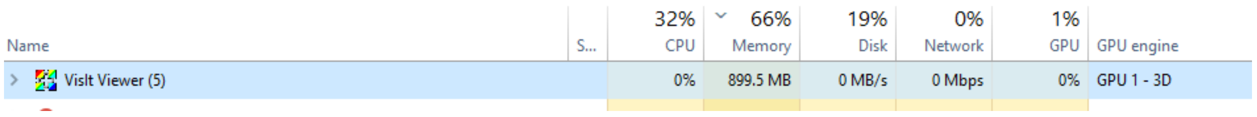
7) **ParaView Lite** [7] is a web-based client application that connects to a ParaView server to perform data processing and visualization. It is designed to be customizable in terms of UI and visualization workflow.

ParaView Lite is a replacement of ParaView Visualizer with the customizable, tailored UI available. As Evoker is a fork of ParaView Lite, the reader will find a detailed explanation in Section 3.

8) **VisIt** [8]: VisIt is an Open Source, interactive, scalable, visualization, animation, and analysis tool. From Unix, Windows, or Mac workstations, users can interactively visualize and analyze data ranging in scale from small (<101 cores) desktop-sized projects to large (>105 core) leadership-class computing facility simulation campaigns. Users can quickly generate visualizations, animate them through time, manipulate them with a variety of operators and mathematical expressions, and save the resulting images and animations for presentations. VisIt contains a rich set of visualization features to enable users to view a wide variety of data including scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured, adaptive and unstructured meshes. Owing to its customizable plugin design, VisIt is capable of visualizing data from over 120 different scientific data formats.

VisIt is a desktop native app.

Adding the same STL of 142MB mentioned in MeshLabJS turns out to consume 900MB in Visit and it is using the GPU to accelerate the rendering (**Figure 15**).



Name	S...	32% CPU	66% Memory	19% Disk	0% Network	1% GPU	GPU engine
VisIt Viewer (5)		0%	899.5 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D

Figure 15: VisIt performance

9) **3DHOP (3D Heritage Online Presenter)** [9] is an open-source framework for the creation of interactive Web presentations of high-resolution 3D models, oriented to the Cultural Heritage (CH) field. *3DHOP* target audience ranges from museum curators with some IT experience to experienced Web designers who want to embed 3D content in their creations, from students

in the CH field to small companies developing web applications for museums and CH institutions. It is not suitable for this FMP.

10) **GMSH** [10]: GMSH is an open-source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light, and user-friendly meshing tool with parametric input and advanced visualization capabilities. GMSH is built around four modules: geometry, mesh, solver, and post-processing. The specification of any input to these modules is done either interactively using the graphical user interface, in ASCII text files using GMSH's scripting language (.geo files), or using the C++, C, Python, or Julia Application Programming Interface (API).

11) **SimScale** [11]: holistic cloud-based simulation software, from preprocessing to postprocessing through simulation. The full version requires the payment of a license. SimScale is generalistic in simulations: heat transfer, solid mechanics, and fluid dynamics. It cannot be customized for a specific field as Evoker.

12) **Ingrid Cloud** [12]: cloud-based CFD software specialized in wind simulations. The full version requires the payment of a license. It cannot be customized for a specific field as Evoker

Figure 16 shows a cases summary of the results found:

Application	250MB file memory	142MB file memory	GPU	Rendering	Performance
MeshLabJS	Error	2GB	Yes	Client	Medium
MeshLab	1.3GB	0.7GB	Yes	Client	Good(loading time is slow)
OpenFlipper	Error	Error	Error	Client	Bad
ParaView Glance	3.1GB	2GB	Yes	Client	Medium
ParaView Visualizer	Not available	Not available	Not available	Server	Not available
ParaView desktop	0.44GB	0.27GB	Yes	Client	Good
ParaView Lite	Not available	Not available	Not available	Server	Not available
VisIt	0.7GB	0.9GB	Yes	Client	Medium(not friendly)
3DHOP	Not available	Not available	Not available	Client	Not available
GMSH	Not available	Not available	Not available	Client	Not available

Figure 16: State-of-the-art comparison results table

All in all, the studied solutions do not match the functionality of the system that this project is aiming at. Applications and libraries from Kitware (VTK and ParaView developing and consulting company) seem to have good quality and are a good starting point to achieve the project goals. Nonetheless, it will require a good amount of dedication and skills (JavaScript, C++, web services, software architecture, HPC, OpenFOAM, meshing …) to complete this project. That is why Kitware offers its commercial consultancy and R&D services.

2.4. Training

Find below a list of areas where the student will grow his skills/knowledge:

1) Client-Server architecture, Ubuntu VM and web server infrastructure setup, firewall, network, and Linux configuration.
2) SSH and key pair access.
3) Azure cloud.
4) VTK, VTK.js, ParaView.
5) OpenFOAM.
6) HPC subject from UOC Computational and Mathematical Engineering.
7) Python.
8) JavaScript frameworks (Angular, VUE) to modify Kitware example apps to add mesh creation.

2.5. Technologies

In this subsection, the reader will be introduced to the technologies used in this project.

2.5.1. VTK core (C++)

The Visualization Toolkit is an open-source software system for 3D computer graphics, image processing, and scientific visualization. VTK is distributed under the OSI-approved BSD 3-clause License. It supports a wide variety of visualization algorithms and advanced modeling techniques, and it takes advantage of both threaded and distributed memory parallel processing for speed and scalability, respectively [13]. VTK is designed to be platform agnostic. This means that it runs just about anywhere, including on Linux, Windows, and Mac; on the Web; and on mobile devices [13].

The core of VTK is a class library written in C++. To build an application based on VTK combining different objects from this library will be needed. VTK was originally part of a textbook [14]. Will Schroeder, Ken Martin, and Bill Lorensen were three graphics and visualization researchers who wrote the book and companion software on their own time, beginning in December 1993, with legal permission from their then-employer, GE R&D. The motivation for the book were to collaborate with other researchers and develop an open framework for creating leading-edge visualization and graphics applications [13].

In 1998 Ken and Will left GE and founded Kitware that is currently the company that supports it. Kitware offers custom solutions, collaboration, training, and support for those companies that want to integrate VTK into their solutions.

VTK is a huge library that has a considerable learning curve. They have this portal [15] to submit questions regarding support, issues, feedback.

2.5.2. VTK in the web (VTK.js)

Web technologies are evolving constantly. As so it is difficult to port a library that has decades of history like c++ VTK to it; by the time someone has ported a functionality to the web, maybe its technology has evolved so much that you enter the risk of being obsolete. So, there are two approaches for using VTK from the browser: **VTK.js** and **vtkWeb**.

2.5.2.1. VTK.js

VTK.js is the counterpart of the C++ VTK core in the web browser realm.

VTK.js is a JavaScript library available for scientific visualization in the browser. The library is available via NPM or unpkg.com CDN so it can directly be imported as a script tag inside your Web page [15]. Its implementation consists of an ES6 JavaScript class library that can be integrated into any web application. It adapts the VTK structure and expertise to bring high performance rendering into the browser.

VTK.js is a re-implementation of VTK/C++ in JavaScript that leverages WebGL and focuses on geometry rendering and volume rendering. On top of its rendering aspect, VTK.js offers the same pipeline infrastructure as its C++ father. While VTK.js embraces JavaScript, it tries to keep the same vocabulary and application programming interface as VTK, so it remains familiar to VTK users. In terms of classes, VTK.js uses vtkPolyData and vtkImageData for its rendering pipeline. It also offers a handful of readers (e.g., VTP, VTI, STL, OBJ, etc.) and filters [16]. An example application of VTK.js: ParaView lite [7].

2.5.2.2. vtkWeb

vtkWeb drives the C++ implementation of VTK through a web server via its Python wrapper. As a full package, it can leverage the complete input/output, data processing, and rendering capabilities of VTK [16].

Based on its infrastructure, vtkWeb allows a front-end client and a server to communicate. In particular, it enables method calls (RPCs) and publishes/subscribes types of communication. On top of the generic communication infrastructure, vtkWeb provides a protocol for vtkRenderWindow synchronization. As a result, interactions reflect both on the web page and on the server where the data lives. An example application of vtkWeb: ParaView Visualizer [6].

The decision for implementing Evoker was based on which current Kitware solution was closer and more suitable for Evoker. As explained before, we choose ParaviewLite; and ParaviewLite was based on VTK.js, so Evoker is using VTK.js as well

2.5.3. VTK based solutions evaluation

The first task that it was needed to do was to analyze all VTK web applications to decide which one was the most suited project to fork from. ParaView Glance, ParaView Visualizer, and ParaView Lite were analyzed. These applications have many similarities. ParaView Glance, as its name suggests, is designed to visualize a 3D object and has very few commands to modify the geometry. ParaView Glance is very basic. On the other side, ParaView Visualizer is very rich in features and it is kind of the web counterpart of the Desktop VTK application. ParaView Visualizer is very complex. Between these two applications lies ParaView Lite: it has a good amount of rendering options and a good enough amount of geometry modification commands for our project (Clip, Cut, etc.).

So, it was decided that this FMP will a fork from ParaView Lite.

2.5.4. ParaView

ParaView is an open-source, multi-platform data analysis and visualization application. ParaView is another toolkit different than VTK [18]. The goals of the ParaView team include the following:

- Develop an open-source, multi-platform visualization application.
- Support distributed computation models to process large data sets.
- Create an open, flexible, and intuitive user interface.
- Develop an extensible architecture based on open standards.

Paraview, as contrary to VTK, adds a layer of abstraction. This layer of abstraction is intended to support larger data as it uses several CPU nodes simultaneously employing HPC techniques. Both ParaView and VTK stem from the same underlying capabilities. ParaView adds a multi-machine setup to the equation. VTK is the core library that can be integrated into stand-alone client applications. ParaView has capabilities built-in (filters, rendering settings, …), prepared to be deployed in the server for client-server architecture systems [19].

So, Kitware recommends VTK.js for the client-side and ParaView for the server-side. This is the supported path and the most stressed and supported one. Having said that, as both libraries are based on the same technologies, VTK can also be used on the server and ParaView Web on the client. To do something like this should be based on whether the author knows and has experience in one of these and wants to avoid a learning curve. In the Evoker case, as it is new, VTK.js is going to be used for the client and ParaView for the server to minimize the errors and take maximum support of Kitware support forums.

2.5.5. Web Sockets

The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, messages can be sent to a server and receive event-driven responses without having to poll the server for a reply. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.

Before WebSocket discovery, it was difficult to have real-time two-way communication between a client and server in the browser. Somehow it was based on polling: client requesting if there was data available or if a command was completed every certain period. So, there were no events available that could be fired by the server. If the user has to rely on polling she/he can have two drawbacks: not having a good real-time experience or consuming a lot of network bandwidth by sending polling requests very often.

With WebSockets, the server can start up a communication, i.e. fire an event, with the client. This is a game-changer in Web Applications as it comes near to Native Desktop development.

2.5.6 Vue.js

Vue.js is a progressive framework for JavaScript used to build web interfaces and one-page applications. Not just for web interfaces, **Vue.js** is also used both for desktop and mobile app development with Electron framework [20].

Evoker uses Vue.js for its browser development.

2.5.7. Azure

Azure is a public cloud computing platform that lets you add cloud capabilities to your existing network through its platform as a service (PaaS) model or entrust Microsoft with all of your computing and network needs with Infrastructure as a Service (IaaS).

Evoker server is hosted in **Azure**. In the main server where we run our Apache Web Server and also our ParaView backend. Then, we have a scalable number of HPC nodes to run our OpenFOAM meshing functionality.

We also shared data between the different compute nodes with **Azure Files**.

Figure 17 shows an overview of Evoker technologies in each agent: client, server, and HPC cluster.

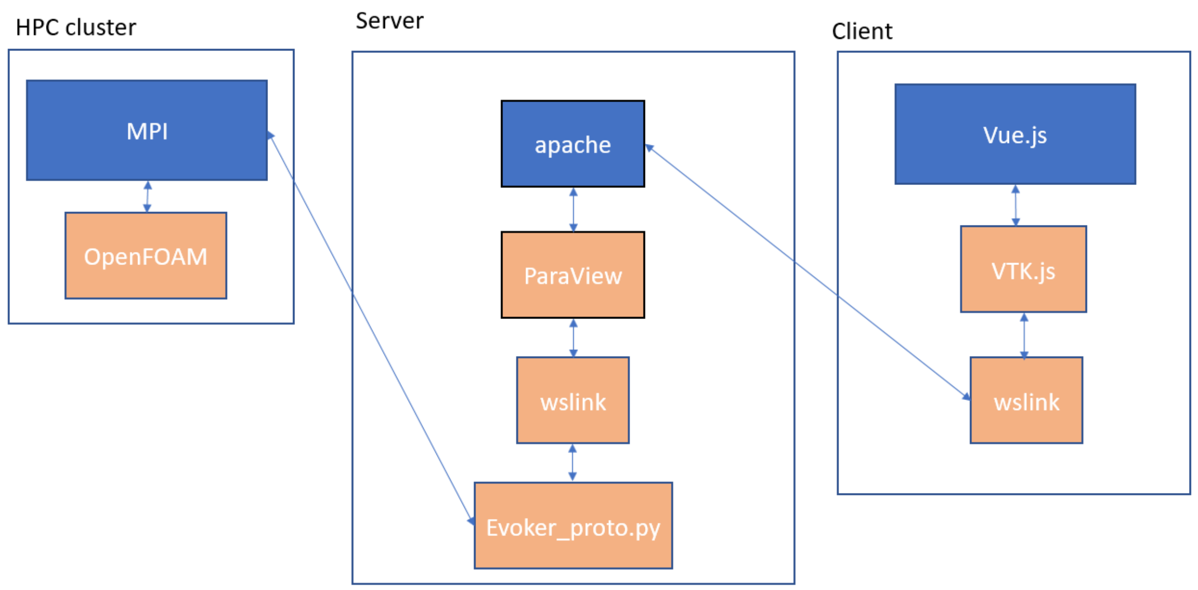


Figure 17: Overview of Evoker technologies

2.6. ICT tools

Find below a list of ICT tools used during the project:

- **Google Chrome**: it is the most popular internet browser with 64% of the market share.
- **Chrome Debugging tools**: Chrome DevTools is a set of web developer tools built directly into the Google Chrome browser. You can access it by pressing F12. It allows you to put breakpoints in your source code, see your traces, monitor the memory and network consumption.
- **Chrome Task Manager**: the Chrome web browser also has a task manager that helps the developer stop troublesome tabs and extensions.
- **Visual Studio Code**: IDE for editing source code.
- **Mendeley**: it is a software that allows keeping track of all bibliography used in a thesis and has an interface to import them to Word.
- **Microsoft Office 365**:
    - **Word**: text editor used to write this thesis.
    - **Excel**: spreadsheet software for data analysis and visualization.
- **MobaXTerm**: software for remote access with ssh, sftp, X11 forwarding.
- **Google Workspace**: usage of google docs to share documents between the author and the tutor.

2.7. Resources

The author has been developing this project on an HP Zbook 15 G3 laptop. It does not pay a lot of difference what computer you use as the heavy lifting is done on the server. The users need a good internet connection to have a good experience when rendering and doing zoom, pan and rotate to the 3D objects. The author has a fiber broadband connection with around 41 Mb/s of download speed and 24 Mb/s of upload speed.

2.8. Temporal planning

1) PEC1 work (around 1 month):
    a. Study of UOC tutorials about FMP: introduction to FMP, presentations, project management, scientific writing: 1 day
    b. State-of-the-art analysis: 7 days
    c. Description, goals, and scope of the project: 7 days
    d. Study the needed hardware infrastructure and solution performance: 7 days
    e. FMP planning: 7 days

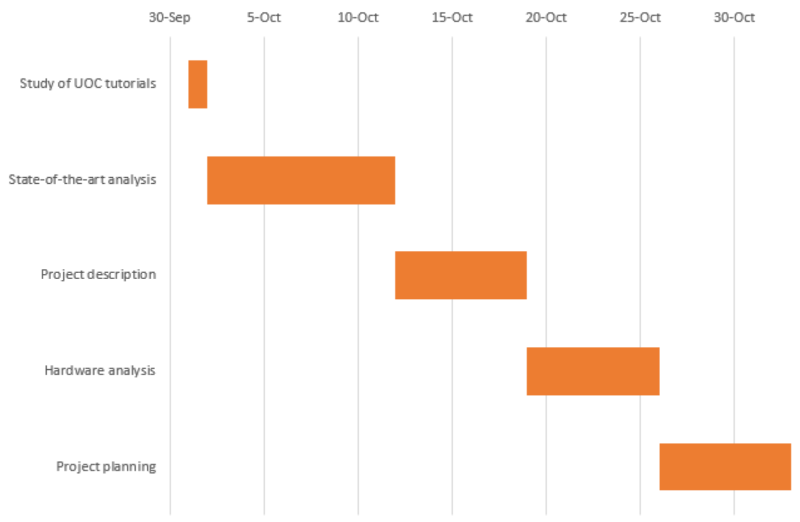


Figure 18:PEC1 planning

2) PEC2 work (around 3 months):
    a. Evaluate VTK-based visualization solutions related to the project: 14 days
    b. Implement an angular client app (named Evoker) emulating ParaView-Lite as a proof of concept: 40 days
    c. Evaluate and learn OpenFOAM meshing solutions: 21 days
    d. List the client pipe-line filter operations wanted in the GUI: 5 days

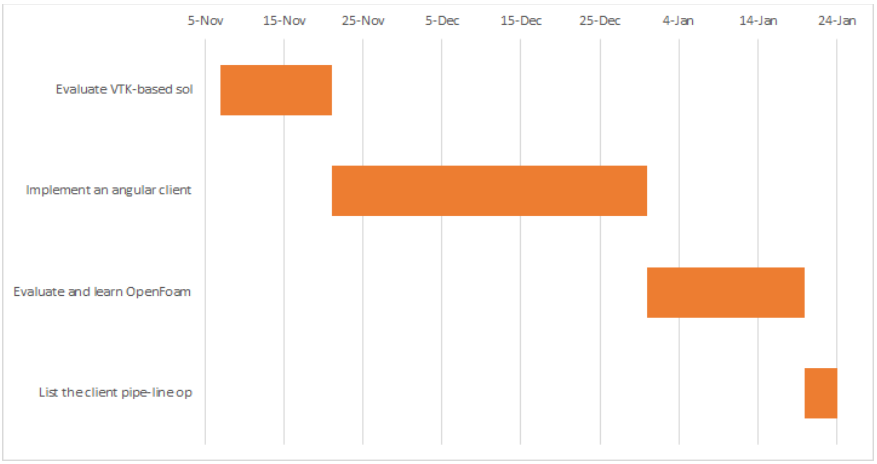


Figure 19: PEC2 planning

3) PEC3 work (around 3 months):

   a. Install OpenFOAM and MPI in the server: 7 days
   b. Extend Evoker client code to create and refine meshes in the server: 21 days
   c. Implement mesh editor filters in the client and server: 14 days
   d. Check the performance and bottlenecks of the system: 2 days
   e. Study different process topologies comparing performance: 28 days
   f. Testing, fine-tuning, and issue fixing: 7 days

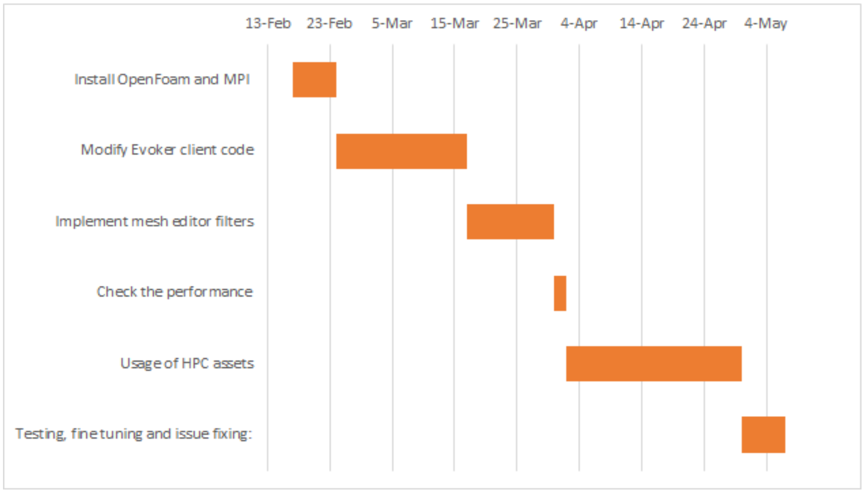


Figure 20: PEC3 planning

4) PEC4 work (around 1 month)

   a. Project documentation, user manual, and final report: 19 days
   b. Video recording: 3 days
   c. Presentation: 8 days

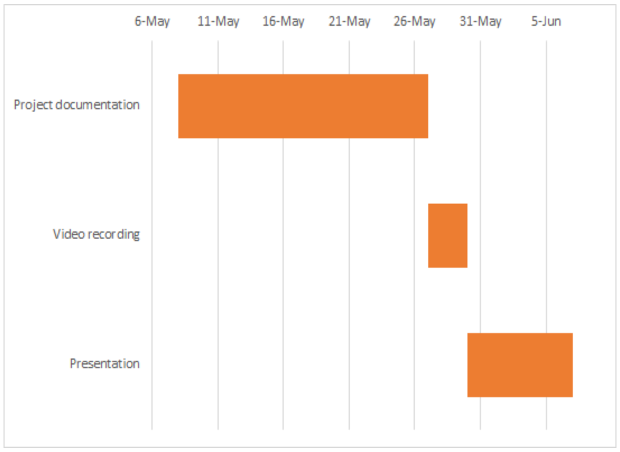
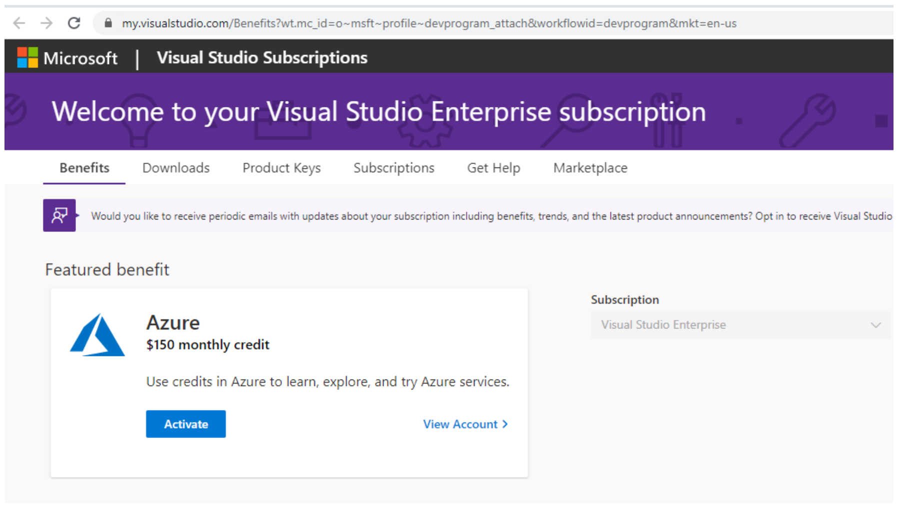


Figure 21: PEC4 planning

2.9. Project financials

As said this project has a client-server architecture with server-side rendering. Consequently, the server should be powerful enough. The server is hosted in Azure using the author MSDN subscription. The Azure scheme is of pay per use. Evoker is consuming credit when the servers are running. It has a $150 monthly credit (**Figure 22**):


my.visualstudio.com/Benefits?wt.mc_id=o~msft~profile~devprogram_attach&workflowid=devprogram&mkt=en-us

Microsoft | Visual Studio Subscriptions

Welcome to your Visual Studio Enterprise subscription

Would you like to receive periodic emails with updates about your subscription including benefits, trends, and the latest product announcements? Opt in to receive Visual Studio

Featured benefit

Azure
$150 monthly credit
Use credits in Azure to learn, explore, and try Azure services.

Subscription: Visual Studio Enterprise

Figure 22: MSDN subscription

For the main ParaView and Web Server, an Ubuntu 18.04-LTS with Standard E16ds_v4 (16 vcpus, 128 GiB memory) is being used.

For the HPC cloud, up to four Ubuntu 18.04-LTS with Standard E4ds_v4 (4 vcpus, 32 GiB memory) are being used.

To not consume credit, the VM's are being started and stopped for every development needed. Before adding Azure Files, the consumption was far below the monthly credit as can be seen in below November and December billings (**Figures 23 and 24**):

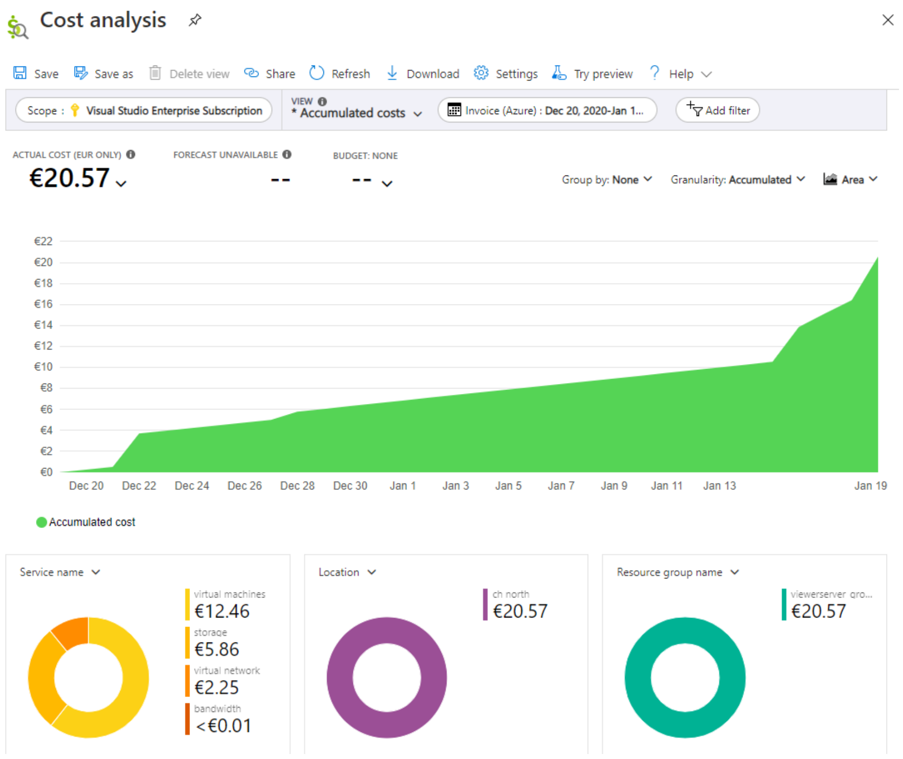
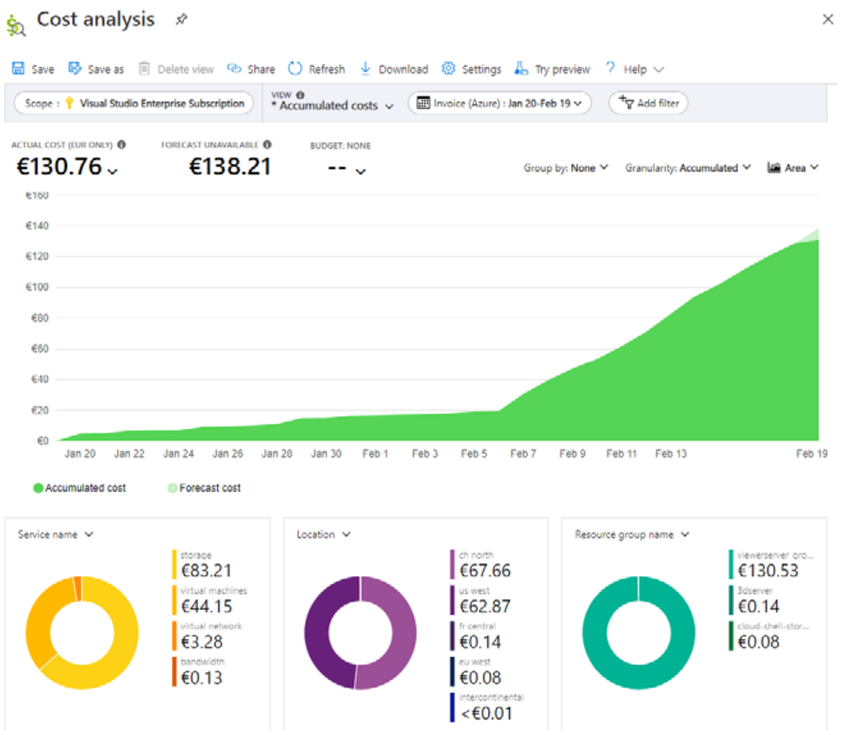


Figure 23: Cost analysis without Azure Files

But when storage for Azure Files to run OpenFOAM in parallel was added in different nodes the consumption increase substantially:



Scope : Visual Studio Enterprise Subscription

VIEW: * Accumulated costs

Invoice (Azure) : Jan 20-Feb 19

ACTUAL COST (EUR ONLY): €130.76

FORECAST UNAVAILABLE: €138.21

BUDGET: NONE: --

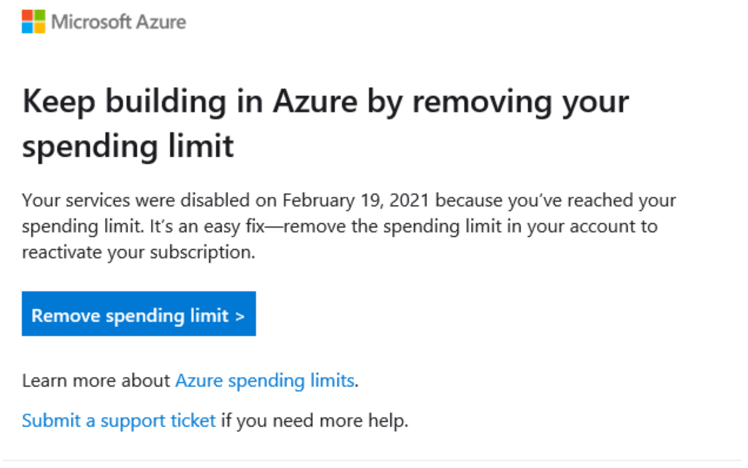
Service name	
storage	€83.21
virtual machines	€44.15
virtual network	€3.28
bandwidth	€0.13

Location	
ch north	€67.66
us west	€62.87
fr central	€0.14
eu west	€0.08
intercontinental	<€0.01

Resource group name	
viewerserver gro...	€130.53
3dserver	€0.14
cloud-shell-stor...	€0.08

Figure 24: Cost analysis with Azure Files

During the evolution of the project, the monthly credit was exceeded, and the account got disabled (**Figure 25**):



Microsoft Azure

Keep building in Azure by removing your spending limit

Your services were disabled on February 19, 2021 because you've reached your spending limit. It's an easy fix—remove the spending limit in your account to reactivate your subscription.

Remove spending limit >

Learn more about Azure spending limits.

Submit a support ticket if you need more help.

Figure 25: Spending limit reached

This put the project at risk, and it was looked at ways to reduce consumption. It was perceived that my **Files Shares** had a default provisioned capacity of 1000 GB and they were not needed. At this point, the author reduced them to 10% of the original size. Fortunately, when the billing period expires your account is re-enabled and you can continue using your Virtual Machines and Storage. Azure user has the option to provide a credit card when she/he consumes all her/his monthly credit.

3. Three-dimensional scientific visualization

In this section, the configuration and code development of the 3D visualization part of the project is explained.

3.1. 3D scientific visualization workflows

As with other solutions analyzed, Evoker will not upload files to the server from the web app (browser). It is expected that the files to visualize are already there (the user has uploaded by FTP, SCP, …). The folder where the data files are stored will be passed to the ParaView backend process with the data argument:

```
$ pvpython --force-offscreen-rendering ~/evoker/client/server/pvw-lite.py --data
~/3d_samples --port 49773
```

The client code connects to a VTK server with a known hardcoded IP static address behind the scenes. The reason for this hardcoded IP address is that this is just an academic project and that we want to ease the introduction of the web URL (avoid long typing for getting the same results). It is also possible to pass another IP address as a query parameter. So, if the user enters the IP address of the server in the browser without any parameter, the JavaScript code will connect through a web socket to the ParaView server with the same IP address and using port 49773. But the user can also pass a different socket to connect as a query parameter. So, we have these two options:

1) http://51.103.138.135:

   JavaScript code will connect to session URL = ws://51.103.138.135:49773/ws

2) http://51.103.138.135/?sessionURL=ws://51.103.135.214:49773/ws:

   JavaScript will connect to provided ParaView backend ws://51.103.135.214:49773/ws

Once the webserver and the VTK have been started you can enter the IP address of the webserver in any browser and start your 3D scientific visualization. The app interaction through buttons and menus is very easy. When the app has started there should not be any object to render. In the application, there are the concepts of files and datasets. Files are any ParaView supported file (obj, STL, etc.). Datasets are in-memory objects that ParaView has extracted from the files and converted to their internal format. So, the first thing the user needs to do is to add a file to the app. When you click the **Files** button the app will browse the folder configured in the server and will show the files and directories on the left side of the app (**Figure 26**):

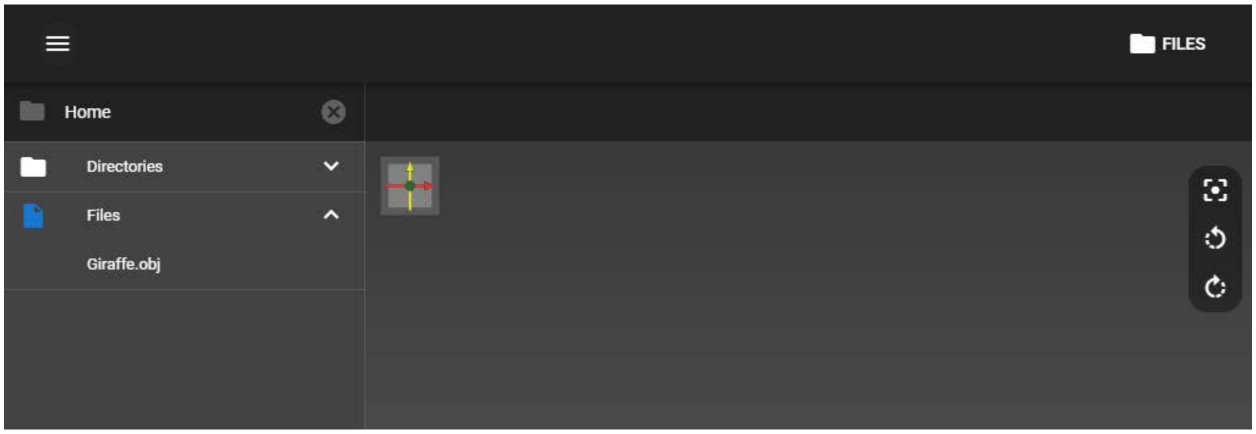


Figure 26: Files and Datasets in Evoker

Then the user can click any of the files and ParaView will generate a Dataset from it (**Figure 27**).

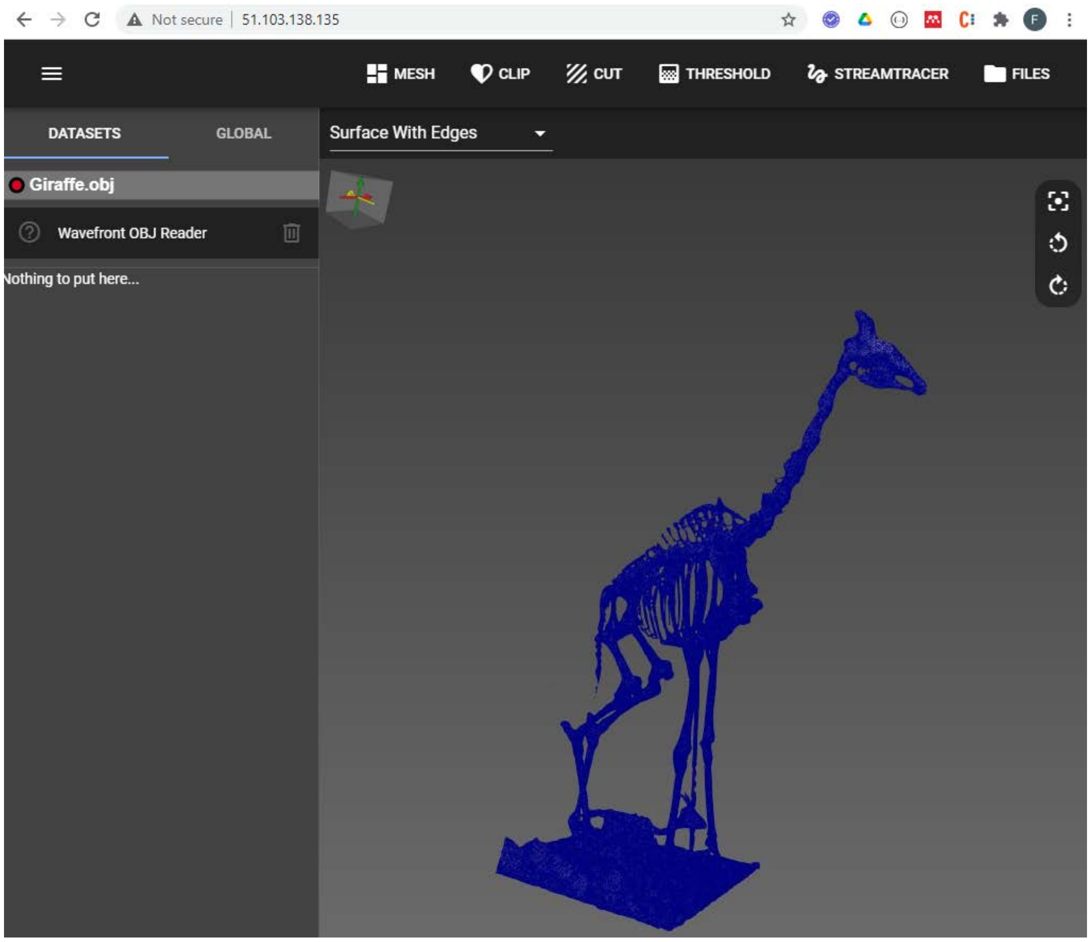


Figure 27: A rendered Dataset in Evoker

The user can change the rendering of the object through a combo box (**Figure 28**):

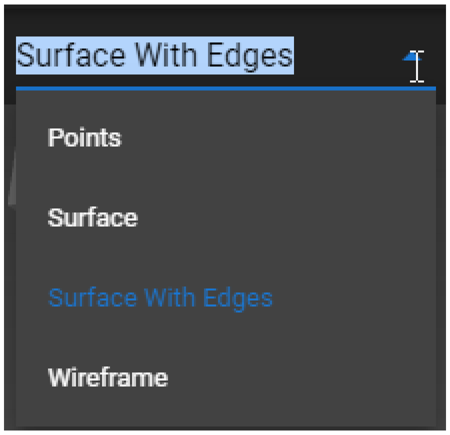


Figure 28: Changing representation in Evoker

Find below some of the operations the user can do with Evoker:

Clip: cutting the object and generating a new 3D object (**Figure 29**).

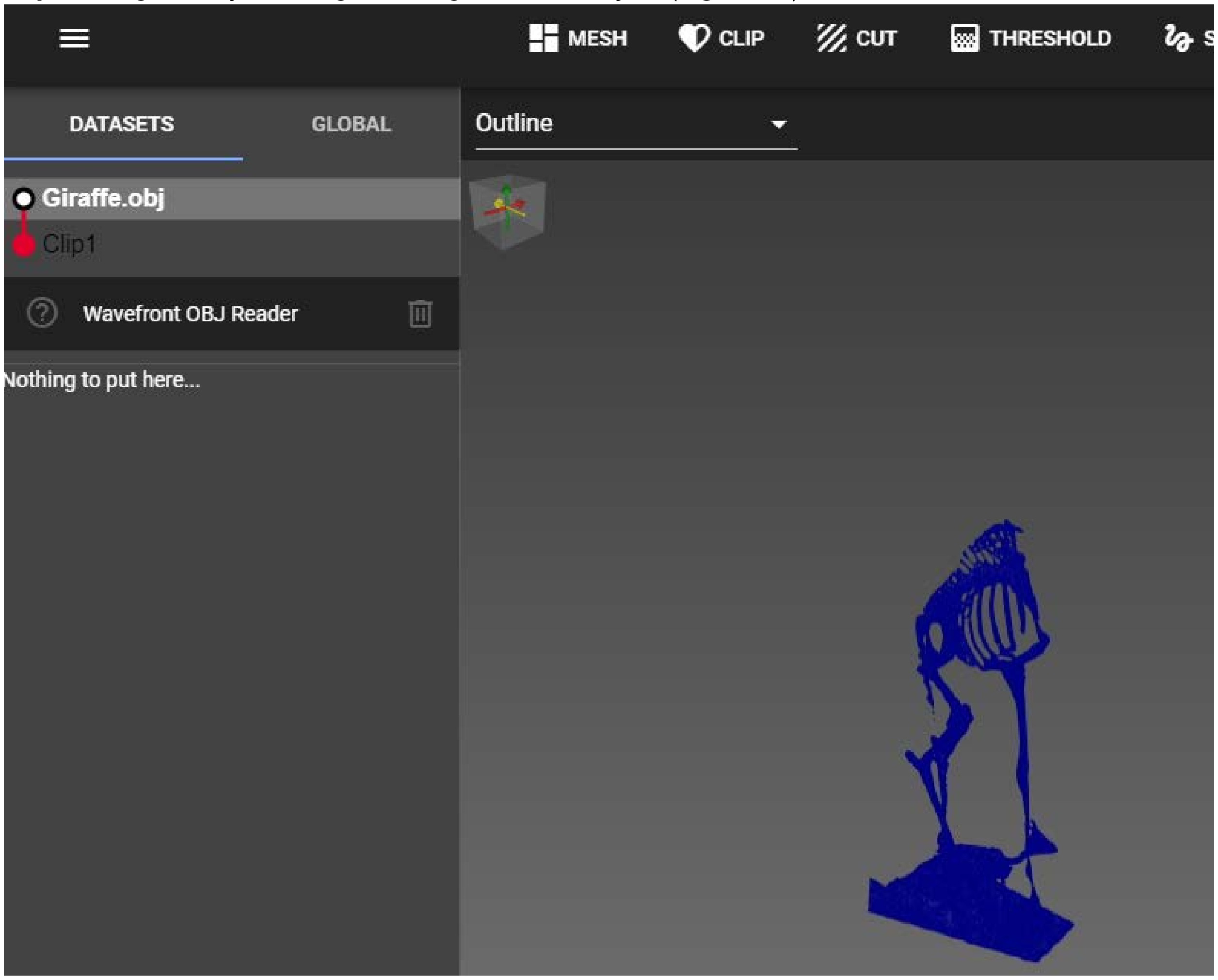


Figure 29: Clip command in Evoker

Cut: generates a 2D contour that is the intersection of the object with a plane (**Figure 30**).

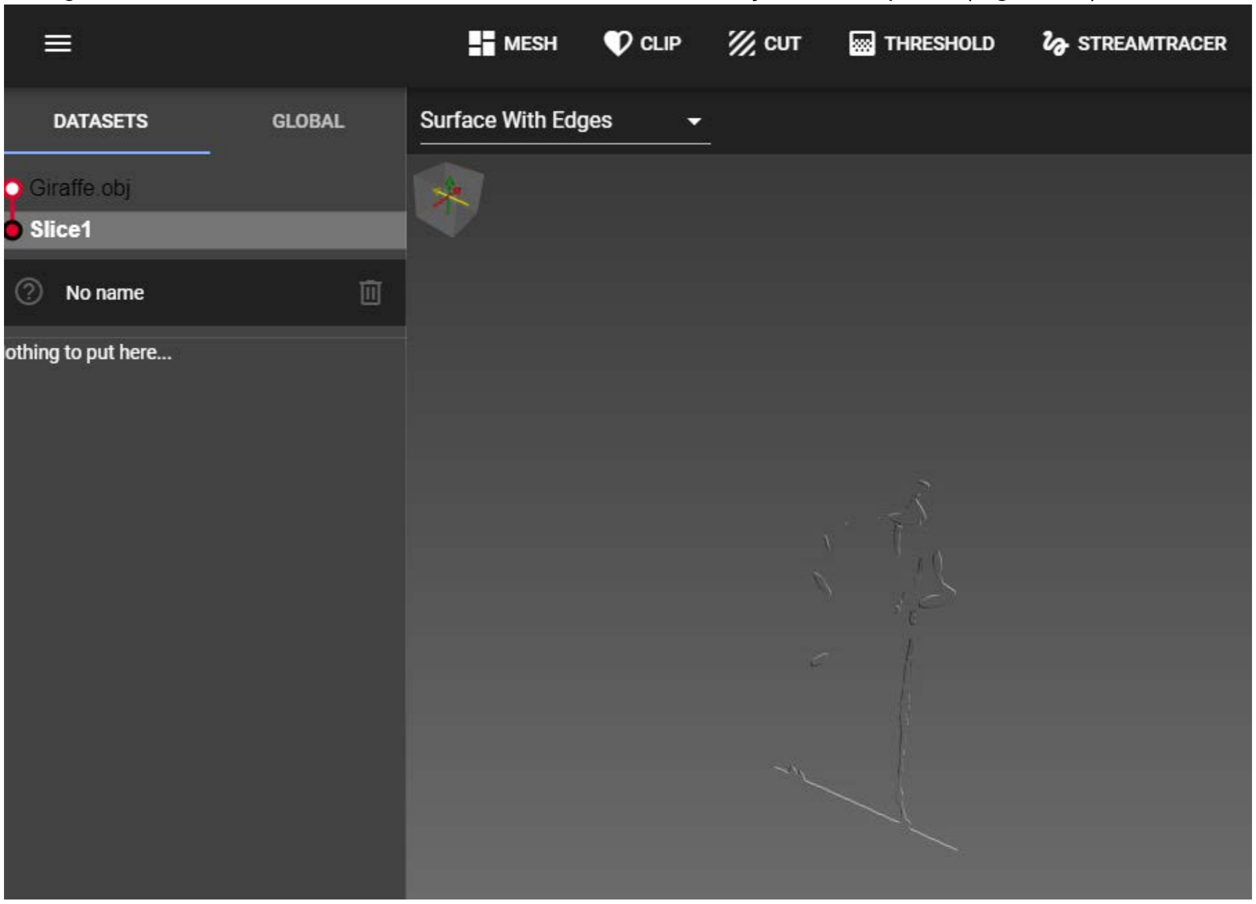


Figure 30: Cut command in Evoker

All these operations create a new dataset from the original. The user can show and hide any object in the list by clicking at the red circle at the left of its name in the Dataset menu (**Figure 31**):

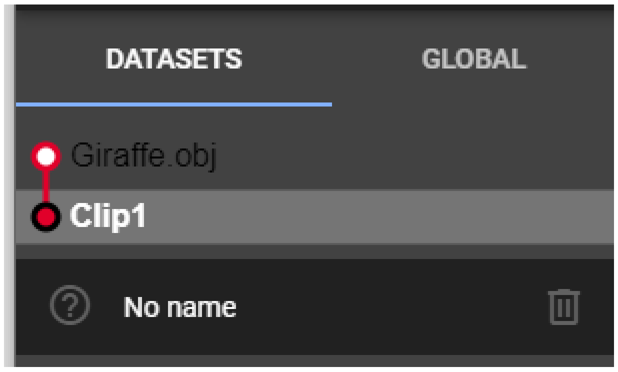


Figure 31: Dataset pipeline

The user can also delete a dataset with the **Trash** icon (**Figure 31**).

Once the object is rendered, it can be zoomed, panned, and rotated. The solution gives a good user experience with no noticeable delays (latencies) when doing these visualization operations.

3.2. Client code implementation

After three more weeks, the project had a customized version of ParaView Lite up and running having implemented the below changes with feedback from the instructor Sergio Iserte:

- Remove the Sources button (this was used to create cones, spheres, etc.).
- Remove line width, contrast, and color selectors.
- Limit object rendering to WireFrame, Surface, Surface with Edges and Points.
- Remove the About button.
- Remove the ParaView Lite logo.
- Remove Contour geometry commands.

So, discarding the work done in Angular and re-do it in Vue was a good decision.

ParaView Lite is implemented using the Vue.js JavaScript framework. As the author has some experience with Angular development, he thought about implementing the client-side (rendering and meshing in the browser) using this technology and use ParaView Lite as a reference for VTK backend interactions. It did progress well for about 3 weeks and a good prototype was implemented that was able to perform browsing of files hosted at the server and render them on the screen. However, it reached a point in which the rendered objects did not respond to Zoom, Pan, and Rotate and the author was unable to troubleshoot it. Help was requested from the Kitware company through their forums, but they do not have a ParaView application implemented with Angular. So, they told the requester that unfortunately, they could not help the Evoker project.

It was a tough moment for the Project as a good amount of time was invested with the Angular implementation and because the author feels comfortable with Angular and he had zero experience with Vue.js. But it was considered that VTK and ParaView codebase were huge and not very well documented (at least it requires a considerable learning curve to master it). Sooner or later more problems will appear with an Angular project interacting with the VTK backend, and it will not have supported by Kitware. So, it was decided to restart the implementation with Vue.js. A Vue book [21] was bought and an online course at Lynda [22].

3.2.1. Remove Sources button

In order to remove Sources button from the command toolbar, a change in file **src/modules/registerModules.js** was needed. Below code lines were removed:

```
import Sources from 'paraview-lite/src/modules/Sources';
import sourcesModule from 'paraview-lite/src/modules/Sources/module';
store.commit(
    'PVL_MODULES_ADD',
    Object.assign({}, sourcesModule, { component: Sources })
);
```

3.2.2. Remove line width, contrast, and color selectors

To remove these widgets from the GUI some lines of **src/components/core/RepresentationToolbar/template.html** needed to be removed. Find below the final code for this file:

```
<v-toolbar dense :class="$style.container">
  <v-layout row v-if="activeProxyData">
    <v-combobox
      :class="$style.comboboxFirst"
      dense
      single-line
      hide-details
      ref="representation"
      @input="blur('representation')"
      v-model="representation"
      :items="representationItems"
      label="Representation"
    />
  </v-layout>
  <div v-if="!activeProxyData" :class="$style.empty"></div>
</v-toolbar>
```

3.2.3. Limit object rendering to WireFrame, Surface, Surface with Edges and Points

To limit the object rendering to the required values, the file **src/components/core/RepresentationToolbar/script.js** needs to be edited. A filter was added to **representationItems** method as highlighted below:

```
    representationItems() {
      if (this.activeProxyData) {
        return this.activeProxyData.ui
          .find((prop) => prop.name === 'Representation')
          .values.filter(
            (value) =>
              value === 'Surface' ||
              value === 'Surface With Edges' ||
              value === 'Points' ||
              value === 'Wireframe'
          );
      }
      return [];
    },
```

3.2.4. Remove ParaView Lite logo and About Logo

To remove the Logo and the About Logo some lines needed to be removed from **src/components/core/RootNode/template.html**. Below you can find this code with commented lines (removed in the final project) highlighted:

```
    <v-tooltip bottom>
      <a
        slot="activator"
        href="#"
        v-on:click.prevent="toggleLanding"
      >
        <!-- <svg-icon
          :icon="iconLogo"
          height="52px"
          style="margin-top: 6px"
        /> -->
      </a>
      <span v-if="landingVisible" key="if-landingVisible">Go to app</span>
      <span v-else key="if-landingVisible">Back to landing page</span>
    </v-tooltip>
    <v-spacer />
    <action-modules v-if="!landingVisible" :smallScreen="smallScreen" />
    <v-btn
      v-if="errors.length"
      key="if-has-errors"
      :class="$style.toolbarButton"
      flat
      color="error"
      v-on:click="errorDialog = true"
    >
      <v-icon>{{ $vuetify.icons.pvLite.error }}</v-icon>
      <span>{{ errors.length }}</span>
      <span v-show="!smallScreen">error(s)</span>
    </v-btn>
    <!-- <v-btn :class="$style.toolbarButton" flat v-on:click="aboutDialog
= true">
      <!- - width of other icons in toolbar - ->
      <svg-icon icon="kitware-logo" width="24px" />
      <span v-show="!smallScreen">About</span>
    </v-btn> -->
  </v-toolbar>
```

3.3. Server set-up

In this subsection, the set-up of the server is explained. The first approach is by compiling the source code and then installing VTK binaries.

3.3.1. First approach: compiling ParaView from source

When someone starts approaching a huge, successful, and consolidated toolkit as VTK, she/he does not find a short and straight tutorial that brings you from point A to point B. Instead, he/she read a lot of online documentation not always connected and following a cohesive discourse. The engineer is not sure what article he reads on the internet would be his tailored solution for the problem he has in his hands. VTK and ParaView do not have an easy learning curve for new beginners.

So, one of the articles the author read about VTK said that if you want to use a client-rendering architecture with the data processing in the backend hosted on Ubuntu, you need to compile the source code. This is not an easy task as you will see.

Mainly tutorials: [23], [24], and [25] have been followed.

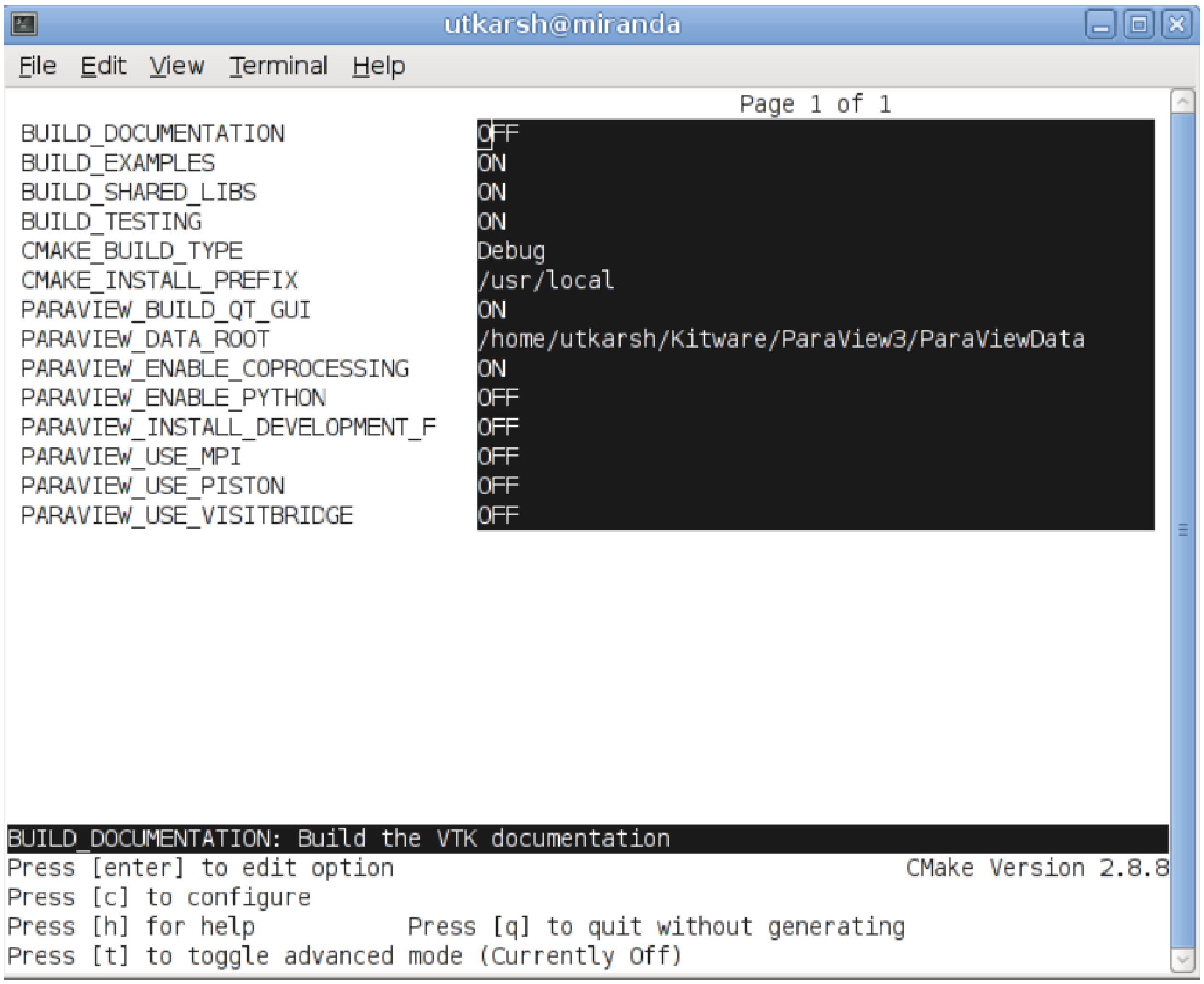
And these are the notes I took during the installation:

```
$ git clone --recursive https://gitlab.kitware.com/paraview/paraview.git

cmake:
   $ sudo apt remove --purge cmake
   $ hash -r
   $ sudo snap install cmake --classic

qt:
   $ sudo apt-get install qt5-default

install mpi:
   $ sudo apt-get update -y
   $ sudo apt-get install -y mpi

python:
   $ sudo apt update
   $ sudo apt install software-properties-common
   $ sudo add-apt-repository ppa:deadsnakes/ppa
   $ sudo apt update
   $ sudo apt install python3.8
   $ sudo ln -sf /usr/bin/python3.8 /usr/bin/python
   $ sudo apt install python3.8-venv python3.8-dev

   lib path: /usr/lib/x86_64-linux-gnu/libpython3.8.so
   $ sudo apt install python3-pip
   $ sudo pip3 install autobahn twisted
   $ sudo apt install python-autobahn

mesa:
   $ sudo apt-get update -y
   $ sudo apt-get install -y libosmesa6-dev
```

Before building ParaView you need to configure it with ccmake:

```
$ cd paraview_build
$ ccmake ../paraview
```

and a graphical window like this will appear (**Figure 32**):



```
                                                Page 1 of 1
 BUILD_DOCUMENTATION              OFF
 BUILD_EXAMPLES                   ON
 BUILD_SHARED_LIBS                ON
 BUILD_TESTING                    ON
 CMAKE_BUILD_TYPE                 Debug
 CMAKE_INSTALL_PREFIX             /usr/local
 PARAVIEW_BUILD_QT_GUI            ON
 PARAVIEW_DATA_ROOT               /home/utkarsh/Kitware/ParaView3/ParaViewData
 PARAVIEW_ENABLE_COPROCESSING     ON
 PARAVIEW_ENABLE_PYTHON           OFF
 PARAVIEW_INSTALL_DEVELOPMENT_F   OFF
 PARAVIEW_USE_MPI                 OFF
 PARAVIEW_USE_PISTON              OFF
 PARAVIEW_USE_VISITBRIDGE         OFF

BUILD_DOCUMENTATION: Build the VTK documentation
Press [enter] to edit option                               CMake Version 2.8.8
Press [c] to configure
Press [h] for help            Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

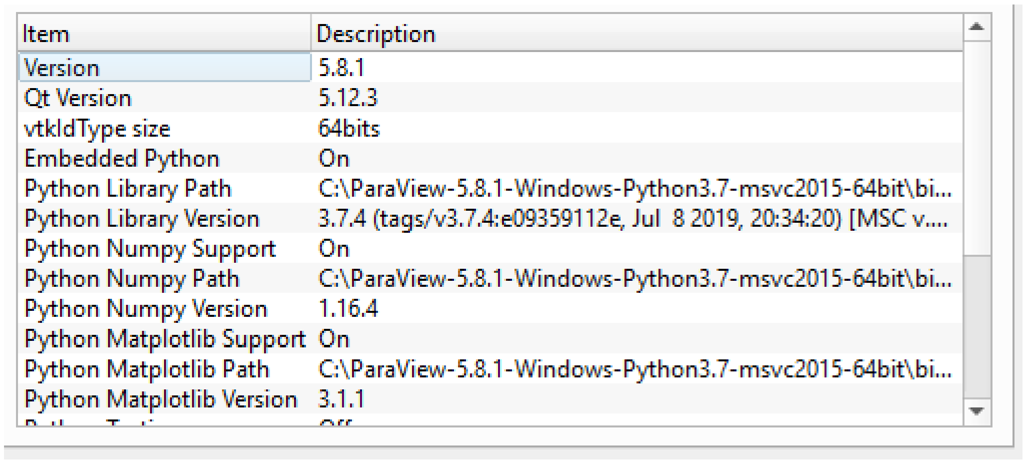
Figure 32: ParaView build GUI options

Where the engineer configures different options of the build.

After that you can go ahead and build it:

```
$ make -j 16
```

The building of ParaView from source has been a long process with lots of pitfalls in the way. Some of them are because it is a completely new subject to the author. But when finally, the author got it to make it work and try to connect it to the ParaView Desktop native app with version (**Figure 33**):



Item	Description
Version	5.8.1
Qt Version	5.12.3
vtkIdType size	64bits
Embedded Python	On
Python Library Path	C:\ParaView-5.8.1-Windows-Python3.7-msvc2015-64bit\bi...
Python Library Version	3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v....
Python Numpy Support	On
Python Numpy Path	C:\ParaView-5.8.1-Windows-Python3.7-msvc2015-64bit\bi...
Python Numpy Version	1.16.4
Python Matplotlib Support	On
Python Matplotlib Path	C:\ParaView-5.8.1-Windows-Python3.7-msvc2015-64bit\bi...
Python Matplotlib Version	3.1.1

Figure 33: ParaView desktop app version

An error mismatch appeared:

a) In the client (windows):

```
ERROR: In
C:\bbd\8fbebceb\build\superbuild\paraview\src\VTK\Parallel\Core\vtkSocketC
ommunicator.cxx, line 592
vtkSocketCommunicator (000001D7F58AAD50): Client/server version hash
mismatch.
ERROR: In
C:\bbd\8fbebceb\build\superbuild\paraview\src\Remoting\Core\vtkTCPNetworkA
ccessManager.cxx, line 333
vtkTCPNetworkAccessManager (000001D7EDDDB750):
```

```
Connection failed during handshake. vtkSocketCommunicator::GetVersion()
returns different values on the two connecting processes
(Current value: 100).
```

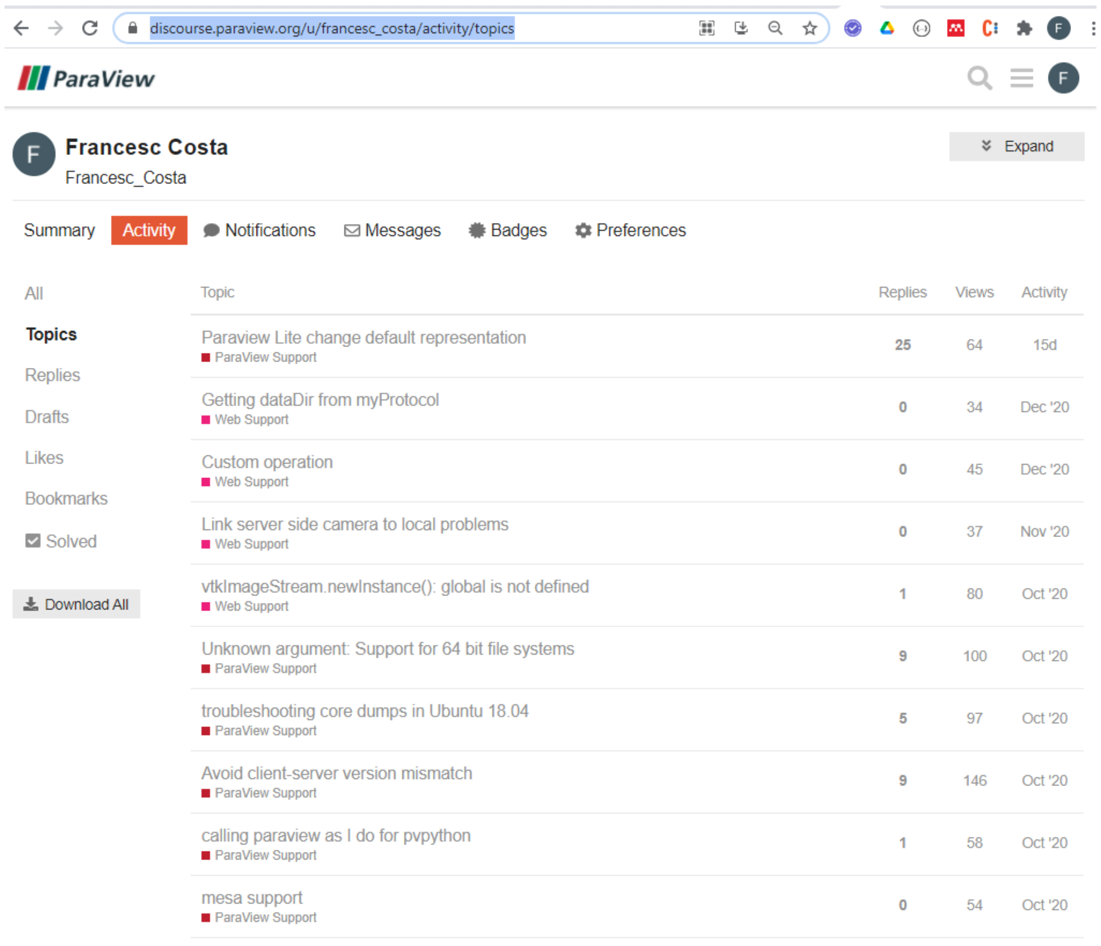
b) And in the ubuntu server:

```
( 282.183s) [pvserver ]vtkSocketCommunicator.c:481 ERR|
vtkSocketCommunicator (0x555a2fc3cde0): Client/server version hash
mismatch.
( 282.183s) [pvserver ]vtkTCPNetworkAccessMana:333 ERR|
vtkTCPNetworkAccessManager (0x555a2fb10ce0):
```

```
Connection failed during handshake. vtkSocketCommunicator::GetVersion()
returns different values on the two connecting processes
(Current value: 100).
```

I have submitted a lot of topics to the ParaView discourse page [26] as you can see in the below link and screenshot in **Figure 34**:

https://discourse.paraview.org/u/francesc_costa/activity/topics


discourse.paraview.org/u/francesc_costa/activity/topics

ParaView

Francesc Costa
Francesc_Costa

Expand

Summary | Activity | Notifications | Messages | Badges | Preferences

All
Topics
Replies
Drafts
Likes
Bookmarks
Solved

Download All

Topic	Replies	Views	Activity
Paraview Lite change default representation ParaView Support	25	64	15d
Getting dataDir from myProtocol Web Support	0	34	Dec '20
Custom operation Web Support	0	45	Dec '20
Link server side camera to local problems Web Support	0	37	Nov '20
vtkImageStream.newInstance(): global is not defined Web Support	1	80	Oct '20
Unknown argument: Support for 64 bit file systems ParaView Support	9	100	Oct '20
troubleshooting core dumps in Ubuntu 18.04 ParaView Support	5	97	Oct '20
Avoid client-server version mismatch ParaView Support	9	146	Oct '20
calling paraview as I do for pvpython ParaView Support	1	58	Oct '20
mesa support ParaView Support	0	54	Oct '20

Figure 34: Francesc Costa's posted topics in the ParaView forum

In the end, a support engineer from Kitware suggested to me that I do not need to compile ParaView from Source but installed pre-compiled binaries in Ubuntu.

3.3.2. Final approach: installing VTK binaries

For the Ubuntu 18.04-LTS server, the following steps were needed:

1) Install ParaView:
   a. `$ wget -O paraview.taz.gz "https://www.paraview.org/paraview-downloads/download.php?submit=Download&version=v5.8&type=binary&os=Linux&downloadFile=ParaView-5.8.1-osmesa-MPI-Linux-Python3.7-64bit.tar.gz"`

```
b. edit .profile and add:
 PATH="$HOME/ParaView-5.8.1-osmesa-MPI-Linux-Python3.7-64bit/bin:$PATH"
```

2) Set up web app:
    a. First install node v10.22.1
```
$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
$ sudo apt install nodejs
$ node --version
```
    b. Clone FMP code:
```
$ git clone https://github.com/FrancescSM/evoker.git
```
    c. Build client code
```
$ cd evoker\client
$ npm install
$ npm run build
```
3) Set-up apache web server:
    a. install
```
$ sudo apt install apache2
```
    b. copy dist
```
$ cd ~/evoker/client
$ mv /var/www/html /var/www/html_ori
$ sudo rm -rf /var/www/dist
$ sudo cp -r dist /var/www/dist
```
    c. restart apache
```
$ sudo /etc/init.d/apache2 restart
```

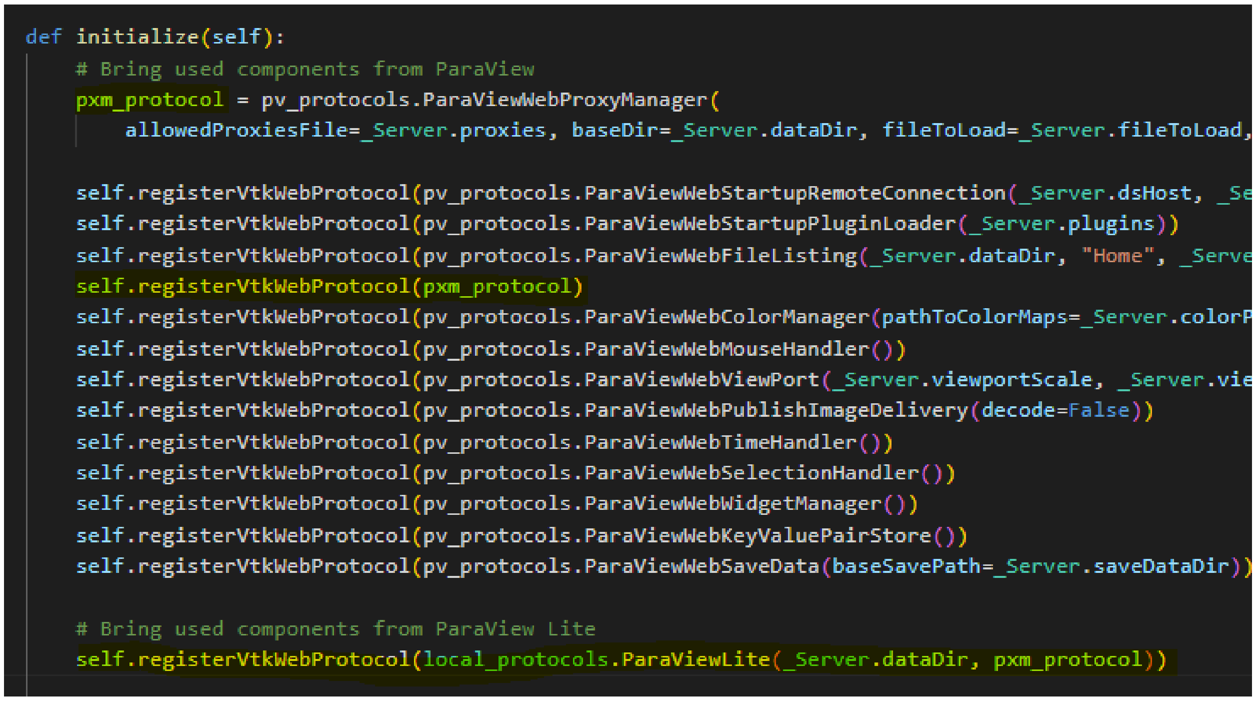
4) Create script for easing the start-up of the system. As the system is hosted in Azure and I have a limited budget each month (150$) I only start the Ubuntu server and the ParaView server under request. So, I create a shell script to avoid typing and errors:
```
a.   $ touch run.sh
b.   $ edit run.sh

$ pvpython --force-offscreen-rendering ./evoker/client/server/pvw-
lite.py --data ~/3d_samples --port 49773

c.   $ chmod +777 ./run.sh
```
5) Run script:
```
a. $ ./run.sh
```

3.4. Changing default representation

The default representation of objects in ParaView is ‘Surface’. The FMP instructor and the author agreed that for meshing workflows it will be more helpful to have ‘Surface with Edges’ instead of ‘Surface’ so the user of Evoker will be able to see the cubes of the mesh. As it is tiresome to change the representation every time you select a new file, it was thought it would be nice to change the default modifying the source code.

The reason to mention this feature in the project memory is to raise awareness of the difficulty of developing and extending the ParaView toolkit. One experienced programmer would think that changing the default representation is an easy task just modifying a line of code of the Vue project. But it turned out to not be so straightforward. The default representation is managed by the server protocol. So, a developer needs to modify the python code that wraps the VTK library. After creating a new topic in the ParaView Discourse web portal, one of the engineers replied that it was needed to change the code in the protocol to override the default behavior when a new representation gets created. Evoker developer needs to do some interactions with this code and request more support in the Discourse portal but in the end, he/she makes it work. It is not easy and even Kitware engineers have to think twice about how to achieve something.

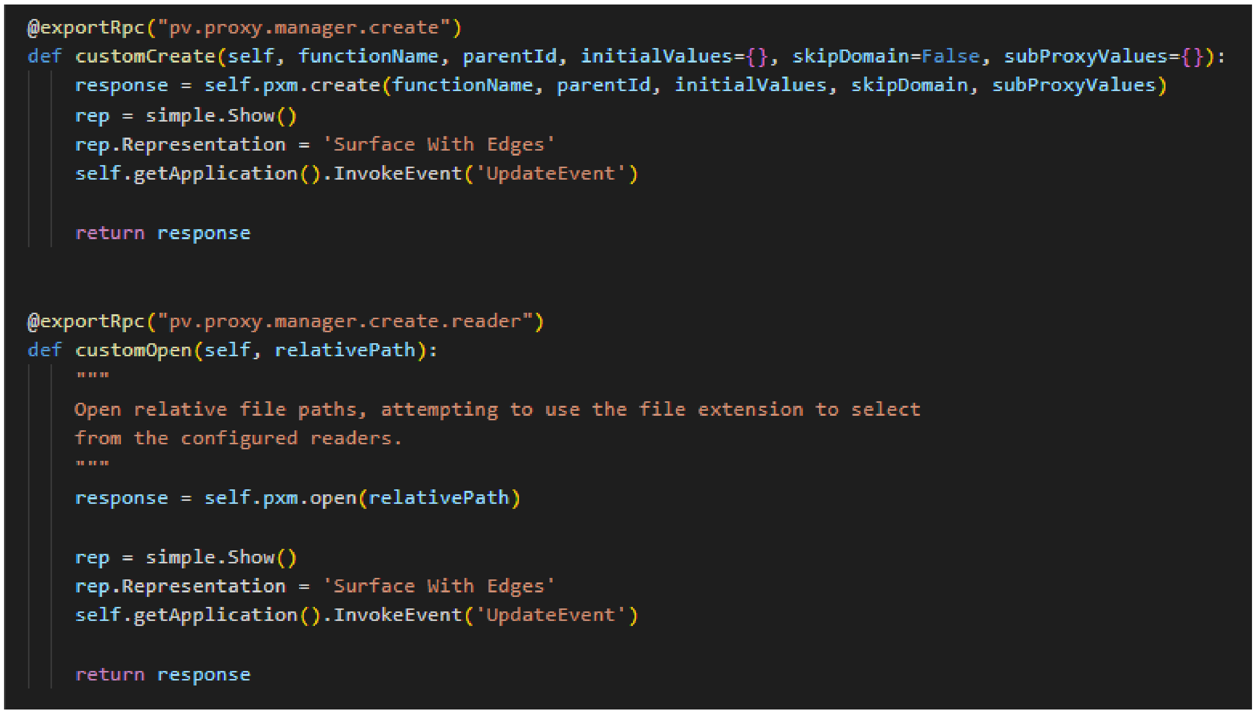
To change the default representation, it was needed to modify two python files that run in the server. In **Figure 35** the reader can see the modifications needed in the **pvw-lite.py** file: it is needed to pass **ParaViewWebProxyManager** object to **ParaviewLite** constructor.



```
def initialize(self):
    # Bring used components from ParaView
    pxm_protocol = pv_protocols.ParaViewWebProxyManager(
        allowedProxiesFile=_Server.proxies, baseDir=_Server.dataDir, fileToLoad=_Server.fileToLoad,

    self.registerVtkWebProtocol(pv_protocols.ParaViewWebStartupRemoteConnection(_Server.dsHost, _Se
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebStartupPluginLoader(_Server.plugins))
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebFileListing(_Server.dataDir, "Home", _Serve
    self.registerVtkWebProtocol(pxm_protocol)
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebColorManager(pathToColorMaps=_Server.colorP
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebMouseHandler())
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebViewPort(_Server.viewportScale, _Server.vie
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebPublishImageDelivery(decode=False))
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebTimeHandler())
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebSelectionHandler())
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebWidgetManager())
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebKeyValuePairStore())
    self.registerVtkWebProtocol(pv_protocols.ParaViewWebSaveData(baseSavePath=_Server.saveDataDir))

    # Bring used components from ParaView Lite
    self.registerVtkWebProtocol(local_protocols.ParaViewLite(_Server.dataDir, pxm_protocol))
```

Figure 35: Modifications in pvw-lite.py for changing default representation.

In **Figure 36** the reader can see the modifications needed in the **lite_protocols.py** file: **customCreate** and **customOpen** methods are overwritten by class inheritance.



```
@exportRpc("pv.proxy.manager.create")
def customCreate(self, functionName, parentId, initialValues={}, skipDomain=False, subProxyValues={}):
    response = self.pxm.create(functionName, parentId, initialValues, skipDomain, subProxyValues)
    rep = simple.Show()
    rep.Representation = 'Surface With Edges'
    self.getApplication().InvokeEvent('UpdateEvent')

    return response


@exportRpc("pv.proxy.manager.create.reader")
def customOpen(self, relativePath):
    """
    Open relative file paths, attempting to use the file extension to select
    from the configured readers.
    """
    response = self.pxm.open(relativePath)

    rep = simple.Show()
    rep.Representation = 'Surface With Edges'
    self.getApplication().InvokeEvent('UpdateEvent')

    return response
```

Figure 36: Modifications in lite_protocols.py for changing default representation

3.5. Usage from tablet and smartphone

Evoker does not need to install anything in the client to run. As proof of it, you can see it running from a smartphone (**Figure 37**) and a tablet (**Figure 38**).

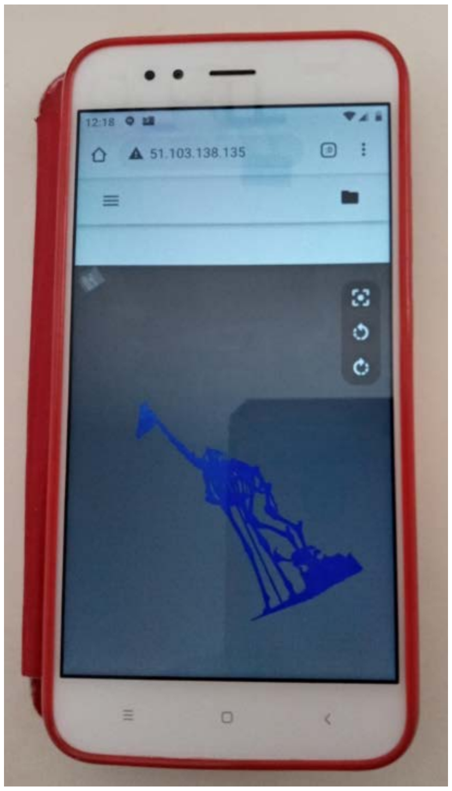


Figure 37: Running Evoker on a smartphone

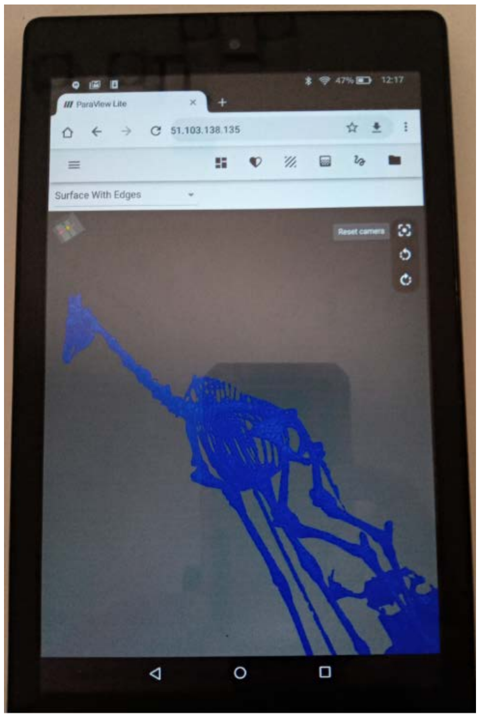


Figure 38: Running Evoker on a tablet

Finally, simultaneous visualization is showed in **Figure 39**.

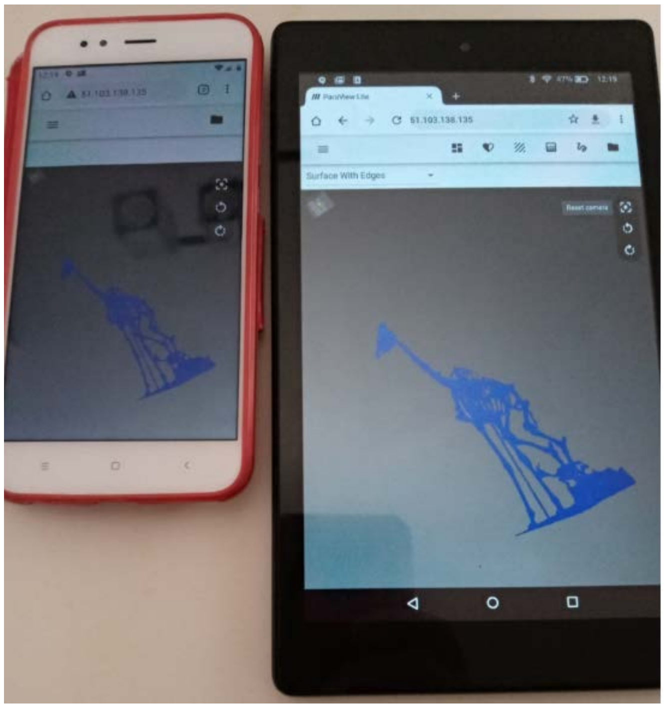


Figure 39: Sharing an Evoker session on a tablet and smartphone

4. Meshing workflows

In this section, the configuration and code development of the meshing tool in Evoker through OpenFOAM is explained.

4.1. OpenFOAM introduction

OpenFOAM is a collection of C++ libraries for solving partial differential equations for continuum mechanics problems, most prominently including CFD. OpenFOAM has been developed for over two decades and has reached a good level of maturity thanks to its large user community. OpenFOAM code is fully open [41].

Covering an OpenFOAM project end to end is a huge engineering process that by no means it is trying to be achieved in Evoker. As an overview of it, find below a list of the main steps of an OpenFOAM project [27]:

1. Create an OpenFOAM case
2. Running applications
3. Mesh generation and conversion
4. Models and physical properties
5. Solving
6. Post-processing

Evoker will help the OpenFOAM user engineer with step #3 (**Mesh generation and conversion**). The most typical use of OpenFOAM is from an interactive shell with the engineer having to edit text files manually. Sometimes these tasks can be cumbersome, and Evoker can be handy.

In this project, one of the goals is to develop a web application that provides a web graphical user interface for mesh settings (cell size, refinement iterations, and processor topology) edition and also execute this job of generating a mesh from a geometry and boundary surfaces in an HPC cloud. The term HPC cloud is being used instead of the HPC cluster because the number of compute nodes is static per execution. Evoker administrator can change the number of compute nodes as a backend administration task.

Evoker has to be configured its settings for a specific field to provide a user-friendly interface. Evoker has a modular design allowing it to customize its settings (cell size, refinements, and processor technology) to other ones modifying a class in the Vue.js front-end, and adding some methods to the python backend.

As a prerequisite, the user must provide the OpenFOAM project (case) and Evoker expects to find some files, folders, and settings in them to execute OpenFOAM meshing commands (i.e. blockMesh, snappyHexMesh).

Find in **Figure 40** depicts a schematic overview of the OpenFOAM process covered in Evoker:

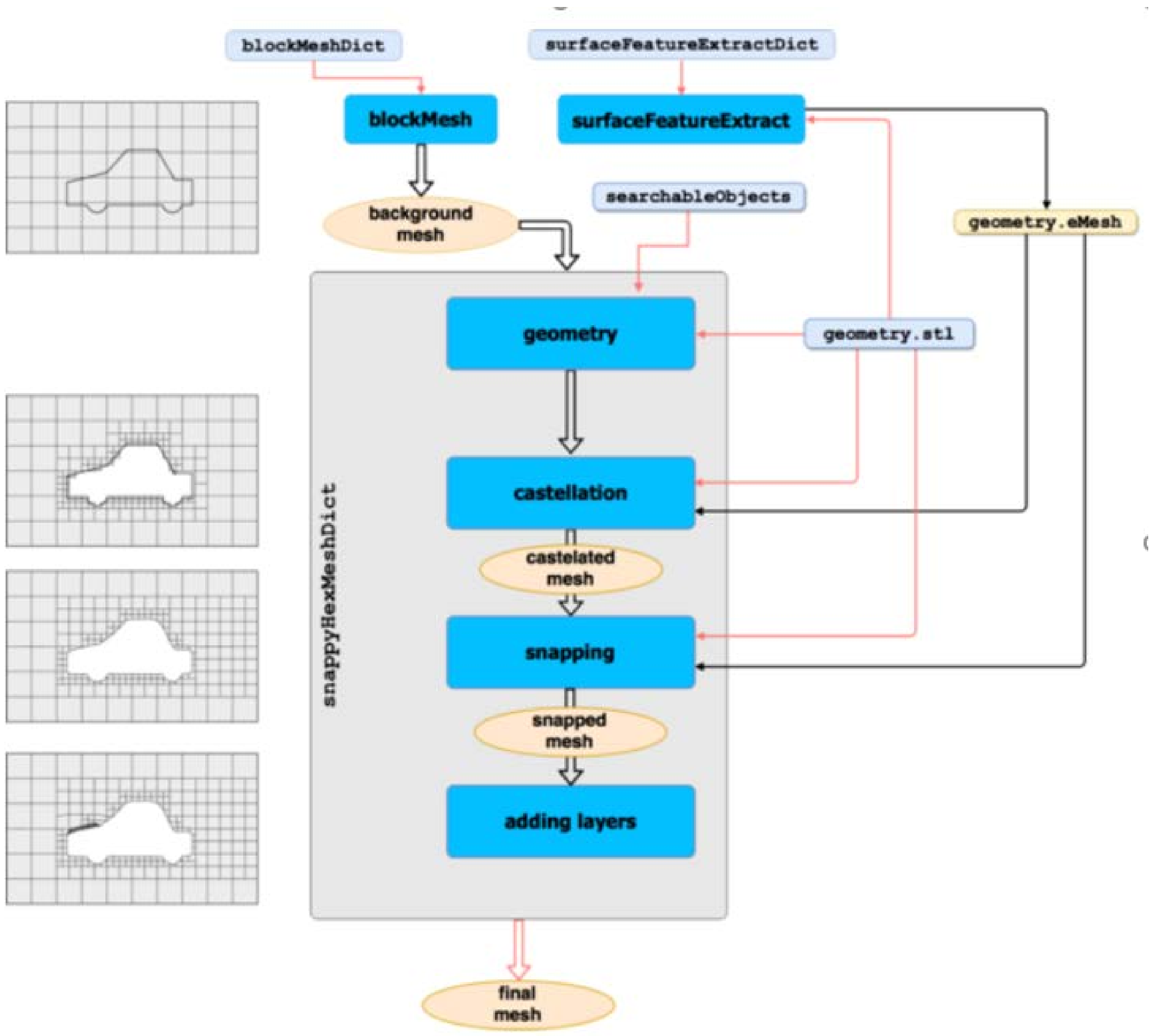


Figure 40: snappyHexMesh overview (original Figure from [55])

So, Evoker has a list of prerequisites (see Section 4.4), offers some settings, and allows to choose the number of processors to parallelize the meshing in an HPC cloud.

4.2. MPI installation

MPI is necessary to parallelize the execution of the meshing in the HPC cloud.

MPI is a standard for message-passing between processes. It is highly used in HPC where the user can run jobs in different compute nodes, and the framework provides the needed communication between these jobs.

In Evoker we are using MPICH implementation [28].

Below are the steps to install it in Ubuntu [29]:

```
$ sudo apt-get update -y

$ sudo apt-get install -y mpich
```

4.3. OpenFOAM installation

Below are the steps to install OpenFOAM in Ubuntu [30]:

```
$ sudo sh -c "wget -O - https://dl.OpenFOAM.org/gpg.key | apt-key add -
"sudo add-apt-repository http://dl.OpenFOAM.org/ubuntu

$ sudo apt-get update

$ sudo apt-get -y install OpenFOAM7
```

4.4. Prerequisites

Evoker has below prerequisites from the OpenFOAM project to work:

1. The OpenFOAM project must be ready to execute **blockMesh** and **snappyHexMesh** commands from an interactive shell in version 7 of OpenFOAM (https://OpenFOAM.org/release/7/).
2. The OpenFOAM project must be uploaded to the server.
3. It has to have a file named **foam.foam** (this file can be empty, and it is just a convention to open it with ParaView)
4. The **/constant/triSurface** directory has the surface definition files that specify your boundary mesh.
5. The OpenFOAM project must have **/constant/polyMesh/blockMeshDict** file and inside this file, the vertices must be defined without being references to other files utilizing the **include** directive.
6. Evoker will extract the name of the refinement surfaces from **/constant/polyMesh/blockMeshDict** file inside the **boundary** section. These surfaces can be created with software such as SolidWorks [65] or HydroSludge [66]. This is a dynamic step Evoker does: the number of refinement surfaces and the names of them are automatically retrieved by the system reading this file.

4.5. Files preparation

Evoker needs to read and write values in the OpenFOAM project folder hosted in the backend to operate. It needs to know the vertices and the surfaces that specify the boundary of the mesh and it needs to write the user settings in some files so that **blockMesh** and **snappyHexMesh** take them into account.

Find below the tasks that Evoker automatically performs related to backend file preparation for meshing:

1. First, it reads the boundary surfaces from **/constant/polyMesh/blockMeshDict**
2. It modifies **/constant/polyMesh/blockMeshDict**:
    a. adding #include "../../UISettings"
    b. Change explicit nodes to the variable $Nodes that will be written in file UISettings
3. It writes (or overwrites if already exists) **/system/decomposeParDict**:

```
FoamFile{
     version     2.0;
     class  dictionary;
     format ascii;
     location    "system";
     object decomposeParDict;}

#include   "../UISettings"

numberOfSubdomains     $numberOfSubdomains;
method     simple;
simpleCoeffs
{
     n     ( $xTopology $yTopology $zTopology );
     delta  0.001;
}
hierarchicalCoeffs
{
     n     ( 1 1 1 );
     delta  0.001;
     order  xyz;
}
metisCoeffs
{
     processorWeights ( 1 1 1 1 );
}
manualCoeffs
{
     dataFile    "";
}
distributed no;
roots ();
```

where we are using variables *$numberOfSubdomains*, *$xTopology*, *$yTopology*, *$zTopology* defined in UISettings file.

4. It writes (or overwrites if already exists) **/system/snappyHexMeshDict**:

```
FoamFile{
    version 2.0;
    class   dictionary;
    format  ascii;
    location     "system";
    object  snappyHexMeshDict;}

#include    "../UISettings"

castellatedMesh  true;
snap             true;
addLayers        false;

geometry {
freeSurface.stl {
    name   freeSurface;
    type   triSurfaceMesh;}
inlet.stl {
    name   inlet;
    type   triSurfaceMesh;}
recirculacion.stl {
    name   recirculacion;
    type   triSurfaceMesh;}
tuboRec.stl {
    name   tuboRec;
    type   triSurfaceMesh;}
pasamuros.stl {
    name   pasamuros;
    type   triSurfaceMesh;}
outlet.stl {
    name   outlet;
    type   triSurfaceMesh;}
tubopared.stl {
    name   tubopared;
    type   triSurfaceMesh;}
Default.stl {
    name   Default;
    type   triSurfaceMesh;}
}
castellatedMeshControls {
    locationInMesh     (2 2 -1);
    refinementSurfaces {
freeSurface{
    level  ($minfreeSurface $maxfreeSurface); }
inlet{
    level  ($mininlet $maxinlet); }
recirculacion{
    level  ($minrecirculacion $maxrecirculacion); }
tuboRec{
    level  ($mintuboRec $maxtuboRec); }
pasamuros{
    level  ($minpasamuros $maxpasamuros); }
outlet{
    level  ($minoutlet $maxoutlet); }
tubopared{
    level  ($mintubopared $maxtubopared); }
Default{
    level  ($minDefault $maxDefault); }
}
    refinementRegions { }
    features ();
    minRefinementCells 0;
    maxGlobalCells     10000000;
    resolveFeatureAngle      5;
    nCellsBetweenLevels      1;
    maxLocalCells500000;
    allowFreeStandingZoneFaces    true;
}
snapControls {
    nSmoothPatch 3;
    tolerance    1;
    nSolveIter   300;
    nRelaxIter   5;
    nFeatureSnapIter   10;
    implicitFeatureSnap true;
    explicitFeatureSnap false;
}
addLayersControls {
    layers {
        freeSurface{
            nSurfaceLayers 0; }
        inlet{
            nSurfaceLayers 0; }
        recirculacion{
            nSurfaceLayers 0; }
        tuboRec{
            nSurfaceLayers 0; }
        pasamuros{
            nSurfaceLayers 0; }
        outlet{
            nSurfaceLayers 0; }
        tubopared{
            nSurfaceLayers 0; }
        Default{
            nSurfaceLayers 0; }
    }
    nSmoothSurfaceNormals    5;
    slipFeatureAngle   30.0;
    nBufferCellsNoExtrude    0;
    nRelaxIter   5;
    relativeSizesfalse;
    minMedianAxisAngle 90.0;
    maxFaceThicknessRatio    0.5;
    nSmoothNormals     3;
    maxThicknessToMedialRatio      0.3;
    nLayerIter   50;
    minThickness 0.05;
    nSmoothThickness   10;
    nGrow  10;
    nRelaxedIter 20;
    concaveAngle 90.0;
```

```
        featureAngle 60;
        firstLayerThickness     0.05;
        expansionRatio     1;
    }
    meshQualityControls {
        minTetQuality1.0E-20;
        minVol  1.0E-14;
        maxInternalSkewness     4.0;
        maxBoundarySkewness     20.0;
        maxConcave   80.0;
        minFaceWeight0.05;
        minVolRatio  0.01;
        minTwist     0.05;
        minArea -1.0;
        maxNonOrtho  65.0;
        minTriangleTwist   -1.0;
        minDeterminant     0.01;
        errorReduction     0.75;
        nSmoothScale 4;
        relaxed {
            maxNonOrtho    75.0; }
    }
    mergeTolerance   1e-03;
    debug 0;
```

where we are setting minimum and maximum refinement surfaces values defined in UISettings file and we are settings castellatedMesh and snap to true, and addLayers to false.

5. Evoker will write user settings to be applied in the mesh job to UISettings file:

```
resolution 0.5;
NODES (54 16 54);
xTopology 2;
yTopology 2;
zTopology 1;
numberOfSubdomains 4;
min$sup[i] 0;
max$sup[i] 0;
```

and considering the cell size entered in Evoker front-end settings.

4.6. Workflow

A web-based client-server meshing generator is an innovative functionality that is not common in the industry.

The workflow design will be as follows: users will be able to select OpenFOAM projects folders through the current Files button and they need to select a file named **foam.foam**. This file is an empty file and it is a standard way of starting up an OpenFOAM project for visualization in the ParaView realm. After the user has selected the **foam.foam** the application will render current OpenFOAM with VTK backend and add it to the DataSet of the application like any other ordinary file (obj, stl, etc.). At this point, the application will show a Mesh button (**Figure 41**) similar to ParaView Lite's other commands like Clip, Cut, …

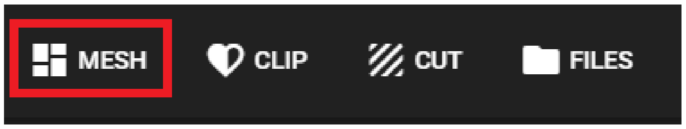
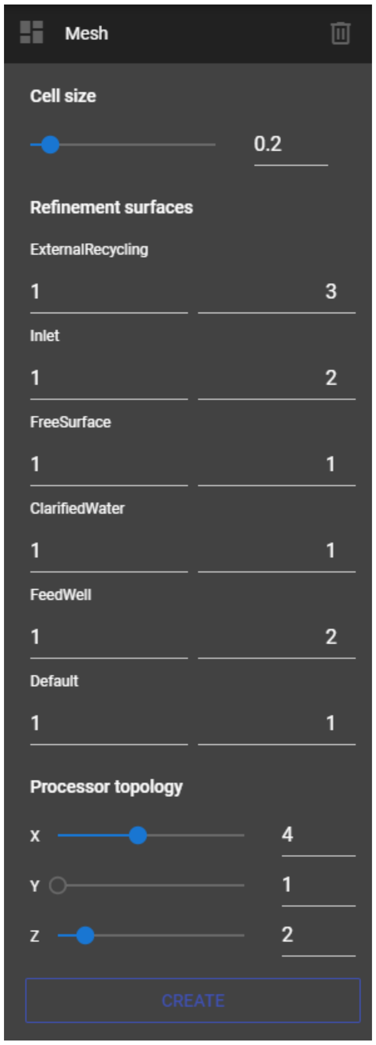


Figure 41: Evoker GUI commands

When the user clicks Mesh a meshing options menu will appear at the left of the rendering area. These options will be the **Cell size,** the **Refinement surfaces,** and the **Processor topology** as you can see in **Figure 42**:



Mesh

Cell size

0.2

Refinement surfaces

ExternalRecycling	1	3
Inlet	1	2
FreeSurface	1	1
ClarifiedWater	1	1
FeedWell	1	2
Default	1	1

Processor topology

X	4
Y	1
Z	2

CREATE

Figure 42: Evoker meshing settings

From the cell size, Evoker calculates the number of cells that the OpenFOAM blockMesh operation will have. The backend running at the server will read blockMeshDict file to get the vertices. From the vertices, it will obtain the min and max values for the X, Y, Z axes. And with below formula it will calculate the number of nodes:

$$Nodes\_t = (max\_t - min\_t) / cell\_size$$

where t can be X, Y, and Z

For the refinements, when the user clicks the Mesh button the number and the names of the surfaces are read dynamically for each OpenFOAM project by the backend reading the blockMeshDict file. After the cell size, the minimum, and the maximum refinement values are set, the user can click the Create button. All these settings will be passed to the backend running on the server. The backend will calculate the number of nodes and will write them in the UISettings file. Find below a sample of the UISettings file:

```
resolution 340;
NODES (54 16 54);
xTopology 2;
yTopology 2;
zTopology 1;
minwalls 0;
maxwalls 0;
minSFango 1;
maxSFango 3;
minEntrada 0;
maxEntrada 1;
minSupLibre 0;
maxSupLibre 1;
minSAguaClar 1;
maxSAguaClar 3;
mincampana 2;
maxcampana 3;
minDefault 0;
maxDefault 0;
numberOfSubdomains 4;
```

The backend after saving the settings in the UISettings file will call **blockMesh** and **snappyHexMesh** from OpenFOAM with the folder path passed from the web client (it is the folder of the **foam.foam** file). During these executions, the UI will be waiting for completion. Once completed, the UI will open **foam.foam** file again and the VTK will generate a DataSet to get rendered as we did before meshing it. But this time VTK will use the new time step folders (0.1, 0.2, …) to render the OpenFOAM project that will contain the results of our calls to **blockMesh** and **snappyHexMesh**. This is a nice extension and integration of the Kitware sample app ParaView Lite with a mesh generator that executes transparently and smoothly for the user.

Depending on the value of

$$numberOfSubdomains = xTopology * yTopology * zTopology$$

one of the below OpenFOAM api calls can be used:

1) numberOfSubdomains = 1
    a. blockMesh -case <fullPath>
    b. snappyHexMesh -case <fullPath>
2) numberOfSubdomains > 1
    a. blockMesh -case <fullPath>
    b. decomposePar -case <fullPath>
    c. mpirun -np <numberOfSubdomains> snappyHexMesh -parallel -case <fullPath>
    d. reconstructParMesh -case <fullPath>

Every time we execute a mesh operation, we clean up previous mesh outputs from the last execution in the server by removing timestamps and processor folders:

```
$ rm -rf /processor*

$ rm -rf /0.*
```

4.7. The client-server communication

For adding a new meshing workflow to the fork of ParaView Lite we need to provide a few new remote procedure calls between client and server.

For this purpose, **wslink** library [31] is being used.

Wslink library was already being used by ParaView Lite. Evoker is extending the Lite protocol with three new operations:

1) **paraview.lite.mesh.surfaces**: it returns the refinement surfaces as an array from the folder path passed.
2) **paraview.lite.mesh.run**: it receives folder path, cell size, refinement surfaces, and processor topology as parameters. It calculates the nodes and saves them together with refinement surfaces maximum and minimum values to the UISettings file. It also saves in the UISettings file the x, y, and z processor topology values and the number of processor subdomains that is the multiplication of these 3 topology values. It calls blockMesh and snappyHexMesh from the OpenFOAM library.
3) **paraview.lite.mesh.persistence**: it reads all meshing values (cell size, refinement surfaces min and max values, x, y, and z processor topology) from the last meshing execution in the folder path passed. This API improves the user experience of the app and also allows iterating the meshing process with the helper of remembering the last values.

4.8. OpenFOAM in the HPC cloud with MPI

So far, Evoker has an Ubuntu server that runs the webserver, the ParaView backend, and the OpenFOAM operations. Now it should make the system scalable and have an Ubuntu server for the webserver and the ParaView but having a cloud of compute nodes to run OpenFOAM.

4.8.1. Clone existing VM for OpenFOAM nodes

As Evoker has already set up all needed applications and libraries for the first Ubuntu server in Azure, it is desirable to clone that VM to mount the other compute nodes without needed to re-install them again. The goal is to have four Ubuntu servers for our HPC cloud.

To clone the VM it is needed to create a snapshot to the attached disk. Then it is needed to create a Managed Disk from the snapshot. Finally, a VM can be created from the Managed Disk.

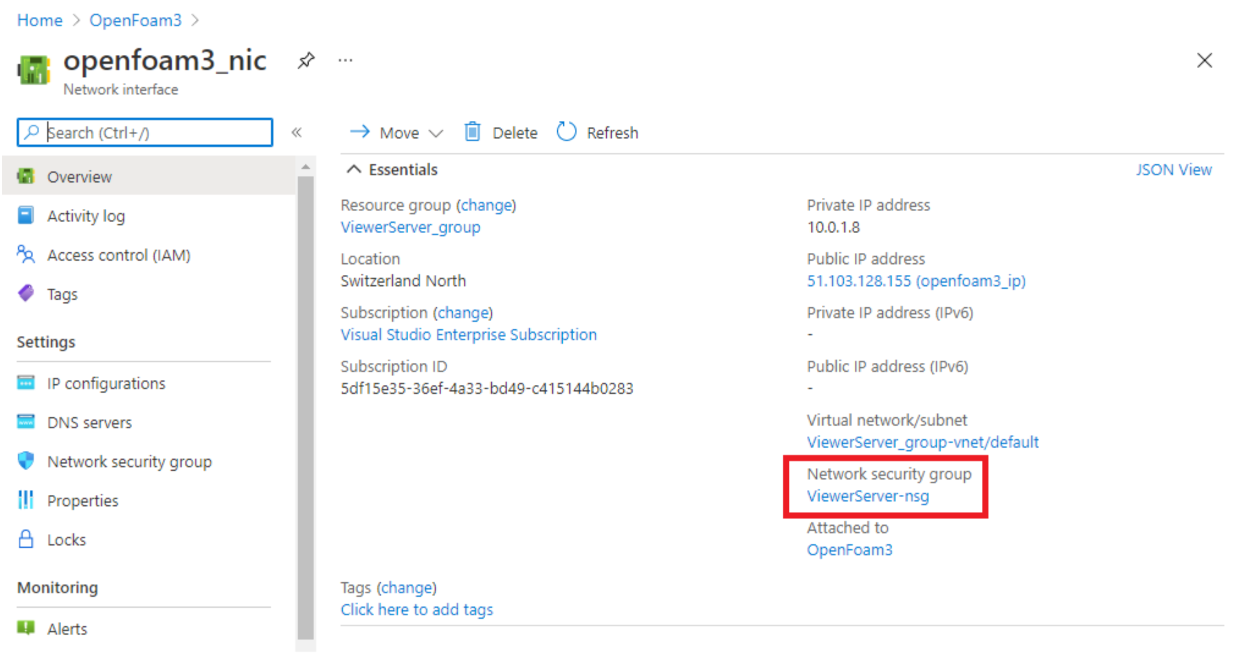
This is the Powershell script to create the Managed Disk [32]:

```
$resourceGroupName ='ViewerServer_group'
$snapshotName = 'Disk_evoker_vm_february_10'
$diskName = 'Disk_evoker_vm_february_10_managed_disk'
$diskSize = '30'
$storageType = 'Premium_LRS'
$location = 'Switzerland North'
Select-AzureRmSubscription -SubscriptionId '5df15e35-36ef-4a33-bd49-
c415144b0283'
$snapshot = Get-AzureRmSnapshot -ResourceGroupName
$resourceGroupName -SnapshotName $snapshotName
$disk = New-AzureRmDiskConfig -AccountType $storageType -Location
$location -CreateOption Copy -SourceResourceId $snapshot.Id
New-AzureRmDisk -Disk $disk -ResourceGroupName $resourceGroupName -
DiskName $diskName
```

And this is the script to create the VM from the Managed Disk [32]:

```
$virtualNetworkName = 'ViewerServer_group-vnet'
$virtualMachineName = 'OpenFOAM1'
$virtualMachineSize = 'Standard_E4ds_v4'
$VirtualMachine = New-AzureRmVMConfig -VMName $virtualMachineName -
VMSize $virtualMachineSize
$VirtualMachine = Set-AzureRmVMOSDisk -VM $VirtualMachine -
ManagedDiskId /subscriptions/5df15e35-36ef-4a33-bd49-
c415144b0283/resourceGroups/ViewerServer_group/providers/Microsoft.C
ompute/disks/Disk_evoker_vm_february_10_managed_disk -CreateOption
Attach -Linux
$publicIp = New-AzureRmPublicIpAddress -Name
($VirtualMachineName.ToLower()+'_ip') -ResourceGroupName
$resourceGroupName -Location $snapshot.Location -AllocationMethod
Dynamic
$vnet = Get-AzureRmVirtualNetwork -Name $virtualNetworkName -
ResourceGroupName $resourceGroupName
$nic = New-AzureRmNetworkInterface -Name
($VirtualMachineName.ToLower()+'_nic') -ResourceGroupName
$resourceGroupName -Location $snapshot.Location -SubnetId
$vnet.Subnets[0].Id -PublicIpAddressId $publicIp.Id
$VirtualMachine = Add-AzureRmVMNetworkInterface -VM $VirtualMachine
-Id $nic.Id
New-AzureRmVM -VM $VirtualMachine -ResourceGroupName
$resourceGroupName -Location $snapshot.Location
```

It is found out that the newly created VM may not have the Network security group of the NIC well configured. It is needed to make sure to set it to the same group as your main Web Server: ViewerServer-nsg (**Figure 43**).



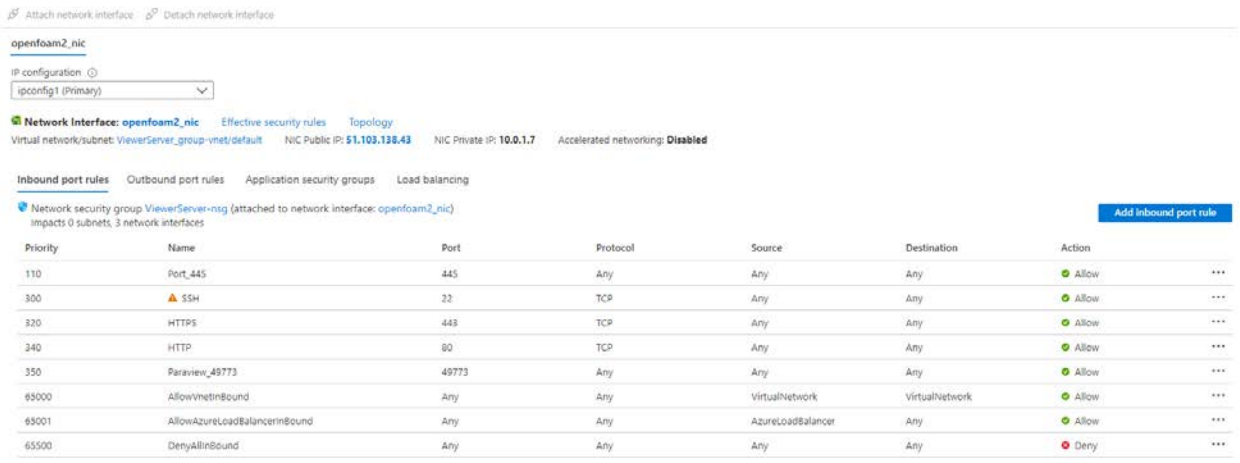
openfoam3_nic
Network interface

Resource group (change)	Private IP address
ViewerServer_group	10.0.1.8
Location	Public IP address
Switzerland North	51.103.128.155 (openfoam3_ip)
Subscription (change)	Private IP address (IPv6)
Visual Studio Enterprise Subscription	-
Subscription ID	Public IP address (IPv6)
5df15e35-36ef-4a33-bd49-c415144b0283	-
	Virtual network/subnet
	ViewerServer_group-vnet/default
	Network security group
	ViewerServer-nsg
	Attached to
	OpenFoam3

Tags (change)
Click here to add tags

Figure 43: Network security group of the new VM

After configuring the network security group for ViewerServer-nsg (or the name given to the main VM), Azure user can check you have the needed ports open as can be seen in **Figure 44**:



Network Interface: openfoam2_nic
Virtual network/subnet: ViewerServer_group-vnet/default NIC Public IP: **51.103.138.43** NIC Private IP: **10.0.1.7** Accelerated networking: **Disabled**

Network security group ViewerServer-nsg (attached to network interface: openfoam2_nic)
Impacts 0 subnets, 3 network interfaces

Priority	Name	Port	Protocol	Source	Destination	Action
110	Port_445	445	Any	Any	Any	Allow
300	SSH	22	TCP	Any	Any	Allow
320	HTTPS	443	TCP	Any	Any	Allow
340	HTTP	80	TCP	Any	Any	Allow
350	Paraview_49773	49773	Any	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Figure 44: Ports properly configured for Evoker

After these two steps for each desired new VM, Evoker admin can go ahead and start the new VM from Azure Portal. This process of cloning the VM worked smoothly and the VM's have all libraries installed and ready to be executed.

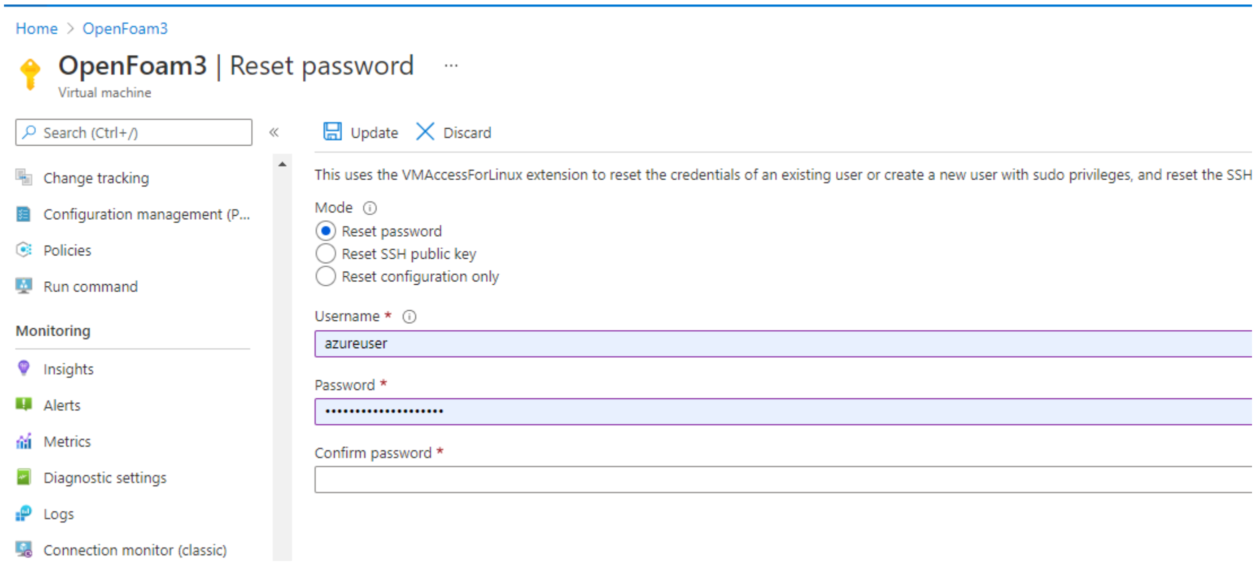
4.8.2. SSH connection between web server VM and OpenFOAM VM's

Before the end-to-end (client web app connecting to the webserver) workflow is changed to use HPC cloud for meshing the admin should run OpenFOAM commands from an interactive shell from the Web Server and make them work on the cloud compute nodes.

In the VM where you the connection destination (OpenFOAM3 hostname):

```
Log in as root
Edit ssh config:
    sudo vi /etc/ssh/sshd_config
Change this line:
    PasswordAuthentication no
to
    PasswordAuthentication yes
Restart daemon:
    sudo systemctl restart sshd
```

It is also found out that cloned VM's also need to restart their password in the Azure portal (this seems an issue of Azure). This can be done as seen in **Figure 45**:



This uses the VMAccessForLinux extension to reset the credentials of an existing user or create a new user with sudo privileges, and reset the SSH

Mode
[✓ Reset password]
Reset SSH public key
Reset configuration only

Username * azureuser

Password * ••••••••••••••••••••

Confirm password *

Figure 45: Resetting password of cloned VM

OpenFOAM cloud execution is based on SSH. So, the first task would be to enable SSH connection from the Web Server machine to OpenFOAM nodes. This is a way to achieve that [33]:

1) Create RSA keys in the Web Server computer(the computer from where the user wants to connect):

```
$ ssh-keygen
```

2) Copy the public key from previous to the OpenFOAM nodes(the computer where the user wants to connect):
    ```
    $ ssh-copy-id username@remote_host
    ```

The reader can see above steps work, execute a bash command (i.e. lscpu) remotely with mpirun:

```
$ mpirun -n 2 --host 51.103.166.0,51.103.138.43 lscpu
```

You can see below the return from last command:

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
Stepping:            7
CPU MHz:             2593.905
BogoMIPS:            5187.81
Virtualization:      VT-x
Hypervisor vendor:   Microsoft
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            1024K
L3 cache:            36608K
NUMA node0 CPU(s):   0-3
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall
nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology cpuid pni
pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes
xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
invpcid_single tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1 hle
avx2 smep bmi2 erms invpcid rtm mpx avx512f avx512dq rdseed adx smap
clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1
xsaves avx512_vnni md_clear arch_capabilities
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  2
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
```

```
Model:                 85
Model name:            Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
Stepping:              7
CPU MHz:               2593.907
BogoMIPS:              5187.81
Virtualization:        VT-x
Hypervisor vendor:     Microsoft
Virtualization type:  full
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              36608K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall
nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology cpuid pni
pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes
xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
invpcid_single tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1 hle
avx2 smep bmi2 erms invpcid rtm mpx avx512f avx512dq rdseed adx smap
clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1
xsaves avx512_vnni md_clear arch_capabilities
```

4.8.3. Sharing data with Azure Files

Figure 46 is an overview of the shared storage scheme; you can see that the front-end and the compute nodes have access to the same storage. This is fundamental to run mpi.

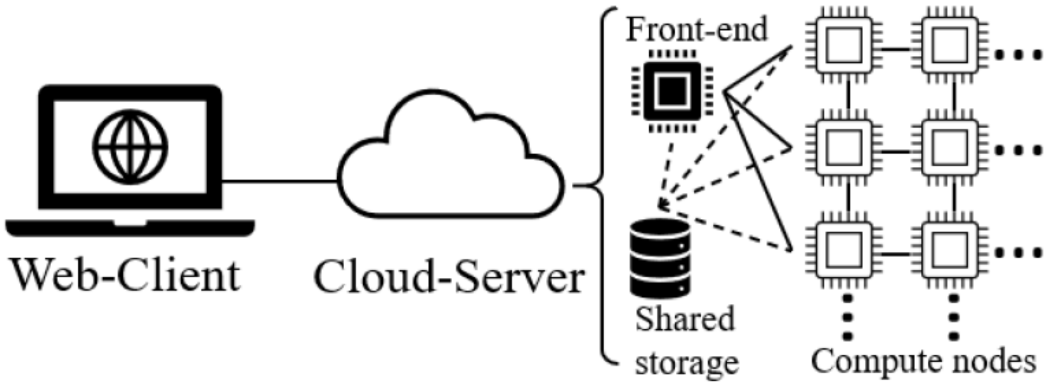


Figure 46: Shared storage scheme

To be able to mesh a project using different nodes we should have access to shared files. Azure offers a feature called Azure Files that offers fully managed file shares in the cloud that are accessible via the industry standard Server Message Block (SMB) protocol or Network File System (NFS) protocol [34].

The first thing the administrator needs to do is to create an Azure file Share [35] as shown in **Figure 47**.

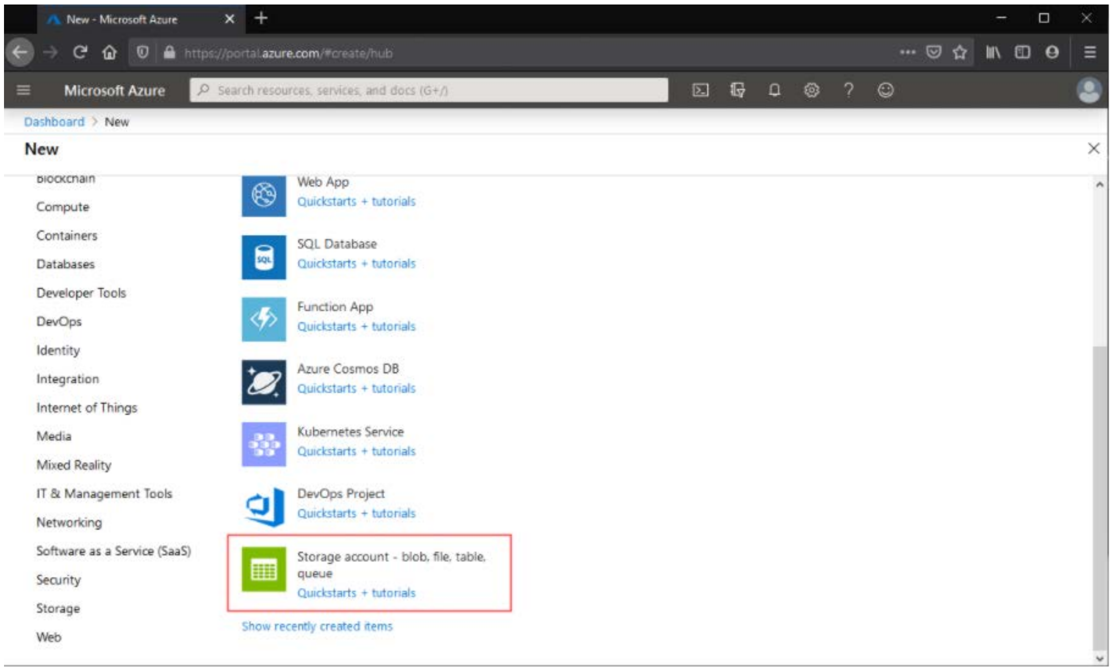


Figure 47: Create an Azure file share

Then it is needed to mount the Azure file share in Ubuntu. The administrator can do it on-demand with mount or permanently with /etc/fstab [36]. In Evoker this has been done permanently.

4.8.4. Executing OpenFOAM in the cloud from the web app

Find below the code snippet of the backend that call OpenFOAM for 2 nodes:

```
  if numberOfSubdomains == 1:
    subprocess.run(["blockMesh", "-case", fullPath])
    subprocess.run(["snappyHexMesh", "-case", fullPath])
  else:
    subprocess.run(["blockMesh", "-case", fullPath])
    subprocess.run(["decomposePar", "-case", fullPath])
    command = "mpirun -n %d --host 51.103.138.43,51.103.166.0 --oversubscribe
bash /home/azureuser/remote.sh %s" % (numberOfSubdomains, fullPath)
    subprocess.run(command.split())
    subprocess.run(["reconstructParMesh", "-latestTime", "-case", fullPath])
```

If the number of processors is one, there is no need to use the HPC cloud and the mesh is done in the main web server. In this case, there is no need to call decomposePar and reconstructParMesh either. When there is more than 1 processor, it needs to call mpirun with ranks and hosts.

When using MPI it has been found that calling snappyHexMesh directly from the backend code did not work because it was not finding libraries in the remote compute node path. It was possible to solve this by calling **remote.sh** stored in the remote compute node. Find below the contents of **remote.sh**:

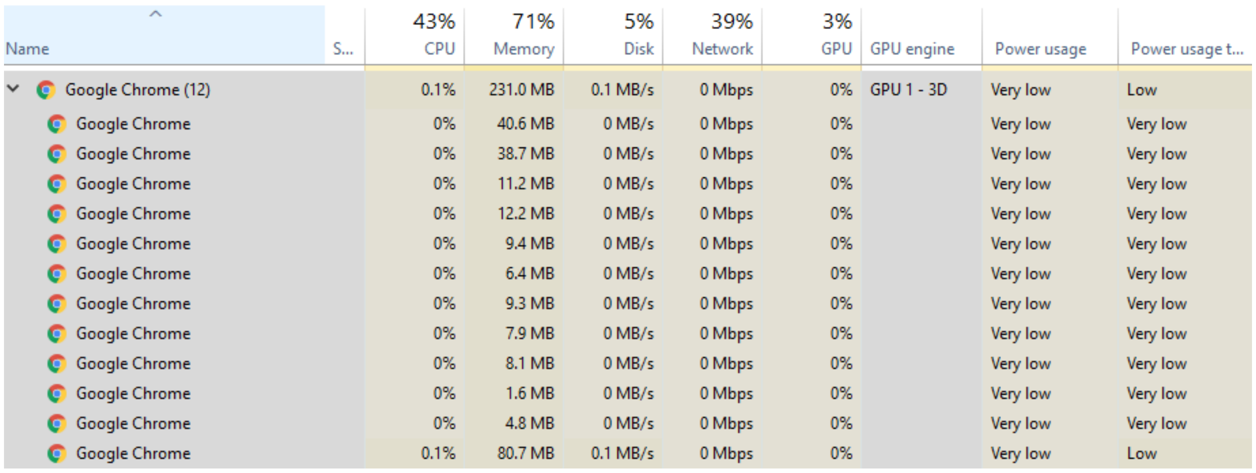
```
. /usr/lib/OpenFOAM/OpenFOAM2012/etc/bashrc
snappyHexMesh -parallel -case $1
```

5. Testing and computational results

In this section, Evoker has been tested against state-of-the-art tests, and also performance test is being done for the meshing functionality in the HPC cloud with different process configurations.

5.1. Three-dimensional visualization results

Adding the same STL of 142MB mentioned in the state-of-the-art section of this thesis shows that the evoker is using the GPU to accelerate the rendering, is not consuming a lot of CPU, and only 231 MB of memory in the browser (**Figure 48**).



Name	S...	43% CPU	71% Memory	5% Disk	39% Network	3% GPU	GPU engine	Power usage	Power usage t...
Google Chrome (12)		0.1%	231.0 MB	0.1 MB/s	0 Mbps	0%	GPU 1 - 3D	Very low	Low
Google Chrome		0%	40.6 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	38.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	11.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	12.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	9.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	6.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	9.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	7.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	8.1 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	1.6 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	4.8 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0.1%	80.7 MB	0.1 MB/s	0 Mbps	0%		Very low	Low

Figure 48: Evoker 3D visualization performance test 1

The reader can see Serpentine01A.stl rendered in **Figure 49**.

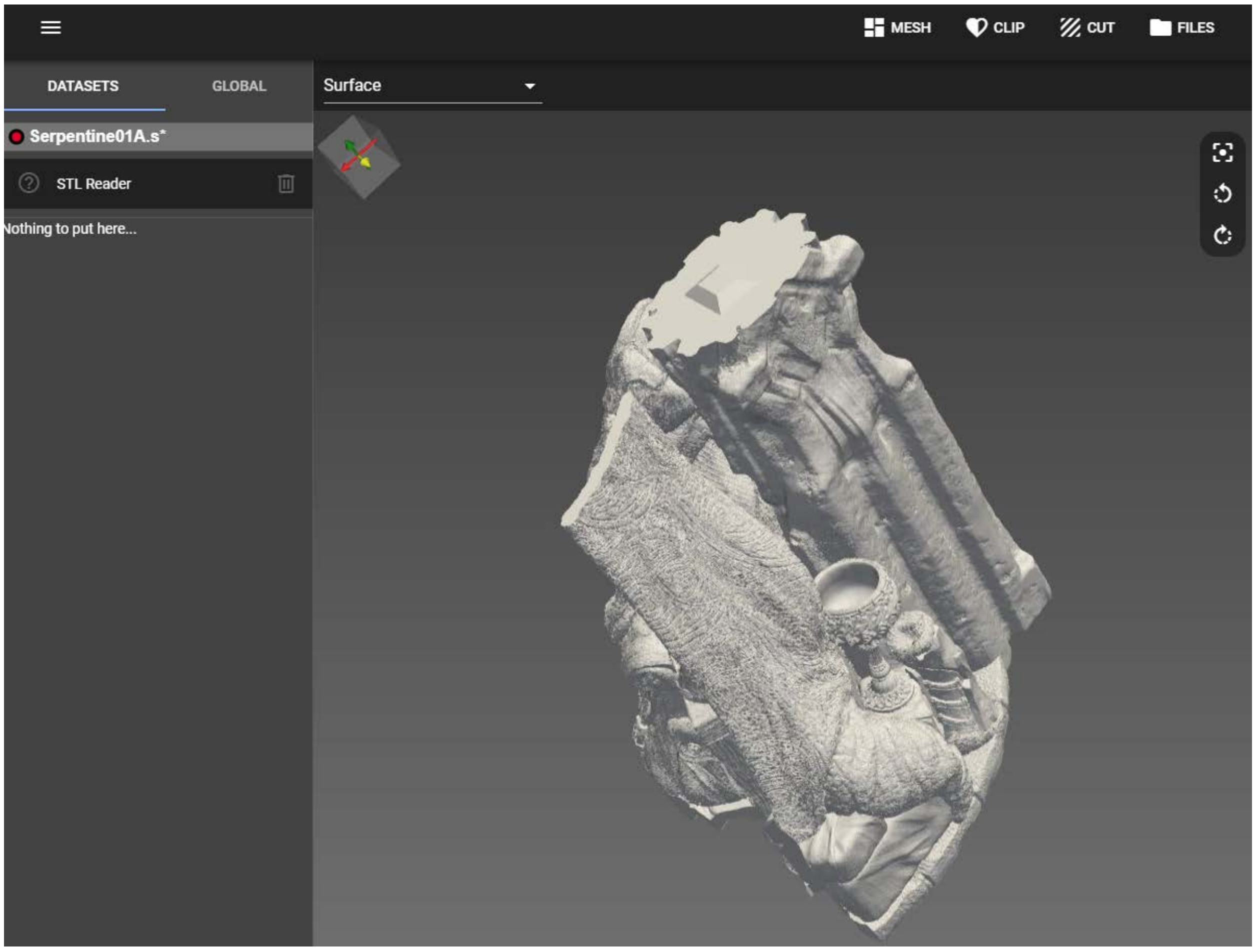
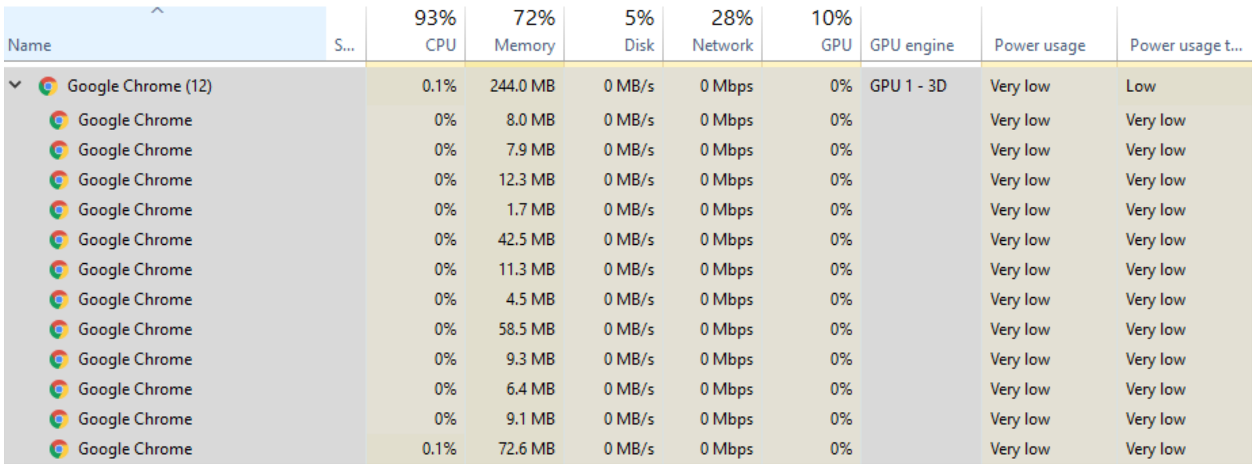


Figure 49: Serpentine01A.stl rendered in Evoker

Adding the second test file (SerpentineMerchant_SPLIT_05.stl in **Figure 51**) and so having 250MB loaded in total, only increases the memory in the browser to 244 MB. This low consumption in the client is due to the fact of doing the rendering in the server (**Figure 50**).



Name	S...	93% CPU	72% Memory	5% Disk	28% Network	10% GPU	GPU engine	Power usage	Power usage t...
Google Chrome (12)		0.1%	244.0 MB	0 MB/s	0 Mbps	0%	GPU 1 - 3D	Very low	Low
Google Chrome		0%	8.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	7.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	12.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	1.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	42.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	11.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	4.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	58.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	9.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	6.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0%	9.1 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome		0.1%	72.6 MB	0 MB/s	0 Mbps	0%		Very low	Very low

Figure 50: Evoker 3D visualization performance test 2

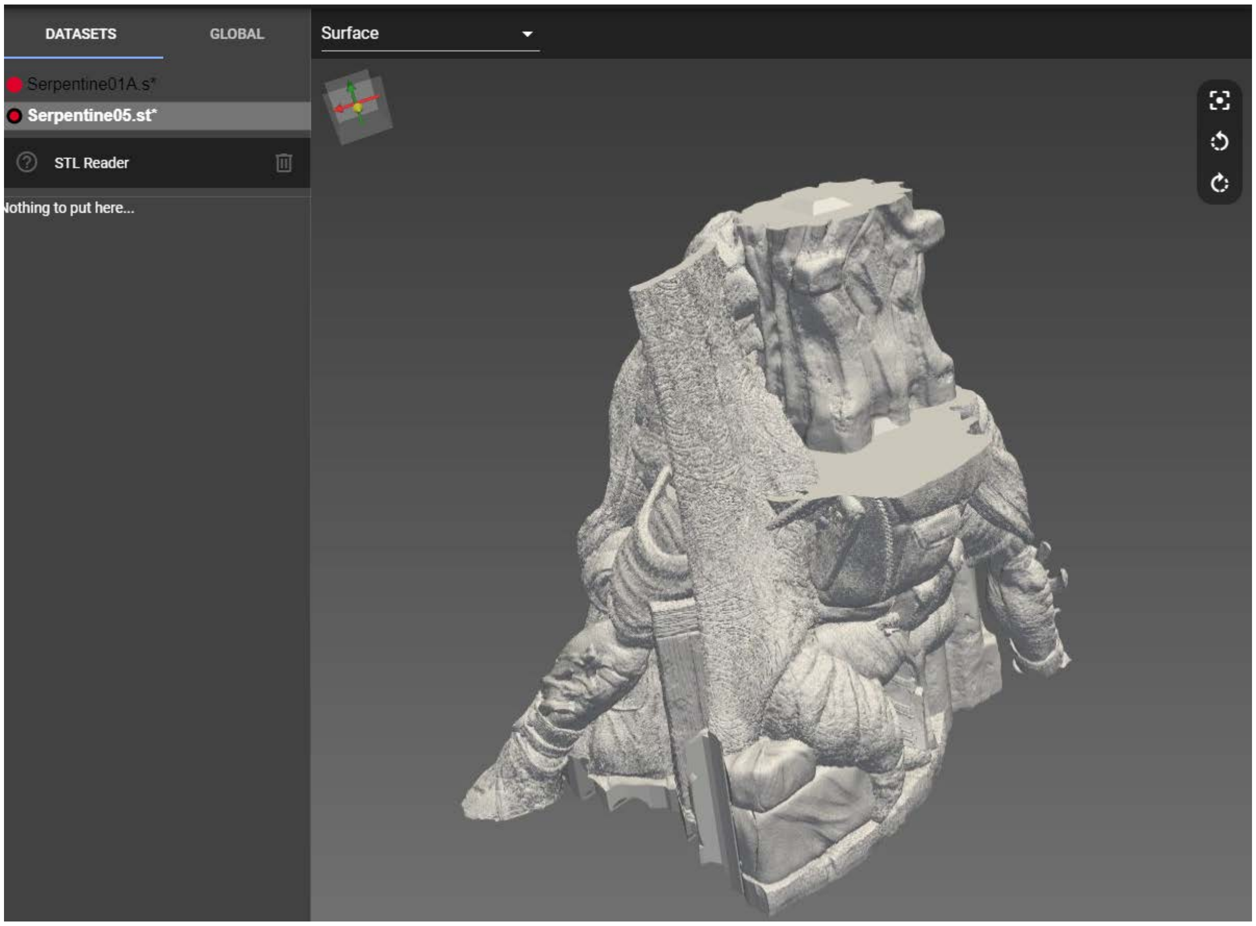
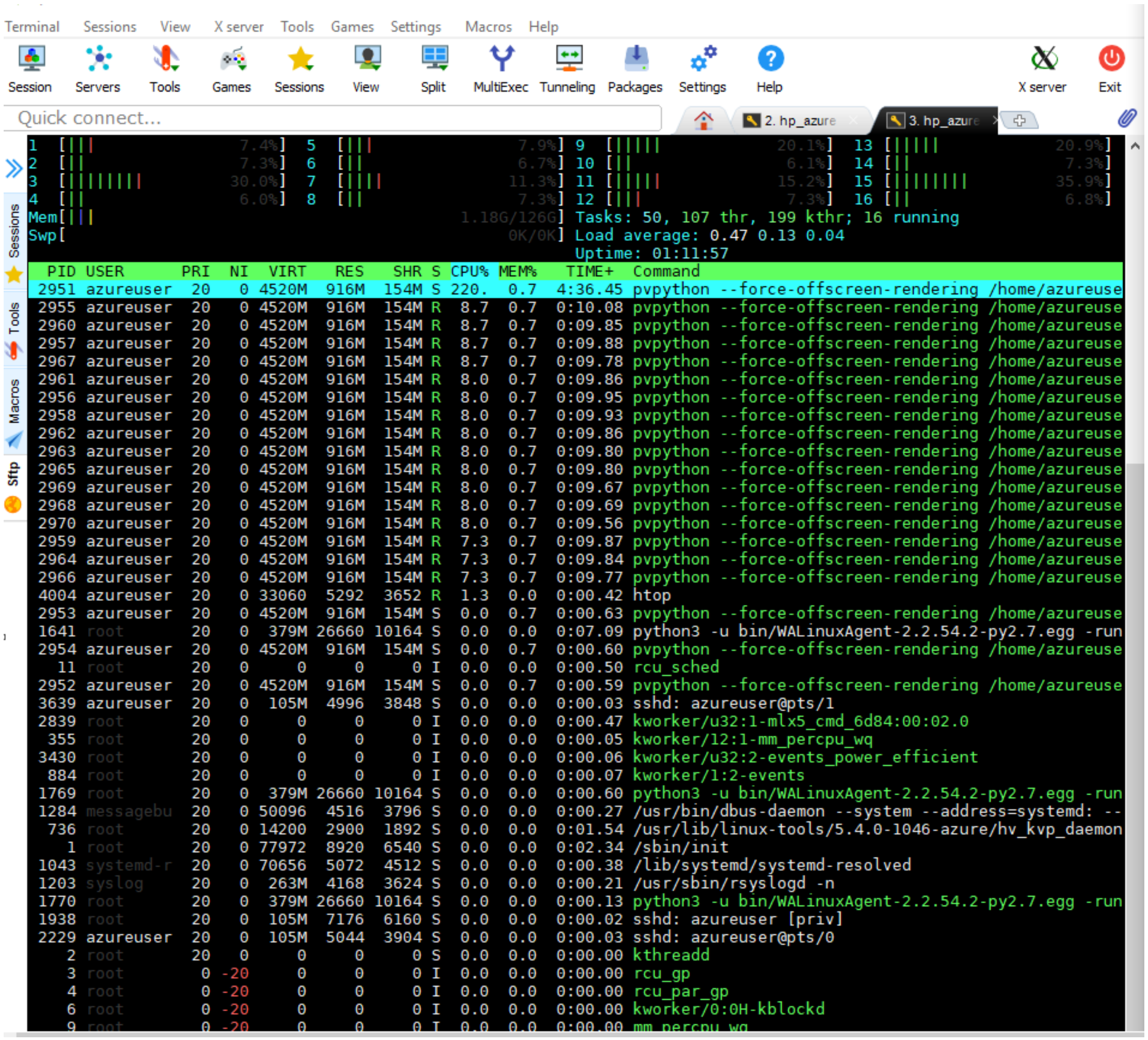


Figure 51: Serpentine05.stl rendered in Evoker

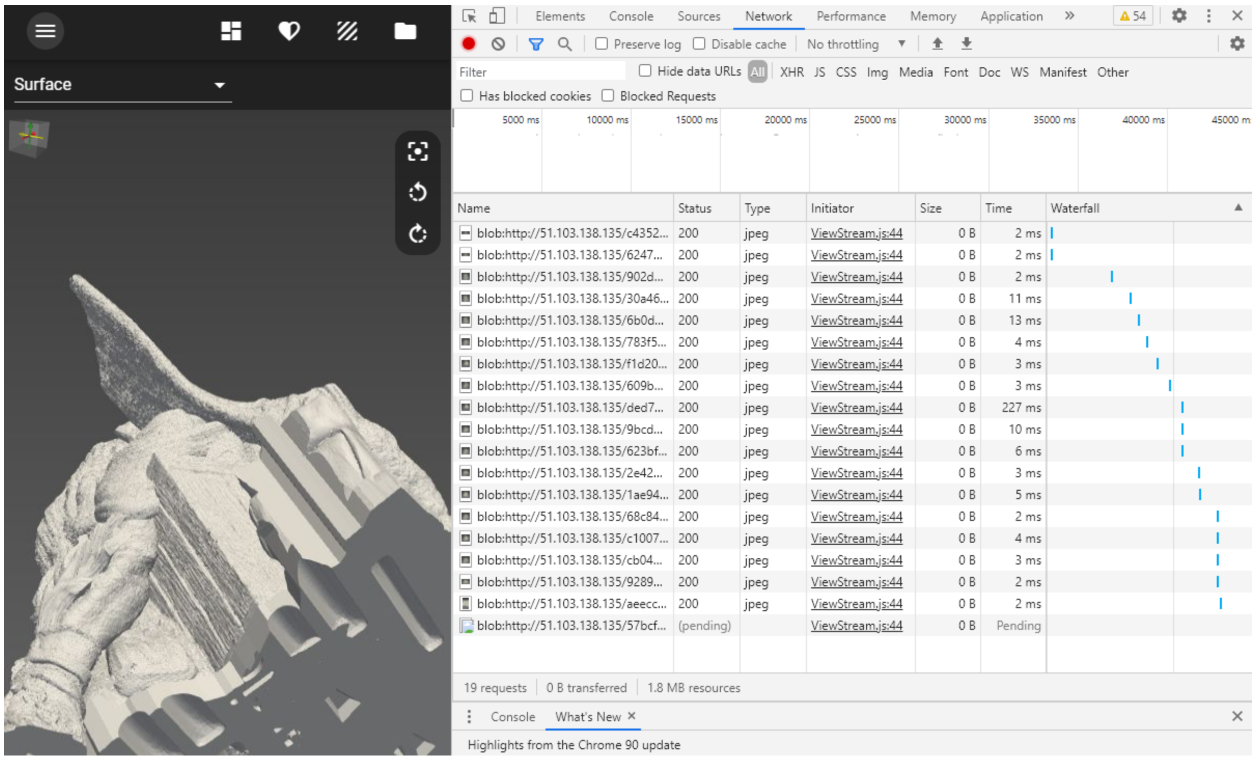
In **Figure 52** the reader can see the percentage of CPU used in the server while rotating both datasets (serpentine01 and serpentine05).



```
1  [|||                7.4%]  5  [|||               7.9%]  9  [||||||            20.1%]  13 [||||||           20.9%]
2  [||                 7.3%]  6  [||                6.7%]  10 [||                 6.1%]  14 [||                7.3%]
3  [||||||||||        30.0%]  7  [||||             11.3%]  11 [||||||            15.2%]  15 [||||||||||       35.9%]
4  [||                 6.0%]  8  [||                7.3%]  12 [|||                7.3%]  16 [||                6.8%]
Mem[|||                                      1.18G/126G]  Tasks: 50, 107 thr, 199 kthr; 16 running
Swp[                                              0K/0K]  Load average: 0.47 0.13 0.04
                                                          Uptime: 01:11:57
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2951 azureuser  20   0 4520M  916M  154M S 220.  0.7  4:36.45 pvpython --force-offscreen-rendering /home/azureuse
 2955 azureuser  20   0 4520M  916M  154M R  8.7  0.7  0:10.08 pvpython --force-offscreen-rendering /home/azureuse
 2960 azureuser  20   0 4520M  916M  154M R  8.7  0.7  0:09.85 pvpython --force-offscreen-rendering /home/azureuse
 2957 azureuser  20   0 4520M  916M  154M R  8.7  0.7  0:09.88 pvpython --force-offscreen-rendering /home/azureuse
 2967 azureuser  20   0 4520M  916M  154M R  8.7  0.7  0:09.78 pvpython --force-offscreen-rendering /home/azureuse
 2961 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.86 pvpython --force-offscreen-rendering /home/azureuse
 2956 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.95 pvpython --force-offscreen-rendering /home/azureuse
 2958 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.93 pvpython --force-offscreen-rendering /home/azureuse
 2962 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.86 pvpython --force-offscreen-rendering /home/azureuse
 2963 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.80 pvpython --force-offscreen-rendering /home/azureuse
 2965 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.80 pvpython --force-offscreen-rendering /home/azureuse
 2969 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.67 pvpython --force-offscreen-rendering /home/azureuse
 2968 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.69 pvpython --force-offscreen-rendering /home/azureuse
 2970 azureuser  20   0 4520M  916M  154M R  8.0  0.7  0:09.56 pvpython --force-offscreen-rendering /home/azureuse
 2959 azureuser  20   0 4520M  916M  154M R  7.3  0.7  0:09.87 pvpython --force-offscreen-rendering /home/azureuse
 2964 azureuser  20   0 4520M  916M  154M R  7.3  0.7  0:09.84 pvpython --force-offscreen-rendering /home/azureuse
 2966 azureuser  20   0 4520M  916M  154M R  7.3  0.7  0:09.77 pvpython --force-offscreen-rendering /home/azureuse
 4004 azureuser  20   0 33060  5292  3652 R  1.3  0.0  0:00.42 htop
 2953 azureuser  20   0 4520M  916M  154M S  0.0  0.7  0:00.63 pvpython --force-offscreen-rendering /home/azureuse
 1641 root       20   0  379M 26660 10164 S  0.0  0.0  0:07.09 python3 -u bin/WALinuxAgent-2.2.54.2-py2.7.egg -run
 2954 azureuser  20   0 4520M  916M  154M S  0.0  0.7  0:00.60 pvpython --force-offscreen-rendering /home/azureuse
   11 root       20   0     0     0     0 I  0.0  0.0  0:00.50 rcu_sched
 2952 azureuser  20   0 4520M  916M  154M S  0.0  0.7  0:00.59 pvpython --force-offscreen-rendering /home/azureuse
 3639 azureuser  20   0  105M  4996  3848 S  0.0  0.0  0:00.03 sshd: azureuser@pts/1
 2839 root       20   0     0     0     0 I  0.0  0.0  0:00.47 kworker/u32:1-mlx5_cmd_6d84:00:02.0
  355 root       20   0     0     0     0 I  0.0  0.0  0:00.05 kworker/12:1-mm_percpu_wq
 3430 root       20   0     0     0     0 I  0.0  0.0  0:00.06 kworker/u32:2-events_power_efficient
  884 root       20   0     0     0     0 I  0.0  0.0  0:00.07 kworker/1:2-events
 1769 root       20   0  379M 26660 10164 S  0.0  0.0  0:00.60 python3 -u bin/WALinuxAgent-2.2.54.2-py2.7.egg -run
 1284 messagebu  20   0 50096  4516  3796 S  0.0  0.0  0:00.27 /usr/bin/dbus-daemon --system --address=systemd: --
  736 root       20   0 14200  2900  1892 S  0.0  0.0  0:01.54 /usr/lib/linux-tools/5.4.0-1046-azure/hv_kvp_daemon
    1 root       20   0 77972  8920  6540 S  0.0  0.0  0:02.34 /sbin/init
 1043 systemd-r  20   0 70656  5072  4512 S  0.0  0.0  0:00.38 /lib/systemd/systemd-resolved
 1203 syslog     20   0  263M  4168  3624 S  0.0  0.0  0:00.21 /usr/sbin/rsyslogd -n
 1770 root       20   0  379M 26660 10164 S  0.0  0.0  0:00.13 python3 -u bin/WALinuxAgent-2.2.54.2-py2.7.egg -run
 1938 root       20   0  105M  7176  6160 S  0.0  0.0  0:00.02 sshd: azureuser [priv]
 2229 azureuser  20   0  105M  5044  3904 S  0.0  0.0  0:00.03 sshd: azureuser@pts/0
    2 root       20   0     0     0     0 S  0.0  0.0  0:00.00 kthreadd
    3 root        0 -20     0     0     0 I  0.0  0.0  0:00.00 rcu_gp
    4 root        0 -20     0     0     0 I  0.0  0.0  0:00.00 rcu_par_gp
    6 root        0 -20     0     0     0 I  0.0  0.0  0:00.00 kworker/0:0H-kblockd
    9 root        0 -20     0     0     0 I  0.0  0.0  0:00.00 mm_percpu_wq
```

Figure 52: CPU consumption in the server

In **Figure 53** you can see the network consumption in Chrome while rotating both datasets. The reader can see that the server is sending multiple jpeg files to the client.



Name	Status	Type	Initiator	Size	Time	Waterfall
blob:http://51.103.138.135/c4352...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/6247...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/902d...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/30a46...	200	jpeg	ViewStream.js:44	0 B	11 ms	
blob:http://51.103.138.135/6b0d...	200	jpeg	ViewStream.js:44	0 B	13 ms	
blob:http://51.103.138.135/783f5...	200	jpeg	ViewStream.js:44	0 B	4 ms	
blob:http://51.103.138.135/f1d20...	200	jpeg	ViewStream.js:44	0 B	3 ms	
blob:http://51.103.138.135/609b...	200	jpeg	ViewStream.js:44	0 B	3 ms	
blob:http://51.103.138.135/ded7...	200	jpeg	ViewStream.js:44	0 B	227 ms	
blob:http://51.103.138.135/9bcd...	200	jpeg	ViewStream.js:44	0 B	10 ms	
blob:http://51.103.138.135/623bf...	200	jpeg	ViewStream.js:44	0 B	6 ms	
blob:http://51.103.138.135/2e42...	200	jpeg	ViewStream.js:44	0 B	3 ms	
blob:http://51.103.138.135/1ae94...	200	jpeg	ViewStream.js:44	0 B	5 ms	
blob:http://51.103.138.135/68c84...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/c1007...	200	jpeg	ViewStream.js:44	0 B	4 ms	
blob:http://51.103.138.135/cb04...	200	jpeg	ViewStream.js:44	0 B	3 ms	
blob:http://51.103.138.135/9289...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/aeecc...	200	jpeg	ViewStream.js:44	0 B	2 ms	
blob:http://51.103.138.135/57bcf...	(pending)		ViewStream.js:44	0 B	Pending	

19 requests | 0 B transferred | 1.8 MB resources

Figure 53: Network consumption in the browser while rotating

5.2. Meshing results

In this subsection, a study case is presented for the meshing functionality of Evoker. First, the configuration of the OpenFOAM project is explained, and then the results and the evaluation of the recommended number of processes for this study case.

5.2.1. Meshing configuration

As explained in Section “**4.5 Files preparation”**, there is some values automatically configured by Evoker behind the scenes. Particularly, Evoker has configured the project to work with *castellatedMesh* and *snap* methods. The castellatedMesh execution of the pipeline will use the refinement configuration set by the user in the GUI of Evoker. On top of that, the engineer using Evoker will have configured OpenFOAM project with below snap configuration settings in *snappyHexMeshDict* file:

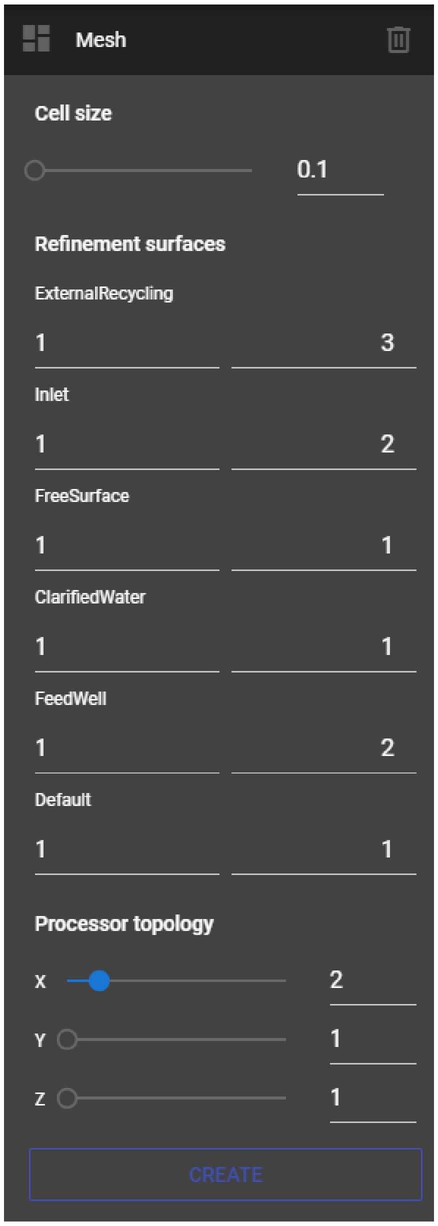
```
snapControls {
        nSmoothPatch 3;
        tolerance     1;
        nSolveIter    300;
        nRelaxIter    5;
        nFeatureSnapIter     10;
        implicitFeatureSnap true;
        explicitFeatureSnap false;
}
```

For our study case, two problems size are defined. The problem size is set in the GUI of Evoker by the cell size parameter:

1) Cell size = 0.1 meters. This is the big size problem definition and is using 185 x 55 x 185 cells, in the X, Y, and Z coordinates respectively, for a total of 1.882.375 cells.

2) Cell size = 0.2 meters. This is the small size problem definition and is using 92 x 27 x 92 cells, in the X, Y, and Z coordinates respectively, for a total of 228528 cells.

And in **Figure 54** the reader can see a screenshot of the refinement surfaces values:


Mesh

Cell size

0.1

Refinement surfaces

ExternalRecycling	1	3
Inlet	1	2
FreeSurface	1	1
ClarifiedWater	1	1
FeedWell	1	2
Default	1	1

Processor topology

X 2

Y 1

Z 1

Figure 54: HPC meshing study case GUI settings

5.2.2. Performance analysis and evaluation

The mesh functionality of Evoker has been executed with the below combinations and results:

5.2.2.1 Problem size: 1.8M cells

Topology	1 node		2 nodes		3 nodes		4 nodes		Optimum	
	ppn	Time(s)	ppn	Time(s)	ppn	Time(s)	ppn	Time(s)	Time(s)	SpeedUp
1x1x1	1	665.75							665.75	1
2x1x1	2	459.02	1	549					459.02	1.45
2x1x2	4	456.15	2	350.17	2	355.42			350.17	1.9
4x1x2			4	326.65	3	324.18			324.18	2.05
4x1x3					3	281.55			281.55	2.36
4x1x4							4	262.9	262.9	2.53

Figure 55: Tabular results of mesh study case with 1.8M cells

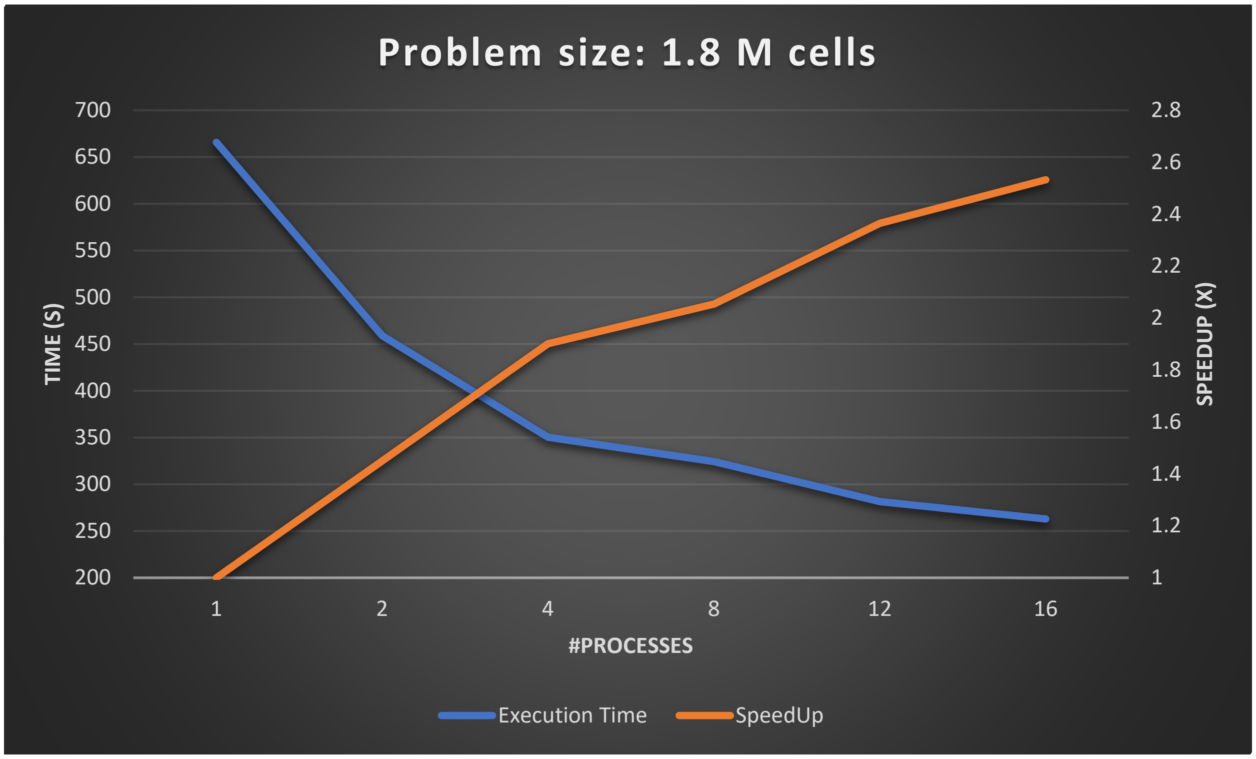


Figure 56: Graphical results of mesh study case with 1.8M cells

For this problem size of 1.8M cells, the optimum processes configuration is with 4 processes. So, with 1 to 4 processes we preserve a strong scaling concept, where if the number of processes/streams is increased, the efficiency can be maintained without the need to increase the size of the problem. Above 4 processes, the gain in performance is negligible and it does not pay off to dedicate so much hardware resources for the problem at hand.

The graphical results show both time and speedup. SpeedUp is a commonly used metric for strong scalability analysis. SpeedUp is defined as the quotient between the serial time and the parallel time of the configuration being evaluated:

$$S = \frac{T_{serial}}{T_{parallel}}$$

where T_{serial} is the time for the one process configuration and $T_{parallel}$ is any configuration where processes are greater than one.

6.2.2.2 Problem size: 0.2M cells

Topology	1 node		2 nodes		3 nodes		4 nodes		Optimum	
	ppn	Time(s)	ppn	Time(s)	ppn	Time(s)	ppn	Time(s)	Time(s)	SpeedUp
1x1x1	1	121.82							121.82	1
2x1x1	2	94.16	1	111.56					94.16	1.29
2x1x2	4	90.57	2	77.84	2	85.18			77.84	1.57
4x1x2			4	86.5	3	84.03			84.03	1.45
4x1x3					3	77.51			77.51	1.57
4x1x4							4	86.67	86.67	1.4

Figure 57: Tabular results of mesh study case with 0.2M cells

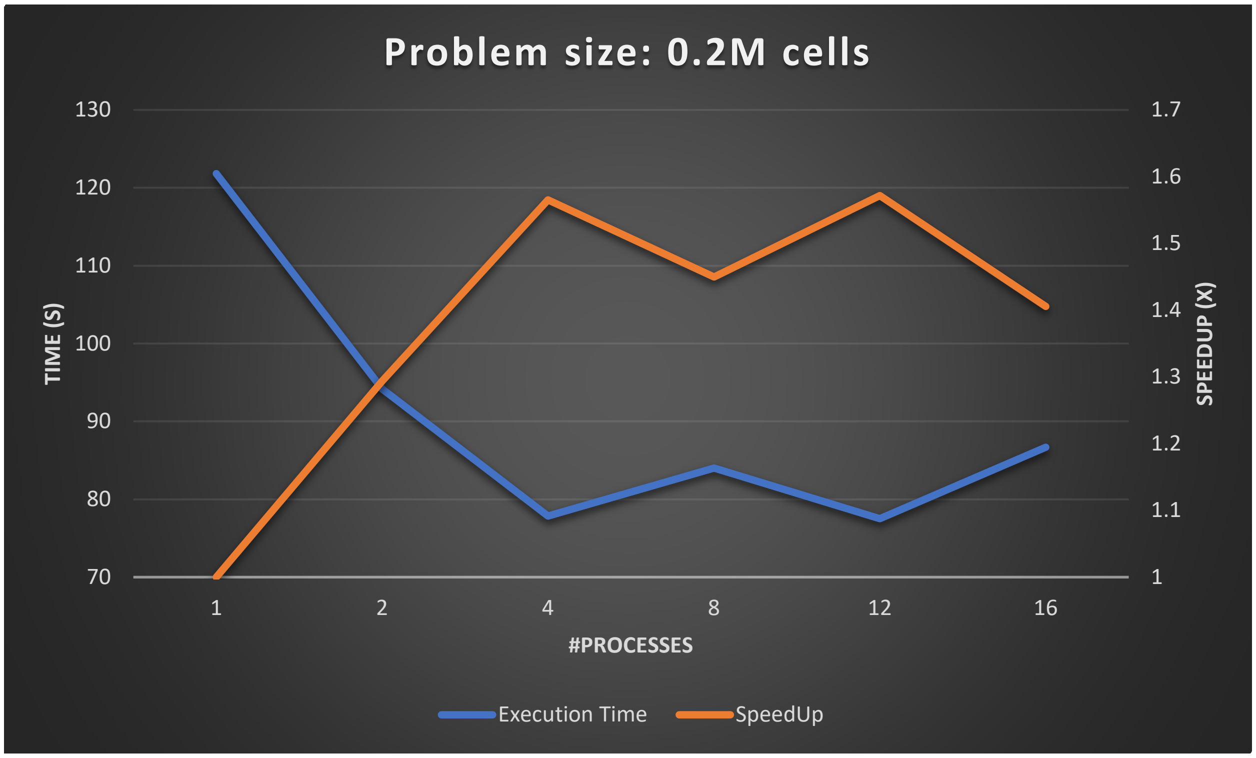


Figure 58: Graphical results of mesh study case with 0.2M cells

For this problem size of 0.2M cells, the optimum processes configuration is also with 4 processes. So, with 1 to 4 processes we preserve a strong scaling concept, where if the number of processes/streams is increased, the efficiency can be maintained without the need to increase the size of the problem. Above 4 processes, the gain in performance is negligible and it does not pay off to dedicate so much hardware resources for the problem at hand.

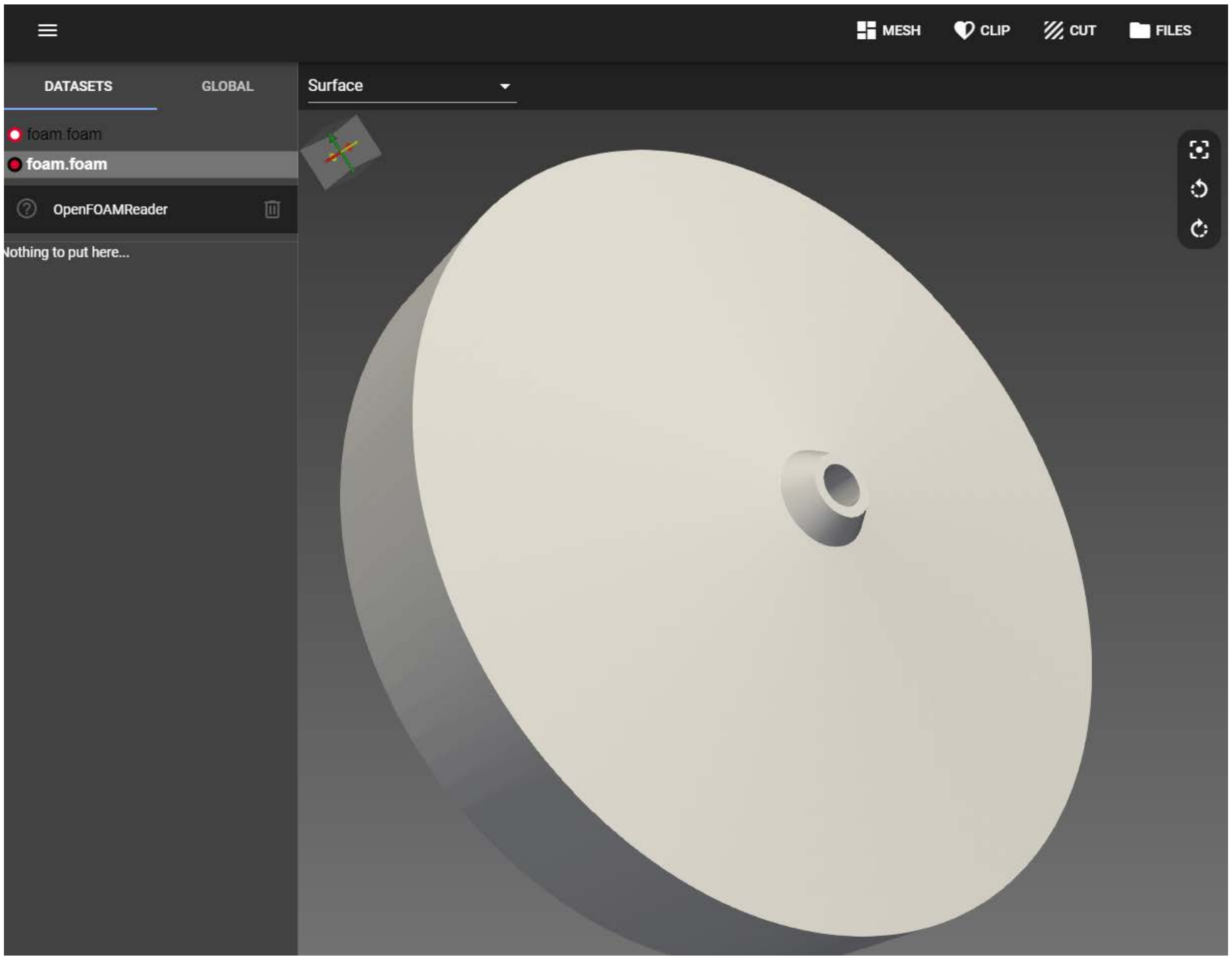


Figure 59: Project meshed with cell size 0.1

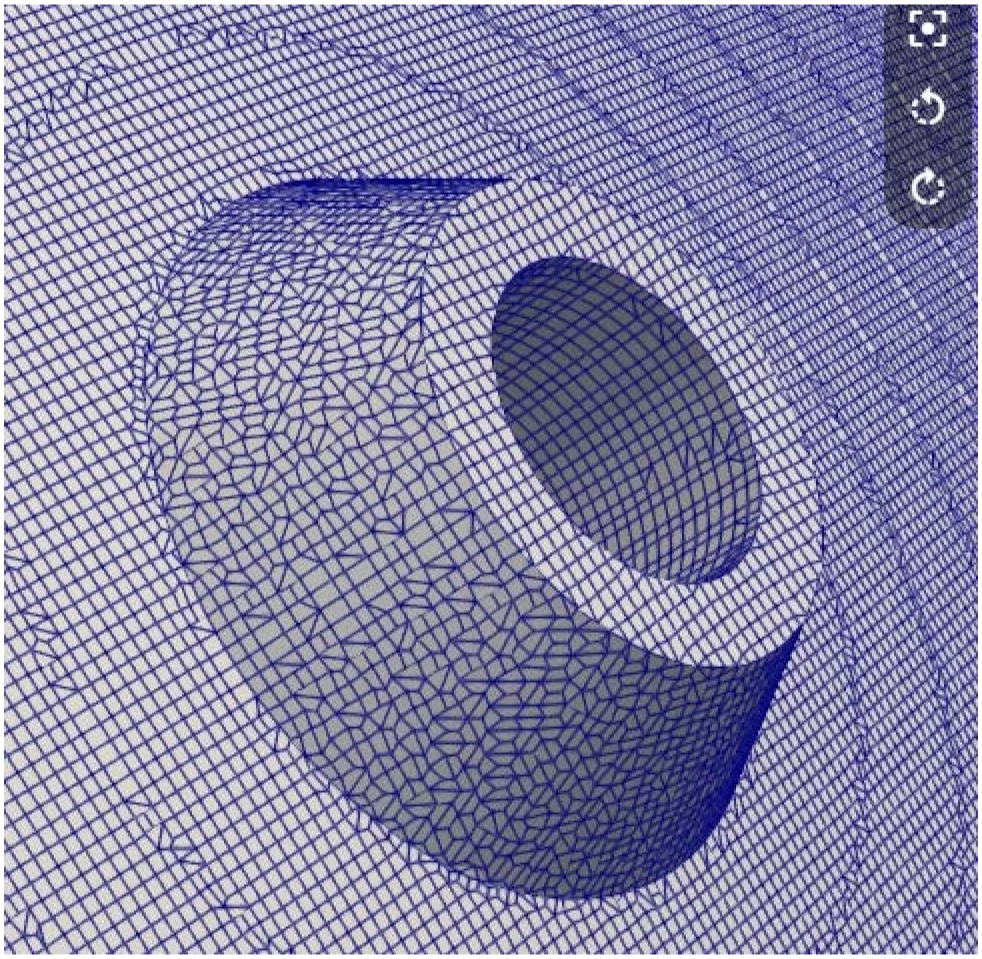


Figure 60: Surface with edges view with cell size 0.1

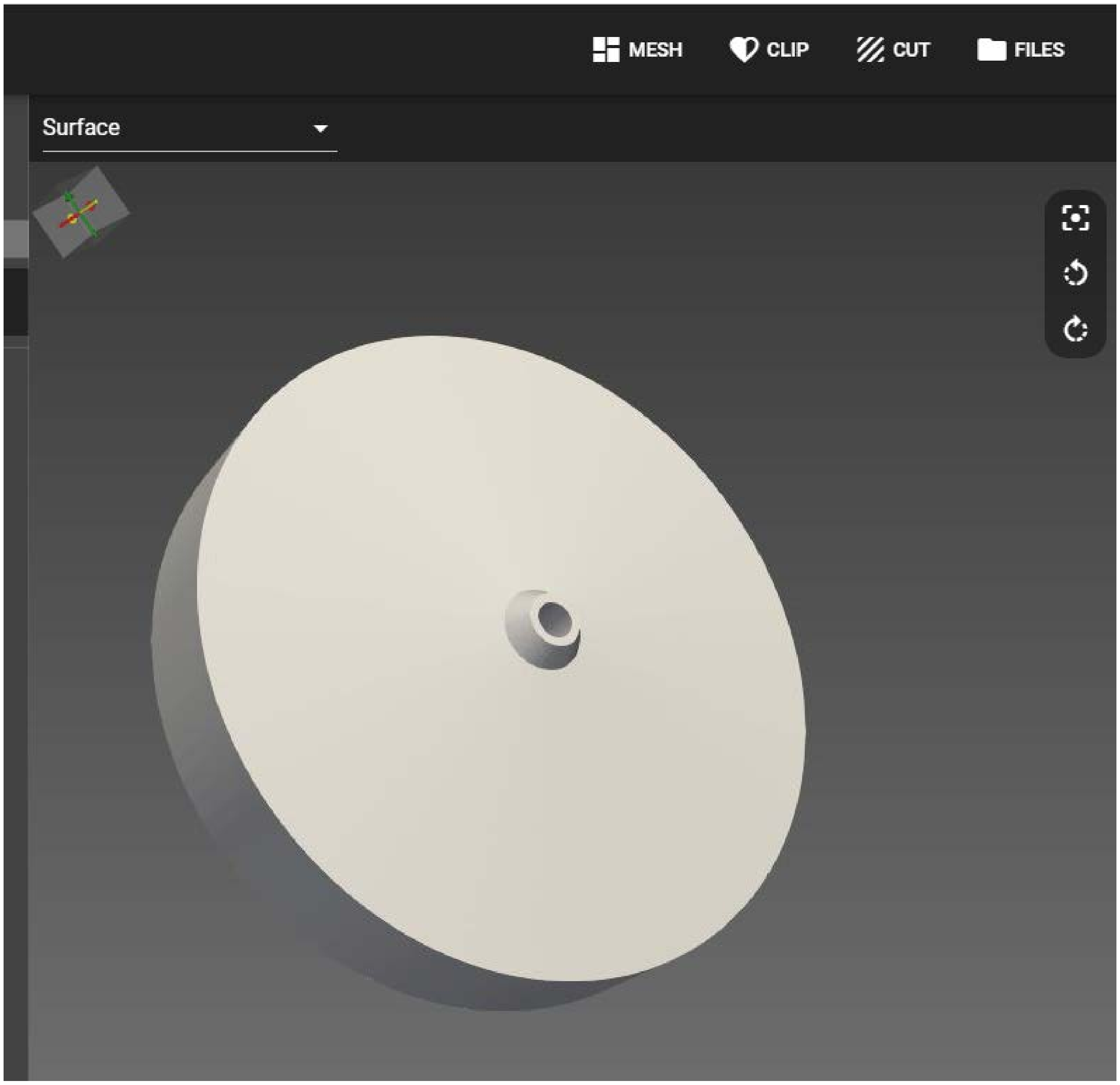


Figure 61: Project meshed with cell size 0.2

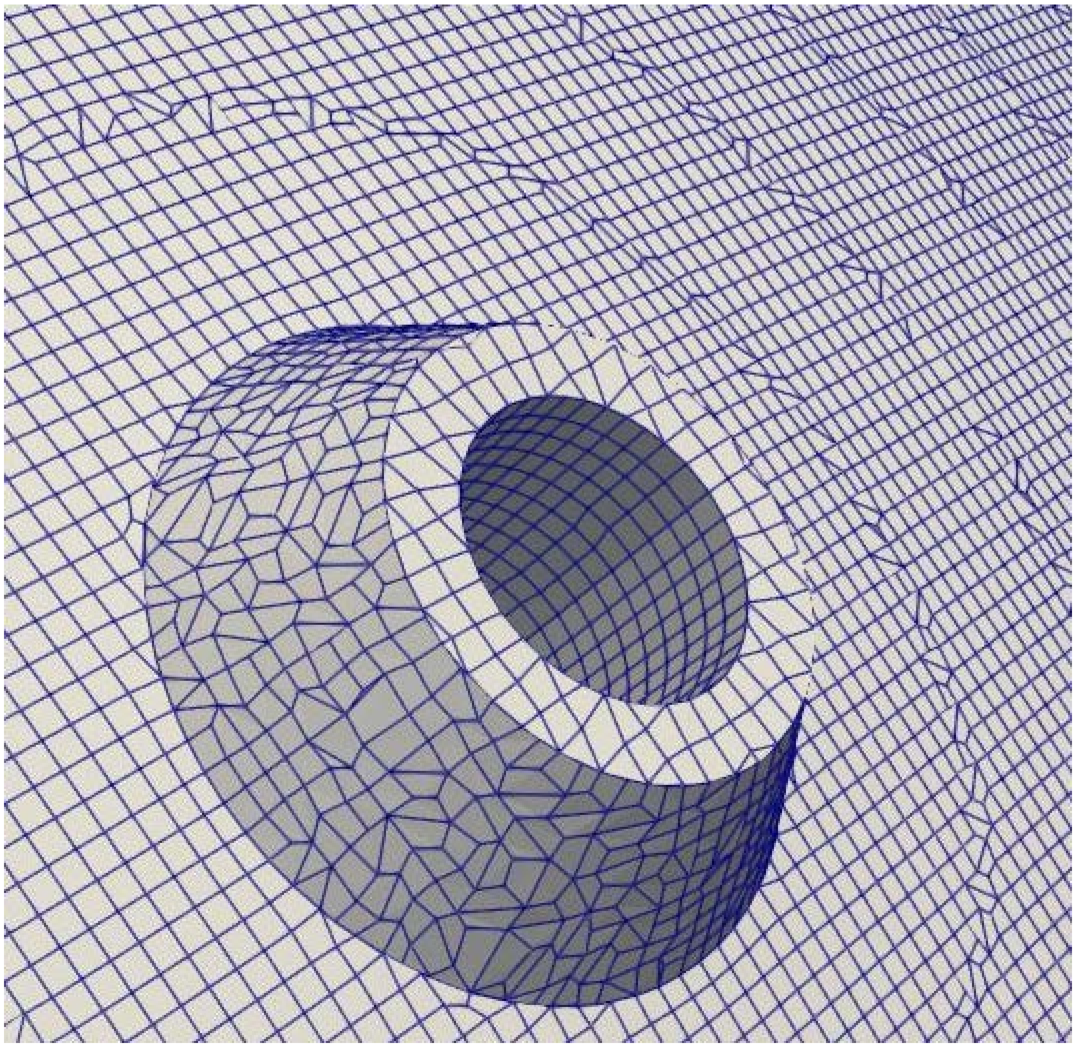


Figure 62: Surface with edges view with cell size 0.2

In **Figure 63**, the reader can see another OpenFOAM project different from the case study running in Evoker.

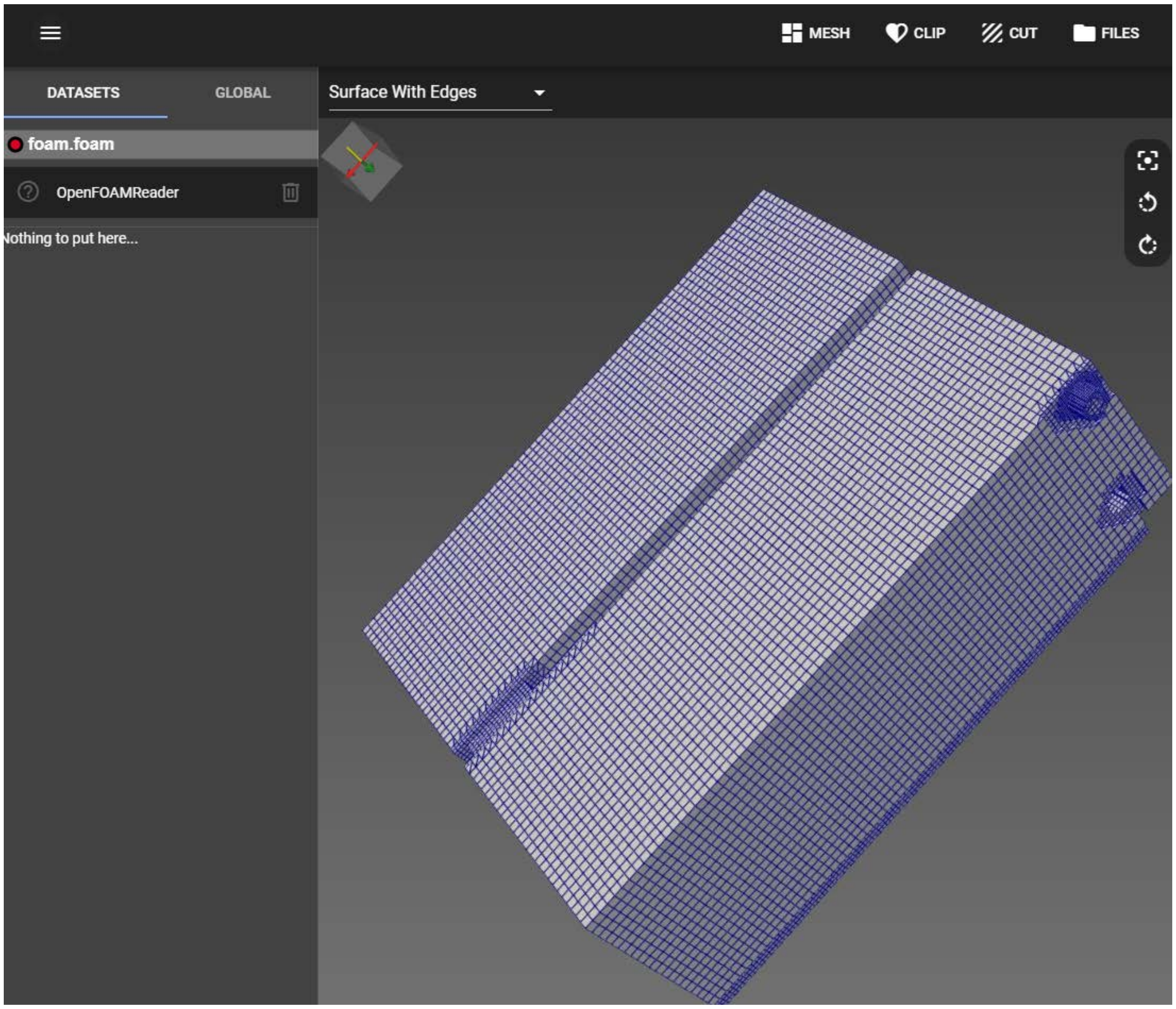


Figure 63: Another OpenFOAM project in Evoker

5.3. Browser support

Evoker has been run on Chrome, Firefox, Opera, and Edge browsers. As expected by its technology and advanced JavaScript framework it has been executed successfully on those browsers (see screenshots below).

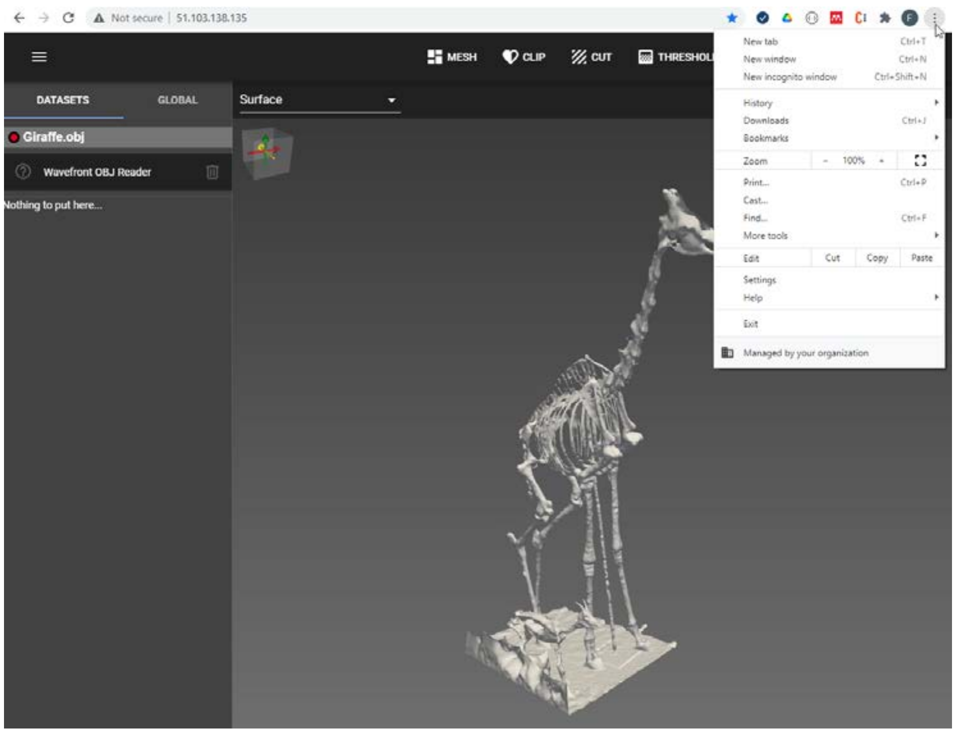


Figure 64: Evoker running on Chrome

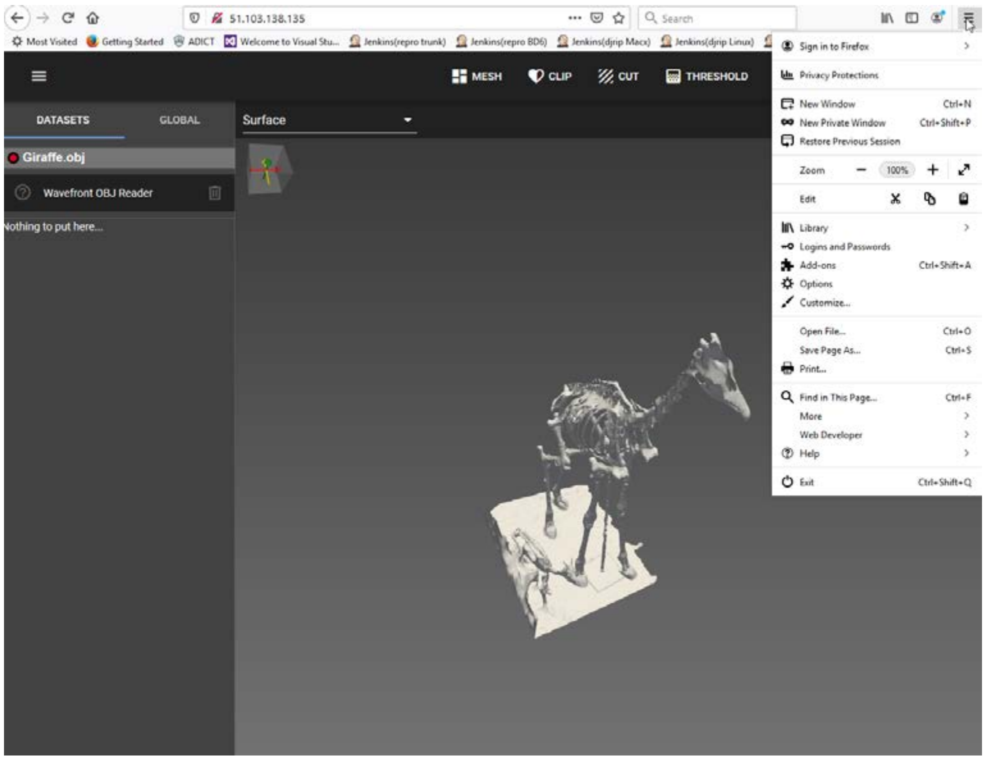


Figure 65: Evoker running on Firefox

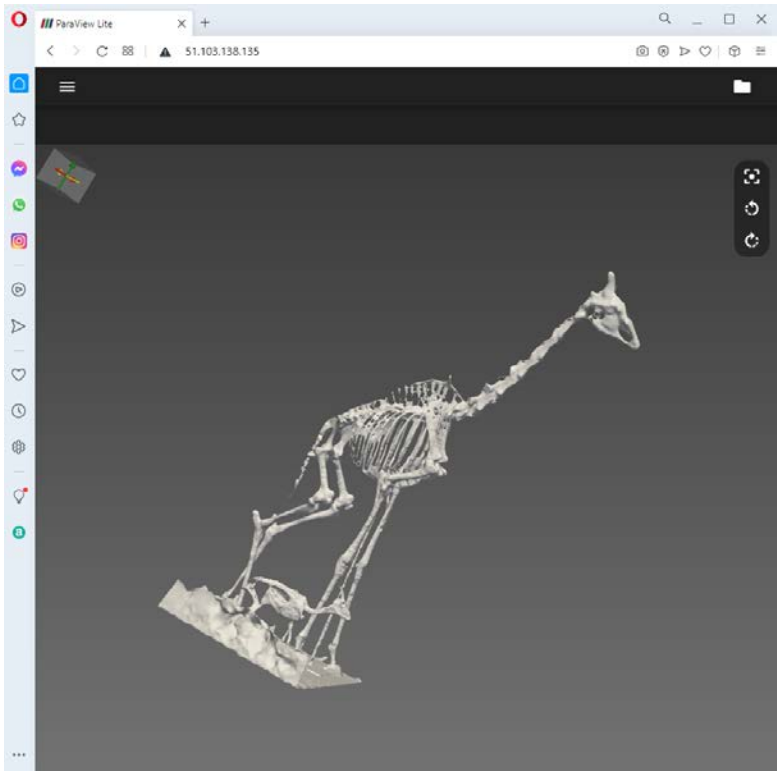
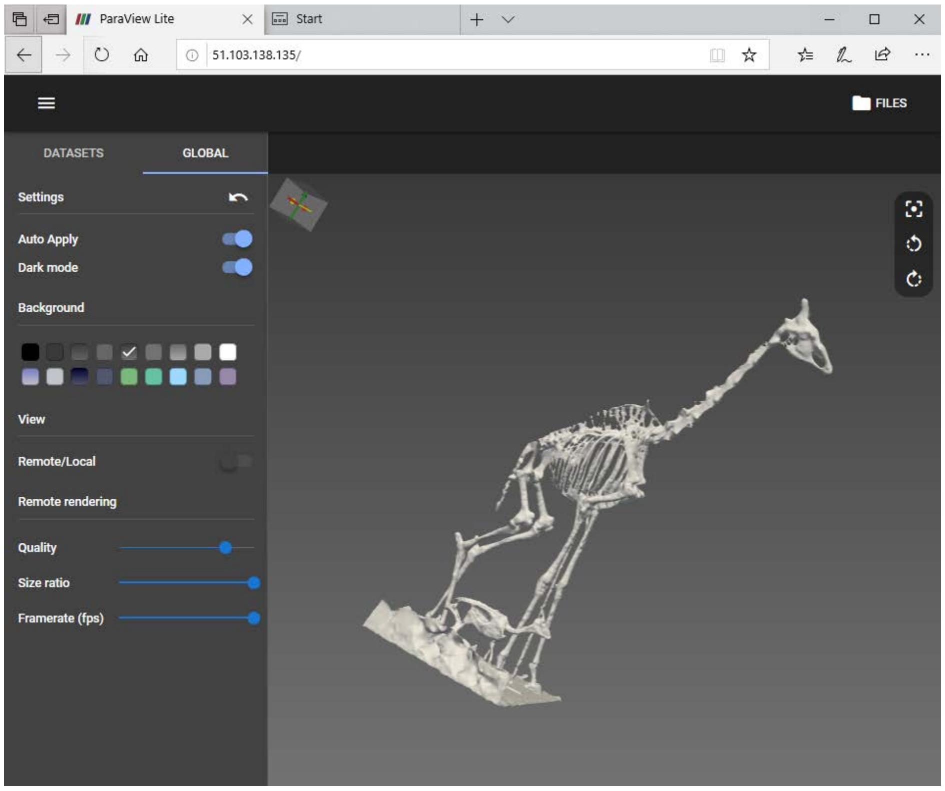


Figure 66: Evoker running on Opera



Settings

Auto Apply

Dark mode

Background

View

Remote/Local

Remote rendering

Quality

Size ratio

Framerate (fps)

Figure 67: Evoker running on Edge

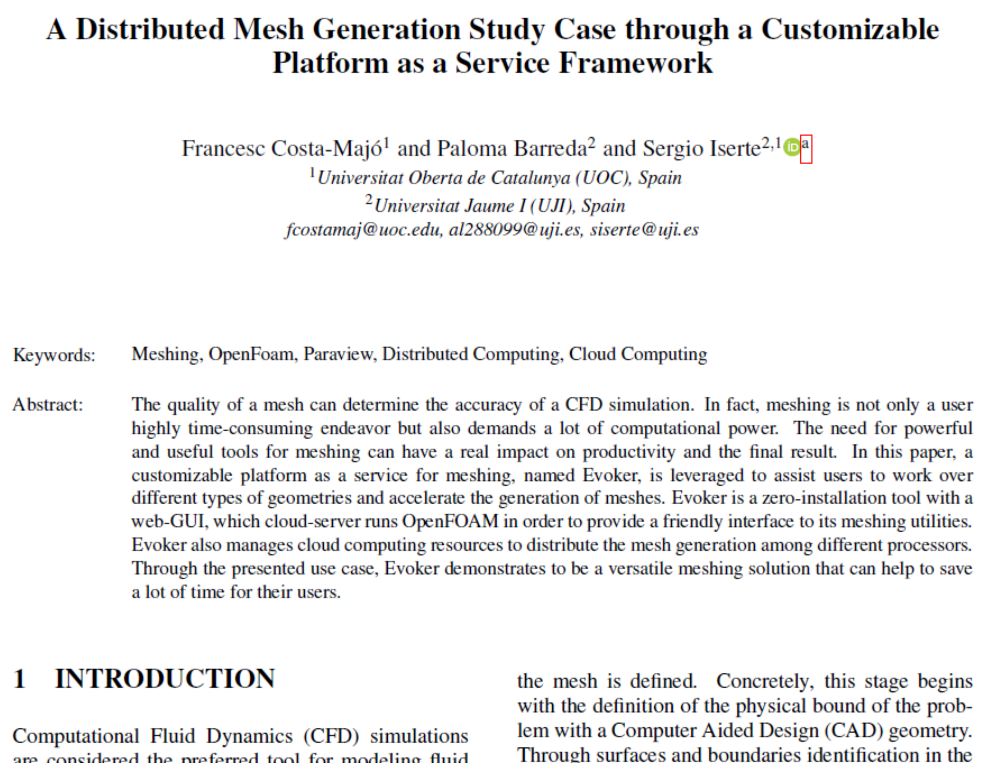
6. Conclusions

This FMP has had a two-fold objective. The first objective has been the analysis, design, build and configuration of a web-based client-server 3D visualizer solution rendering in the backend using VTK and ParaView frameworks. The solution performs very well from the user experience point of view even in low-performance devices thanks to its architecture. It has been shown how to deploy it in Azure and how to accommodate different performance results as per the available budget. The solution has been used by a hydrodynamic engineer having not pinpoint that this was an academic solution, and not developed by a software company.

The other objective of the project is mesh generation. This part of the project has been the implantation of CFD technologies leveraging PaaS techniques in cloud computing environments. Evoker's users can configure the meshing parameters targeted to their needs. Evoker transparently modifies OpenFOAM files in the user project for applying meshing refinements. A case study has been performed looking at the best processes' configuration for strong scalability. Evoker has been found useful for increasing productivity in meshing tasks in targeted problems. It is worth mentioning that Evoker can scale up and down considering the problem size, the resource requirements, and the budget limitations. Meshing a CFD project is a complicated process that requires iterations until the desired solution is achieved; this behavior of CDF projects makes paramount a versatile tool with the capacity of scaling, and consequently, decreasing the computation time. Performing a scalability analysis to determine an efficient processes' topology, and then using that configuration to generate the meshes quickly, can save a lot of time for the user.

6.1. Paper submitted in SIMULTECH 2021

As a result of this project, my colleagues Paloma Barreda, Sergio Iserte, and the author of this FMP have submitted a paper to the 11th International Conference on Simulation and Modeling (SIMULTECH 2021) as you can see in **Figure 68**. The paper has been accepted as **Short Paper [67]**.


A Distributed Mesh Generation Study Case through a Customizable Platform as a Service Framework

Francesc Costa-Majó[1] and Paloma Barreda[2] and Sergio Iserte[2,1][a]
[1]Universitat Oberta de Catalunya (UOC), Spain
[2]Universitat Jaume I (UJI), Spain
fcostamaj@uoc.edu, al288099@uji.es, siserte@uji.es

Keywords: Meshing, OpenFoam, Paraview, Distributed Computing, Cloud Computing

Abstract: The quality of a mesh can determine the accuracy of a CFD simulation. In fact, meshing is not only a user highly time-consuming endeavor but also demands a lot of computational power. The need for powerful and useful tools for meshing can have a real impact on productivity and the final result. In this paper, a customizable platform as a service for meshing, named Evoker, is leveraged to assist users to work over different types of geometries and accelerate the generation of meshes. Evoker is a zero-installation tool with a web-GUI, which cloud-server runs OpenFOAM in order to provide a friendly interface to its meshing utilities. Evoker also manages cloud computing resources to distribute the mesh generation among different processors. Through the presented use case, Evoker demonstrates to be a versatile meshing solution that can help to save a lot of time for their users.

1 INTRODUCTION

Computational Fluid Dynamics (CFD) simulations are considered the preferred tool for modeling fluid

the mesh is defined. Concretely, this stage begins with the definition of the physical bound of the problem with a Computer Aided Design (CAD) geometry. Through surfaces and boundaries identification in the

Figure 68: Paper submitted to SIMULTECH 2021

6.2. Source code

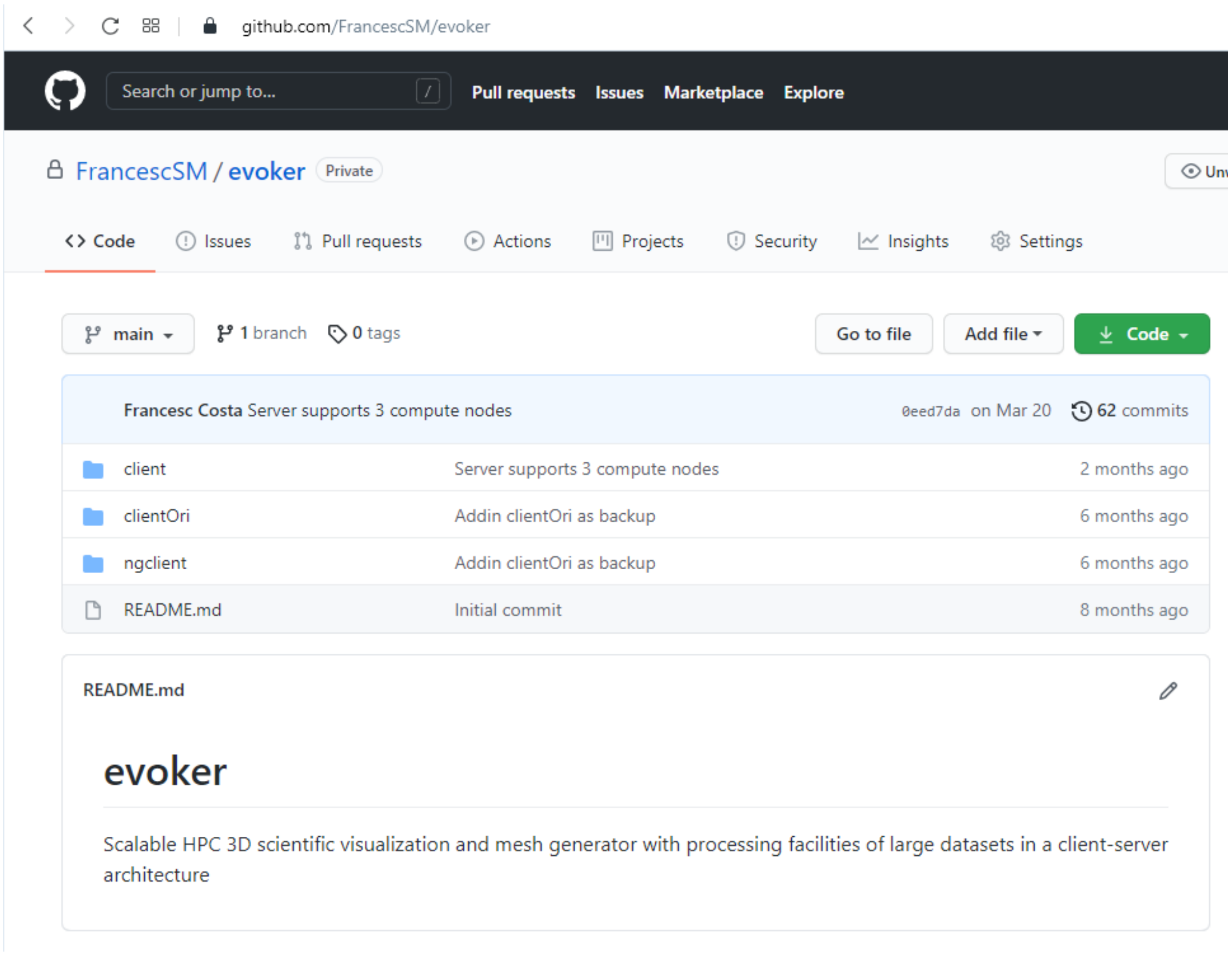
The source code of Evoker is hosted in a private repository in github (**Figure 69**). The interested reader can get access by requesting it from the author.

The repository has three main folders:

- **client**: Evoker current source code
- **clientOri**: ParaView code for reference
- **ngclient**: deprecated initial attempt to implement Evoker in Angular.

Find below the command to clone the repository:

```
$ git clone https://github.com/FrancescSM/evoker.git
```



Francesc Costa Server supports 3 compute nodes 0eed7da on Mar 20 62 commits

client	Server supports 3 compute nodes	2 months ago
clientOri	Addin clientOri as backup	6 months ago
ngclient	Addin clientOri as backup	6 months ago
README.md	Initial commit	8 months ago

README.md

evoker

Scalable HPC 3D scientific visualization and mesh generator with processing facilities of large datasets in a client-server architecture

Figure 69: Evoker source code hosted in github

7. Bibliography

[1] C. Senn, “The Pillars of Industry 4.0,” 2019. https://www.idashboards.com/blog/2019/07/31/the-pillars-of-industry-4-0/.

[2] L. Rose, S. Jourdain, and P. O’Leary, “Vis on the Web,” 2019. https://blog.kitware.com/vis-on-the-web/.

[3] K. Ismail, “Edge Computing vs. Fog Computing: What’s the Difference?” 2018. https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/.

[4] Dr. Leif Kobbelt, “Open Flipper” https://www.graphics.rwth-aachen.de/software/openflipper/

[5] Kitware, “ParaView Glance“: https://kitware.github.io/paraview-glance/app/

[6] Kitware, “ParaView Visualizer“: https://kitware.github.io/visualizer/

[7] Kitware, “ParaView Lite“: https://kitware.github.io/paraview-lite/index.html

[8] Weapon, “VisIt”: https://wci.llnl.gov/simulation/computer-codes/visit

[9] 3DHOP: https://3dhop.net/index.php

[10] GMSH: https://gmsh.info/

[11] SimScale. (2021). Simulation Software: https://www.simscale.com

[12] Ingrid Cloud. (2021). Professional Wind Simulations. https://www.ingridcloud.com/

[13] Kitware, “VTK overview.” https://vtk.org/about/#overview.

[14] W. J. Schroeder and K. M. Martin, “The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics”, 4th Edition.

[15] Kitware, “VTK support.” https://discourse.vtk.org

[16] Kitware, “Visualize Your Data With VTK.js,” 2021. https://kitware.github.io/vtk-js/index.html

[17] Kitware, “VTK web,” 2018, [Online]. Available: https://vtk.org/flavors/

[18] Kitware, “Large Data Visualization Made Easier,” 2019. https://www.paraview.org/overview/

[19] Kitware, “Confusion with tool selection: VTK vs ParaView for Web-based volume rendering of large data sets,” 2019. https://discourse.vtk.org/t/confusion-with-tool-selection-vtk-vs-paraview-for-web-based-volume-rendering-of-large-data-sets/1384/2

[20] E. You, “Vue.js,” 2021. https://vuejs.org/

[21] B. Nelson, *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*. 2018.

[22] Michael Sullivan, "Learning Vue.js Get to know the Vue.JS Javascript framework." https://www.linkedin.com/learning/learning-vue-js-3/get-to-know-the-vue-js-javascript-framework?u=2138274.

[23] Kitware, "Building ParaView," 2021. https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md.

[24] P. P. Wiki, "ParaView:Build And Install." https://www.paraview.org/Wiki/ParaView:Build_And_Install.

[25] VTK Public Wiki, "VTK/Building/Linux," 2019. https://vtk.org/Wiki/VTK/Building/Linux.

[26] Kitware, "Discourse ParaView," 2021, [Online]. Available: https://discourse.paraview.org/.

[27] OpenFOAM. (2021). The open source CFD toolbox: https://www.OpenFOAM.com/documentation/user-guide/index.php.

[28] MPICH, "MPICH," 2020. https://www.mpich.org/.

[29] ZoomAdmin, "How To Install 'mpich' Package on Ubuntu." https://zoomadmin.com/HowToInstall/UbuntuPackage/mpich.

[30] OpenFOAM, "Installation on Ubuntu." https://OpenFOAM.org/download/7-ubuntu/

[31] Kitware, "Python/JavaScript library for communicating over WebSocket" https://github.com/Kitware/wslink

[32] Garg, A. (2018). "How to Quickly Clone a VM in Azure !" https://www.techmanyu.com/how-to-quickly-clone-a-vm-in-azure-c5299ff82496

[33] Jetha, H. (2018). "Cómo configurar las llaves SSH en Ubuntu 18.04. " https://www.digitalocean.com/community/tutorials/como-configurar-las-llaves-ssh-en-ubuntu-18-04-es

[34] Microsoft. (2020). "What is Azure Files? " https://docs.microsoft.com/en-us/azure/storage/files/storage-files-introduction

[35] Microsoft. (2021). "Create an Azure file share." https://docs.microsoft.com/en-us/azure/storage/files/storage-how-to-create-file-share?tabs=azure-portal

[36] Microsoft. (2019). "Use Azure Files with Linux." https://docs.microsoft.com/en-us/azure/storage/files/storage-how-to-use-files-linux

[37] J. P. Ahrens, J. Woodring, D. E. DeMarle, J. Patchett, and M. Maltrud, "Interactive remote large-scale data visualization via prioritized multi-resolution streaming," in *Proceedings of the*

2009 Workshop on Ultrascale Visualization - UltraVis '09, 2009, pp. 1–10, doi: 10.1145/1838544.1838545.

[38] J. Ahrens *et al.*, "The ParaView Guide Community Edition Utkarsh Ayachit With contributions from," 2020. [Online]. Available: http://paraview.orghttp//kitware.com.

[39] A. M. Boutsi, C. Ioannidis, and S. Soile, "INTERACTIVE ONLINE VISUALIZATION of COMPLEX 3D GEOMETRIES," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Jan. 2019, vol. 42, no. 2/W9, pp. 173–180, doi: 10.5194/isprs-archives-XLII-2-W9-173-2019.

[40] DigitalOcean, "ssh-copy-id not working Permission denied (publickey)," 2017. https://www.digitalocean.com/community/questions/ssh-copy-id-not-working-permission-denied-publickey.

[41] J. Doellner, B. Hagedorn, and J. Klimke, "Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps," in *Proceedings, Web3D 2012 - 17th International Conference on 3D Web Technology*, 2012, pp. 97–100, doi: 10.1145/2338714.2338729.

[42] Finet Julien, "Kitware Web tools Webinar," 2020. https://vimeo.com/467338846.

[43] FoamDude, "OpenFOAM: SnapyHexMesh - Castellated - YouTube," 2017. https://www.youtube.com/watch?v=ICJxysKTnVw.

[44] Francesc Costa, "Unknown argument: Support for 64 bit file systems," 2021. https://discourse.paraview.org/t/unknown-argument-support-for-64-bit-file-systems/5568.

[45] A. Garg, "How to Quickly Clone a VM in Azure !," 2018, [Online]. Available: https://www.techmanyu.com/how-to-quickly-clone-a-vm-in-azure-c5299ff82496.

[46] M. Garland and P. S. Heckbert, "Surface Simplification Using Quadric Error Metrics." [Online]. Available: http://www.cs.cmu.edu/.

[47] C. Geuzaine and J.-F. Remacle, "gmsh_general_overview," 2020.

[48] Hanif Jetha, "Cómo configurar las llaves SSH en Ubuntu 18.04," 2018. https://www.digitalocean.com/community/tutorials/como-configurar-las-llaves-ssh-en-ubuntu-18-04-es.

[49] D. E. F. Holtom, *Enjoy Writing Your Science Thesis or Dissertation!* 1999.

[50] S. Jourdain, U. Ayachit, and B. Geveci, "ParaViewWeb: A Web Framework for 3D Visualization and Data Processing," 2011. [Online]. Available: http://www.mirlabs.net/ijcisim/index.html.

[51] József Nagy, "How to open OpenFOAM® results in ParaView - YouTube," 2014. https://www.youtube.com/watch?v=8J59CpaYnVc.

[52] Kitware, "Introduction to VTK on Vimeo," 2011. https://vimeo.com/32232190.

[53] P. Lindstrom, "Out-of-Core Simplification of Large Polygonal Models," 2000. [Online]. Available: www.cc.gatech.edu/.

[54] OpenFOAM, "OpenFOAM user guide," no. December, 2020.

[55] OpenFOAM, "OpenFOAM: User Guide: snappyHexMesh," 2006. https://www.OpenFOAM.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh.html.

[56] A. Razoumov, "Web-based 3D scientific visualization," 2020. [Online]. Available: https://www.youtube.com/watch?v=7aOF8BYIl5A.

[57] A. Razoumov, "Using ParaViewWeb for 3D visualization and data analysis in a web browser," 2017, [Online]. Available: https://www.youtube.com/watch?v=IUN5ln_XnA8.

[58] O. Rosquist, J. Luttu, K. Skolan, F. Datavetenskap, and O. Kommunikation, "Analysis of the performance difference between server-side and client-side rendering for data visualization in real-time using d3. js."

[59] Sarangarjan V, "What is OpenFOAM? | Skill-Lync - YouTube," 2018. https://www.youtube.com/watch?v=lEOrwfu1xhs.

[60] B. Sawicki and B. Chaber, "Efficient visualization of 3D models by web browser," in *Computing*, May 2013, vol. 95, no. SUPPL.1, doi: 10.1007/s00607-012-0275-z.

[61] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," 1992, pp. 65–70, doi: 10.1145/133994.134010.

[62] VTK/Kitware, "Generating 3d body mesh from pointcloud - Support - VTK," 2019. https://discourse.vtk.org/t/generating-3d-body-mesh-from-pointcloud/545.

[63] O. Wiki, "Meshing," 2020. https://wiki.OpenFOAM.com/Meshing.

[64] M. Yirci and I. Ulusoy, "A comparative study on polygonal mesh simplification algorithms," in *2009 IEEE 17th Signal Processing and Communications Applications Conference*, Apr. 2009, pp. 736–739, doi: 10.1109/SIU.2009.5136501.

[65] https://www.solidworks.com

[66] https://www.facsa.com/idi-2/proyecto-hydrosludge

[67] *Costa-Majó, F., Barreda, P. and Iserte, S. A Distributed Mesh Generation Study Case through a Customizable Platform as a Service Framework. In Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2021)*