UNIVERSITÁ DEGLI STUDI DI GENOVA
SCUOLA POLITECNICA

DIME

Dipartimento di Ingegneria Meccanica, Energetica, Gestionale e dei Trasporti

Tesi di Laurea Magistrale
in
Ingegneria Meccanica - Energia e Aeronautica

Implementation of PyCycle and a hybrid mesh generation system in CEASIOMpy

Relatori:
Chiar.^{mo} Prof. Ing. Alessandro Bottaro
Dott. Ing. Jan B. Vos

Correlatore:
Ing. Giacomo Benedetti

Allievi:
Francesco Marcucci
Guido Vallifuoco

A.A. 2024-2025

Abstract

The main achievement of this master's thesis was the implementation of two additional modules within CEASIOMpy that enabled thermodynamic modeling of the engine and advanced mesh generation capabilities. The thesis successfully added the Pycycle module, allowing the calculation of boundary conditions at the engine outlet, and implemented a hybrid mesh generation system using Gmsh, Pentagrow, and the TetGen software. This allowed RANS (Reynolds-Averaged Navier-Stokes) simulations to be performed within CEASIOMpy, thus enabling viscous effects to be captured and the boundary layer to be solved. These new features were then tested through several CFD simulations with different configurations, demonstrating the reliability of the results obtained. Future developments will aim to reduce some of the limitations by including more engine types, improving simulation convergence and automating the RANS simulation setup for complex geometries.

Acknowledgements

The research presented in this paper has been performed in the framework of the Colossus project (Collaborative System of Systems Exploration of Aviation Products, Services & Business models) and has received funding from the European Union Horizon Europe Programme under grant agreement no 101097120. The Swiss participation in the Colossus project was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 22.00609.

Contents

1 Introduction

Fluid mechanics is concerned with the mechanics of fluids (gases, liquids), and its applications go across a wide range of disciplines including aerospace, geophysics, astrophysics and biology. In ancient Greece, Archimedes investigated buoyancy and formulated his famous law which today is still known as the Archimedes principle. Since the Middle Ages rapid advancement in fluid mechanics started with Leonardo da Vinci, followed by Isaac Newton, Blaise Pascal and Daniel Bernoulli among many others. Claude-Louis Navier and George Gabriel Stokes formulated the Navier-Stokes equations in the mid-19th century. Ludwig Prandtl and Theodore von Karman investigated boundary layers, and various other scientists, such as Osborne Reynolds and Andrey Kolmogorov advanced the understanding of turbulence. Many problems in fluid mechanics are still unsolved and are today addressed using numerical methods, which resulted in the discipline of Computational Fluid Dynamics (CFD).

Lewis Fry Richardson was in the 1920s one of the first persons to use finite differences to divide the physical space into cells for weather forecasting, but at that time computers did not exist. It was only in 1950 that it was possible to make a 24-hour weather forecast, which took 24 hours to compute. In the 1950's and 1960's CFD took off, first at the Los Alamos National Laboratory. The initial CFD codes were concerned with solving the potential flow equations, this moved to solving the 2D and 3D Euler equations in the 1970s and 1980s, and starting from the middle of the 1980s CFD codes became available to solve the 3D Navier Stokes equations. The advent of solving the Navier-Stokes equations was driven by the rapid increase in computational power, the reduction in costs of using CFD, and the rapid development of models to model turbulent flows.

The basic operations in a CFD simulation are well summarised in figure 1.1, which presents all the main steps to be taken to obtain the desired results.

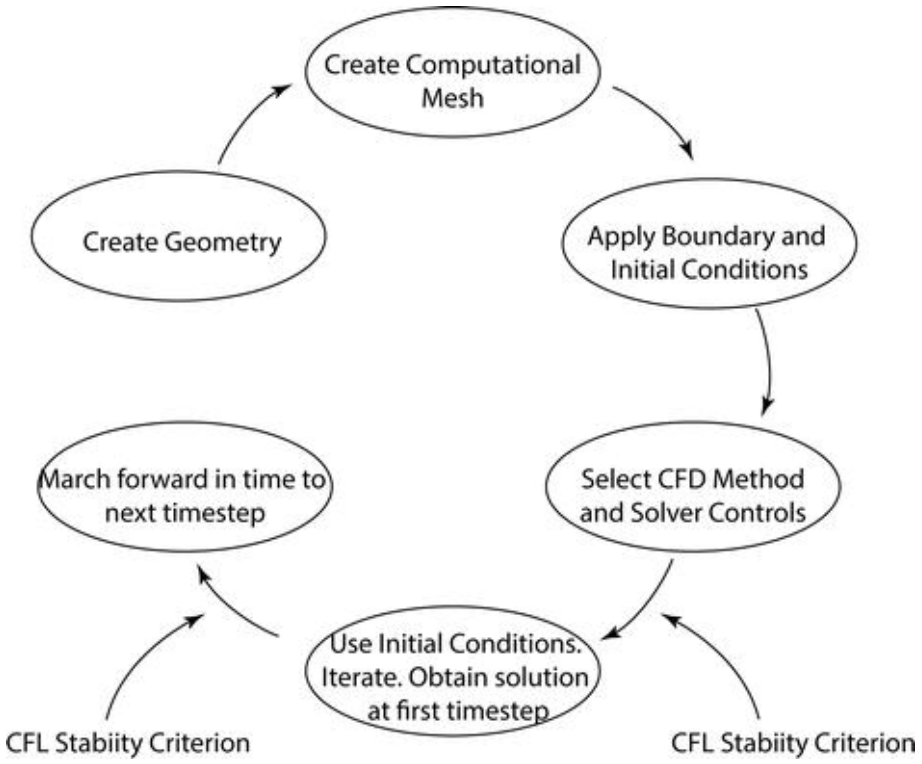


Figure 1.1: *Steps necessary to perform a CFD calculation [1]*

The work performed for this Master thesis consisted of improving the CEASIOMpy software, maintained by the company where it was carried out, CFS Engineering, in three specific areas of CFD analysis: mesh generation, boundary condition formulation and selection of the CFD software. These three areas are strongly interconnected and it was therefore also necessary to integrate them with each other.

At the start of the thesis work, CEASIOMpy permitted to perform Eulerian calculations automatically from a geometry, with this thesis a first attempt was made to integrate a method that allows RANS calculations. To do this, the mesh generation process had to include a method to resolve the boundary layer, and it was decided to use different interconnected open-source software to create a hybrid mesh. For the CFD software used, in this case, the Open Source software SU2, it was necessary to add an input file allowing RANS calculations to be performed automatically by inserting the necessary parameters such as the Reynolds number and viscous boundary conditions.

To model the boundary conditions of aircraft engines the CEASIOMpy software was coupled to the open source software called *PyCycle* developed by Eric Hendricks and Justin Gray, which allows the modelling of an aircraft engine and to obtain the thermodynamic parameters of interest that can be used in a CFD simulation.
The newly developed version of CEASIOMpy was then used to compare the performance of a conventional engine and one equipped with BLI (Boundary Layer Ingestion).

The BLI idea was first developed to reduce aircraft fuel consumption and better integrate the propulsion system with aerodynamics.
The thrust-specific fuel consumption (TSFC), which is the mass of fuel consumed when the engine generates a unit level of thrust $[\frac{kg}{N \cdot s}]$ is crucial to different aspects of civil aeronautics, a low TSFC allows a reduction of the aircraft-associated emissions and costs.

To reduce the fuel burn a large number of different approaches have been used in the last 20 to 30 years, transitioning from turbojet engines to turbofans with progressively higher bypass ratios, which has caused the diameters of the engines to increase to a size that can no longer be enlarged.
Beyond engine improvements, there have been large efforts to reduce the induced drag through the use of winglets, and to cut down the overall weight of the aircraft with the introduction of composite materials.

Today, the tendency is to try to uncouple the fan speed from the low-pressure turbine using a gearbox to transfer the power, allowing the fan to rotate at lower speeds avoiding the increase in drag caused by shock waves at the blade apex.
In this scenario of increasing the aerodynamic efficiency and to reduce the TSFC of the engines, we can place the engines with BLI, which tries to better integrate the vehicle's propulsion system with the rest of the airplane. BLI offers the opportunity to improve the current engine and aircraft configuration, integrating the engine nacelle into the aero frame.

The advantages of this technology have been known since the 1940s and have been applied to torpedoes, submarines, and ships. For about fifteen years, aircraft design has received much attention as a way to reduce fuel burn, which is encouraging the present work.

With an engine embedded in the fuselage or the wings, we can make the engine ingest the boundary layer and energize it as it passes through the aircraft engine. Due to ingestion of the boundary layer, the specific thrust decreases, allowing for an increase in propulsion efficiency. At the same time, we have to take into account the distortion of the airflow, which can cause a decrease in fan efficiency.

1.1 Project goals

This thesis aims to acquire and integrate the implementation of a code called *PyCycle*, which can be used to obtain the boundary conditions at the outlet of the aircraft engine, into a software called *CEASIOMpy*, developed by CFS Engineering, where this thesis was carried out, and Airinnova in Stockholm.
Additionally, the thesis attempts to incorporate a functionality within *CEASIOMpy* for the automatic generation of hybrid meshes, to perform RANS calculations. This is achieved through the combined utilization of three open source codes: Gmsh, Pentagrow, and TetGen.
The entire implementation in *CEASIOMpy* is subsequently validated through Computational Fluid Dynamics (CFD) simulations of different configurations.

1.2 Methodology

The main programs used during the development of this thesis are summarized here with a brief description.

- **PyCycle**: Dedicated code for thermo-fluid dynamics analysis in aerospace, used in this thesis to obtain detailed boundary conditions at the aircraft engine outlet.
- **SU2**: Open-source software for CFD simulation, designed for aerodynamic analysis and aircraft design.
- **Gmesh**: An open source mesh generator that facilitates the creation of unstructured two and three-dimensional meshes.
- **Pentagrow**: Tool used for automatic generation of pentahedral//tetrahedral hybrid meshes, used to generate the boundary layer.
- **TetGen**: A program dedicated to the automatic generation of tetrahedral meshes, used to model the outside of the boundary layer.
- **CEASIOMpy**: Software that is used to automatically run simulations starting from *cpacs* geometries and that can perform various analyses, used to generate the hybrid mesh and run the calculations.
- **NSMB**: CFS Engineering licensed in-house CFD code to validate the results obtained with the SU2 simulations, performed in CEASIOMpy environment.

2 Governing equations

2.1 Navier-Stokes equations

The Navier-Stokes equations are named after Claude-Louis Navier and George Gabriel Stokes, who independently derived these equations in the 19$^{\text{th}}$ century. Navier's work focused on the viscous stresses in fluids, while Stokes contributed to understanding the motion of viscous fluids. Their collaborative efforts resulted in a set of equations that became fundamental in fluid dynamics.
These equations describe the conservation of mass, momentum, and energy for a fluid. In their complete form, they are a set of partial differential equations that take into account the effects of viscosity, pressure, and external forces on fluid motion.

- **Conservation of mass**:
  The conservation of mass is given by the continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{2.1}$$

  where ρ is the density of the fluid, $\mathbf{v}$ is the velocity vector, and $\nabla\cdot$ is the divergence operator. This equation ensures that the mass is conserved within any given volume of fluid.

- **Conservation of momentum:**

  The conservation of momentum equation, also known as the Navier-Stokes momentum equation, can be derived by applying Newton's second law to a fluid element. The general form is:

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{g}, \tag{2.2}$$

  where p is the pressure, τ is the stress tensor, and $\mathbf{g}$ is the gravitational acceleration vector.

- **Conservation of energy:**

  The conservation of energy equation, or the total energy equation, incorporates both internal and kinetic energy. The general form is:

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho \mathbf{v} E) = -\nabla \cdot \mathbf{q} + \nabla \cdot (\tau \cdot \mathbf{v}) + \rho \mathbf{v} \cdot \mathbf{g}, \tag{2.3}$$

  where E is the total energy per unit mass, $\mathbf{q}$ is the heat flux vector, and other terms have their usual meanings.

Combining the conservation of mass, momentum, and energy equations, we obtain the complete set of Navier-Stokes equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{2.4}$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{g}, \tag{2.5}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho \mathbf{v} E) = -\nabla \cdot \mathbf{q} + \nabla \cdot (\tau \cdot \mathbf{v}) + \rho \mathbf{v} \cdot \mathbf{g}. \tag{2.6}$$

These equations govern the behavior of fluid flow and are fundamental in fluid dynamics.

Where the gradient of a scalar function f in three-dimensional Cartesian coordinates, denoted as ∇f, is a vector given by:

$$\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k} \tag{2.7}$$

The Navier–Stokes equations can be solved exactly for laminar flows. However, in the case of transitional or turbulent flows, direct analytical solutions are not feasible. In such cases, numerical approaches must be adopted.

Direct Numerical Simulation (DNS) allows solving the Navier–Stokes equations without introducing turbulence models, but it is computationally extremely expensive and therefore limited to relatively simple or small-scale problems. For more complex and realistic scenarios, turbulence models such as RANS or LES are typically employed to approximate the flow behavior.

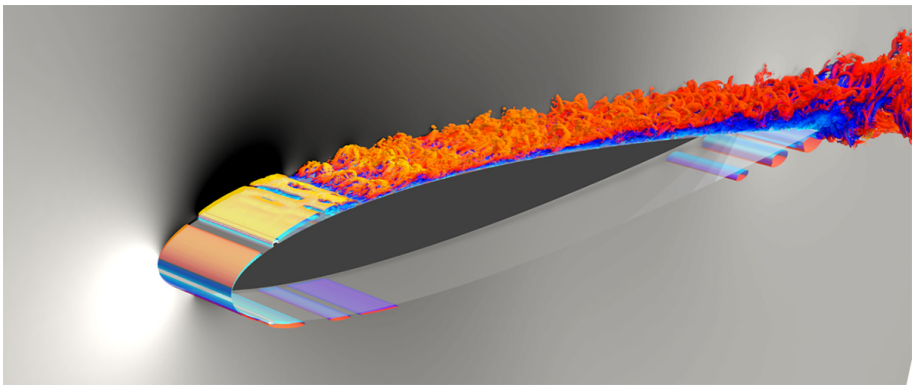


Figure 2.1: *DNS example [20]*

Despite the difficulties, understanding these equations is crucial for predicting and optimizing fluid flow in various applications, such as aircraft design.
When using CFD the equations summarized above are solved iteratively, therefore the result will be approximate, but usually accurate. By discretizing the governing

equations of fluid motion on a computational grid (mesh) and applying appropriate boundary conditions, CFD enables engineers and scientists to predict and visualize fluid behaviour.
There are some limitations due to the grid resolution that affects accuracy, the turbulence modeling that introduces uncertainties, and the boundary conditions that must be carefully defined. Therefore often, results need to be validated with the help of experiments.

2.2 Boundary Layer Theory

The study of fluid dynamics is essential to our understanding of how liquids and gases interact with solid surfaces under various conditions and systems.
One of the key concepts in fluid dynamics that underpin a wide range of scientific and engineering applications is the boundary layer theory.
The boundary layer, a thin layer of fluid adjacent to a solid surface, is a region where the flow properties of the fluid, such as velocity and temperature, change rapidly.

This theory is critical for comprehending the behavior of fluids in proximity to solid boundaries and has great relevance in fields such as aerodynamics and aeronautics.

Fluid flows near solid surfaces often exhibit a unique behavior distinct from their conduct in the bulk of the fluid. Understanding the physics and mathematics of boundary layers is essential to optimize designs, improve efficiency, and predict performance.
In this introductory chapter, we will explore its fundamental principles and mathematical formulations.

Aerodynamic forces are generated by the interactions between airflow and body. The intensity of these forces depends on various factors, including the shape and speed of the object, the mass of the fluid passing by the object, and two other crucial fluid properties: viscosity and compressibility.
Viscosity is a measure of the resistance of a fluid to deformation. It is represented by the symbol μ and is defined as the ratio of shear stress (τ) to the shear rate ($\dot{\gamma}$). Compressibility is a measure of the change in volume of a fluid in response to a change in pressure.
Aerodynamic forces are highly dependent on the viscosity of the air. As an object moves through air, the air molecules stick to the surface. This creates a layer of air near the surface (called the boundary layer), which, in effect, changes the shape of the object [3].
Both viscosity and compressibility can be associated with two dimensionless quantities, the Reynolds number and the Mach number.
The Reynolds number, denoted by Re, is used to predict the flow behavior in different fluid flow situations. It is defined as the ratio of inertial forces to viscous forces and is given by:

$$Re = \frac{\rho \cdot V \cdot L}{\mu} \tag{2.8}$$

where:

$$\begin{aligned} \rho &: \text{Density of the fluid} \\ V &: \text{Velocity of the fluid} \\ L &: \text{Characteristic length} \\ \mu &: \text{Dynamic viscosity of the fluid} \end{aligned}$$

The Mach number, denoted by Ma, represents the ratio of the speed of an object to the speed of sound in the medium. It is given by:

$$Ma = \frac{V}{a} \tag{2.9}$$

where:

$$\begin{aligned} V &: \text{Velocity of the object or fluid} \\ a &: \text{Speed of sound in the medium} \end{aligned}$$

Considering the flow of a fluid over a solid surface, the boundary layer is the thin region of flow adjacent to a surface, where the flow is retarded by the influence of friction between a solid surface and the fluid [4].
So, the molecules of the fluid increase their velocity from zero, at the surface of the body (no-slip condition), to the free stream velocity by moving away from the body, creating a velocity distribution. This distribution is influenced by viscosity; in fact, as viscosity decreases, the thickness of the boundary layer also decreases.
If we consider, for example, the flow over a flat plate, the region above the plate can then be divided into two separate parts: the region where viscous effects are prevalent, i.e. the boundary layer or inner flow, and the region where they can be neglected, i.e. outer flow, which behaves as if it were inviscid.

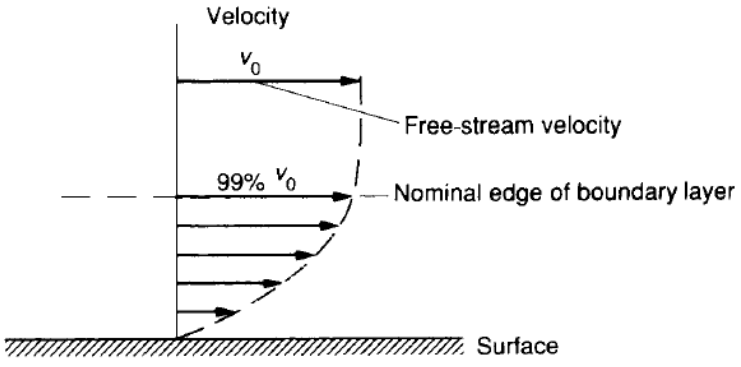


Figure 2.2: *Boundary layer flow on a flat plate [6]*

The boundary layer extends to the point where the velocity is 99 % of that of the free stream velocity and this thickness is called δ. If the thickness increases a lot, it can lead to the detachment of the boundary layer resulting in the separation of the flow from the surface.
There are two other important boundary layer parameters, which are the displacement thickness and momentum thickness:

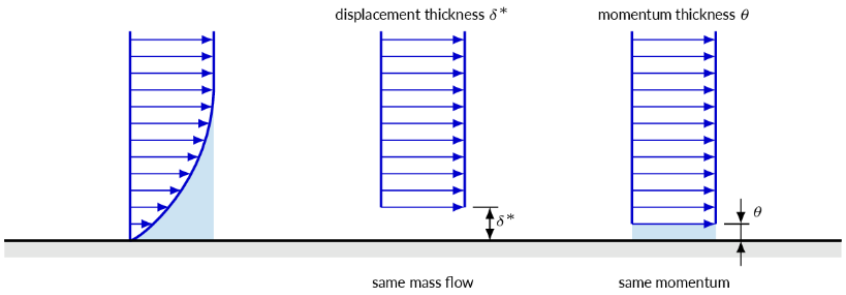


Figure 2.3: *Boundary layer thickness [5]*

- **Displacement Thickness (δ^*)**
  The displacement thickness represents the virtual increase in the thickness of the boundary layer as compared to an inviscid flow. It is defined as the distance by which the solid surface would need to be displaced to make the flow non-viscous. Mathematically, it is given by the integral of the normalized velocity profile across the boundary layer:

$$\delta^* = \int_0^\infty \left(1 - \frac{u}{U}\right) dy \tag{2.10}$$

  where u is the local velocity, U is the free stream velocity, and y is the normal distance from the wall.

- **Momentum Thickness (θ)**:
  The momentum thickness represents the reduction in momentum flux in the boundary layer compared to an inviscid flow. It is defined as the integral of the normalized momentum flux profile across the boundary layer:

$$\theta = \int_0^\infty \frac{u}{U}\left(1 - \frac{u}{U}\right) dy \tag{2.11}$$

  The momentum thickness is related to the displacement thickness by the equation:

$$\theta = \frac{2}{U}\int_0^\infty \left(1 - \frac{u}{U}\right)\frac{u}{U}\, dy \tag{2.12}$$

These parameters are crucial in analyzing boundary layer characteristics and are often used in fluid dynamics studies.

2.3 Types of boundary layers

The boundary layer can be classified as either laminar, turbulent or transient, and the Reynolds number characterizes the distinction.

- **Laminar boundary layer:**

  In a laminar boundary layer, the flow of fluid particles occurs in smooth, parallel layers. This boundary layer type is typically characterized by orderly and predictable fluid motion. The flow remains streamlined, and the fluid particles move in well-defined paths. The laminar boundary layer is more prevalent at lower flow velocities and is characterized by low Reynolds numbers.

- **Transient boundary layer:**

  A transitioning boundary layer evolves dynamically, often due to unsteady phenomena. The area where the transitioning boundary layer is present is critical for many aspects of fluid dynamics such as heat exchange and turbulence formation.

- **Turbulent Boundary Layer:**

  In contrast, a turbulent boundary layer is characterized by chaotic, irregular fluid motion, with the formation of eddies. The fluid particles exhibit random fluctuations, and the flow becomes less predictable. Turbulent boundary layers are associated with higher flow velocities and are characterized by high Reynolds numbers. Turbulence leads to increased mixing of fluid layers and higher rates of heat and momentum transfer between the fluid and the solid surface.

The critical Reynolds number for the transition from laminar to turbulent flow is not fixed and depends on the specific geometry and conditions of the flow. However, in general, when Re exceeds a certain value, the flow tends to become turbulent.
It is also crucial to note that a laminar boundary layer will produce less drag than a turbulent one, but at the same time will have a greater tendency to separate from the wing.
From an aeronautical point of view, the separation of the boundary layer causes a strong reduction in the lift produced by the wings, which can cause the aircraft to crash. In aeronautics's domain, the boundary layer exhibits a prevalent tendency to become turbulent, primarily due to the high velocities involved.
At the same time, a turbulent boundary layer has implications for the aerodynamic performance of an aircraft. Turbulence also introduces challenges such as increased skin friction drag.

Understanding the Reynolds number is essential for predicting the flow behavior in boundary layers and designing systems with optimal performance in various fluid flow conditions.

The boundary layer will usually experience the transition from laminar to turbulent, except when the Reynolds number is low. By undergoing the transition the shape of the boundary layer is going to change, as can be seen in the figure below.

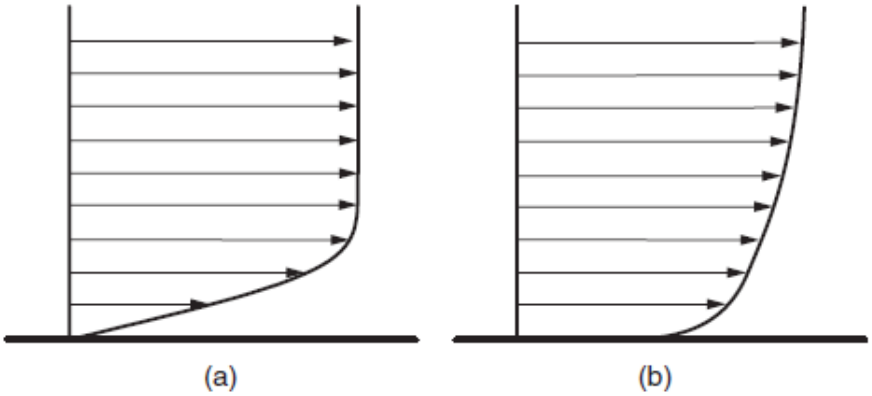


Figure 2.4: *Typical 2D boundary-layer velocity profiles. (a) Laminar. (b) Turbulent [7]*

Because of the velocity gradient, the wall generates a shear stress that can be defined as :

$$\tau = \mu \frac{du}{dy} \tag{2.13}$$

where:

τ : Shear stress at the wall

μ : Viscosity of the fluid

$\frac{du}{dy}$: Velocity gradient at the wall in the direction perpendicular to the flow

3 Boundary layer ingestion theory

As mentioned in the introduction, the main objective of boundary layer ingestion in aeronautics is to better integrate the propulsion system with the aerodynamics of the aircraft.
It is possible to summarize the BLI studies with this sentence from [8] *"The benefit of boundary layer ingestion comes from re-energizing the aircraft wake, allowing lower energy waste."*
There are two main possible geometries for an aircraft with BLI engines: Full annular BLI and asymmetric BLI.
An example of the first type is an engine located at the rear of the aircraft that receives an "injection" of boundary layer symmetrically over the entire circumference of the inlet. The asymmetrical case instead is found both in blended wing bodies, where the engine is incorporated within the fuselage, and in engines incorporated within the wings.

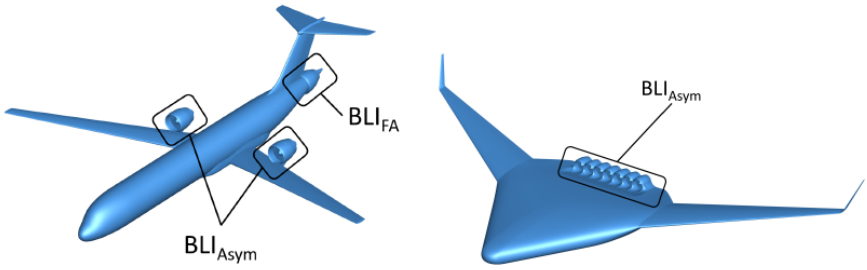


Figure 3.1: *Rear mounted BLI propulsor and BWB Concept featuring asymmetric BLI [10]*

According to [9], the three most important benefits obtained through the BLI are:

- Production of the same amount of force with a lower jet velocity, because part of the fluid entering the inlet has a lower velocity than the free stream.
- A reduction of airframe drag due to a reduction of the nacelle wetted area
- Less weight thanks to the absence of the pylon and a smaller nacelle for the asymmetric one.

On the other hand, BLI has also some disadvantages that need to be taken into account which are:

- A lower fan efficiency is caused by the distortion of the velocity and pressure profiles at the inlet of the engine.
- A lower inlet total pressure recovery.
- Greater complexity for repairs and a more complicated technology, and as a result this concept is not used yet on commercial aircraft.

It is also important to mention that the BLI engine should ideally be located in a part of the aeroplane with sufficient surface area in front of it, to maximise the thickness of the boundary layer that the motor will ingest. The thickness of the boundary layer on the wings is much less than that on the rear of the aircraft, where it can be up to 0.3 metres [13], depending on the conditions.
From this perspective, it is easy to see that configurations with the BLI at the rear of the aircraft and the BWB configuration are those with a greater improvement in overall efficiency, compared to configurations with engines embedded in the wings.
Looking at it from an aerodynamic standpoint, boundary layer ingestion (BLI) involves the propulsion system consuming a fraction of the airframe's viscous drag, potentially eliminating it, and enhancing the efficiency of the entire system. Alternatively, from a propulsion perspective, BLI utilizes the boundary layer to decrease the engine's inflow velocity, consequently minimizing ram drag and improving the overall efficiency of the system.[21]
The concept just presented can be summarised with the image below provided by Plas [8] which schematically represents the advantage of an engine with boundary layer ingestion.
In this figure, it is possible to see a conventional aircraft configuration and an extreme case of BLI where all of the aircraft wake is ingested by the engine. By re-energizing the wake a lot of the losses due to a dissipation of kinetic energy are avoided.

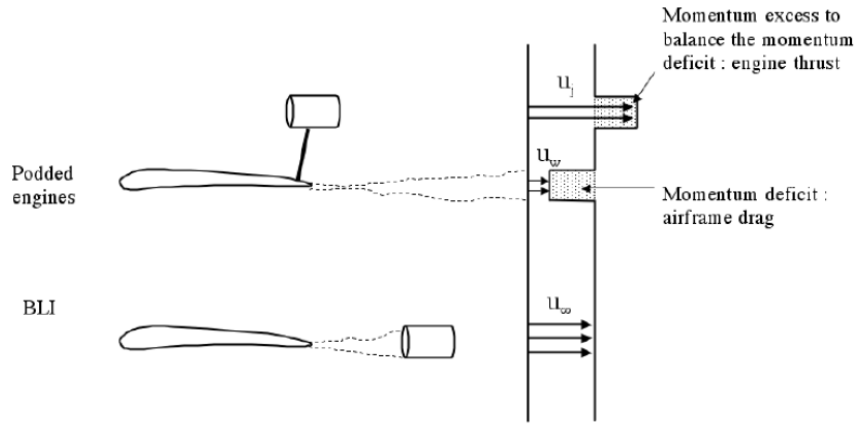


Figure 3.2: *Podded vs BLI engines schemes [8]*

In standard cases, it is important to note that not all of the aircraft wake is ingested, therefore the need to compensate for momentum drag will still be present, but the BLI engines will be more efficient depending on the configurations.

3.1 Efficiency and performance comparison

To evaluate the performance of an aircraft engine with boundary layer ingestion, several aspects must be considered. These include:

- **Thrust-to-Weight Ratio:**
  The ratio of thrust produced by the engine to the weight of the aircraft is a critical indicator of acceleration and climb capability. It is usually related to the dimensions of the engine and the purpose of the aircraft (commercial/military).
- **Thrust Specific Fuel Consumption (TSFC):** TSFC is a measure of the fuel efficiency of an engine relative to its thrust. It is defined as the amount of fuel consumed by an aircraft engine per unit of thrust produced. The lower the TSFC, the more efficient the engine.
- **Drag Reduction:**
  A potential benefit of BLI is a reduction in aircraft drag. Aircraft drag is a key factor in fuel consumption and overall efficiency.

To evaluate the impact of boundary layer ingestion on engine performance, a comparative analysis with a conventional (non-BLI) engine is essential. A detailed study should consider various operating conditions, such as different flight speeds, altitudes, and environmental factors.
Various metrics have been proposed over the years to compare BLI engines with non-BLI engines, but the one that best captures the differences is the one proposed by Smith [22] with the so-called Power Saving Coefficient (PSC).

$$PSC = \frac{Pwr'_{shaft} - Pwr_{\text{shaft}}}{Pwr'_{shaft}} \tag{3.1}$$

where:

Pwr_{shaft} : Power required to obtain a given net force for a BLI engine.
Pwr'_{shaft} : Power required to obtain the same net force for a podded engine.

So, through the use of this value, it is possible to compare the performance of two engines with very different functions. This parameter is used because it is difficult for the BLI to decouple the aerodynamic part from the propulsive part, which is commonly done for podded engines.

4 State of the art about BLI concepts

4.1 STARC-ABL

The Single-Aisle Turboelectric Aircraft with Aft Boundary Layer Propulsion (STARC-ABL), is designed to enhance overall vehicle performance by incorporating innovative aerodynamic designs and technologies.
To achieve this, an electric motor is installed at the rear of the aircraft, drawing its power from the two standard engines in the wings. The rear propeller ingests the boundary layer created by the fuselage in a sort of symmetrical configuration and accelerates it to maximise aerodynamic efficiency, decreasing fuel consumption by up to 6 %.

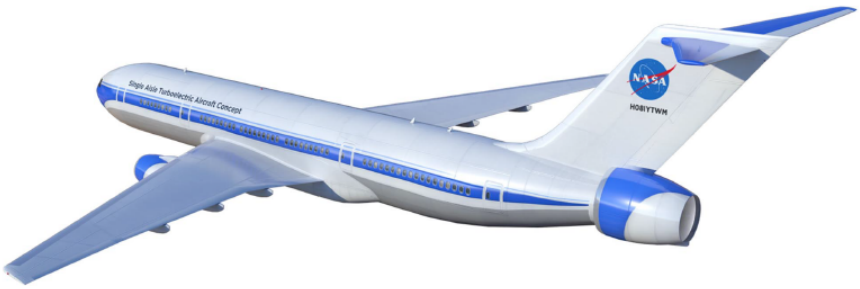


Figure 4.1: *Rendering of the STARC-ABL concept [11]*

4.2 D8 double bubble concept

This concept was introduced during the NASA N3+1 phase (2008-2010) for the conceptual design of a civil aircraft that should enter the market in the 2030-2035 time frame. The most important is the dual-lobed double-bubble fuselage and pi-tail configuration. The main goal for this configuration was to create a new concept for commercial aircraft that would substantially reduce fuel consumption, noise, and emissions.

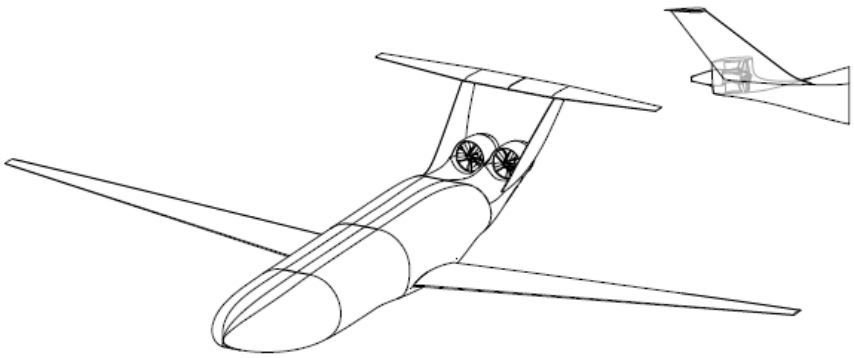


Figure 4.2: *D8 double bubble BLI model configuration [12]*

The D8 is characterized by a twin-aisle lifting body fuselage with integrated Boundary Layer Ingesting (BLI) propulsion [14]. Other studies from Greitzer [23] and Drela predicted a 7 % fuel burn reduction compared to a similar aircraft.

4.3 BWB blended wing body

Another path taken to reduce the fuel consumption of an aircraft by introducing the BLI was the study of a blended wing body. In this case, the standard shape of commercial aircraft is abandoned in favour of an entirely innovative geometry, that allows the implementation of three engines with boundary layer ingestion.

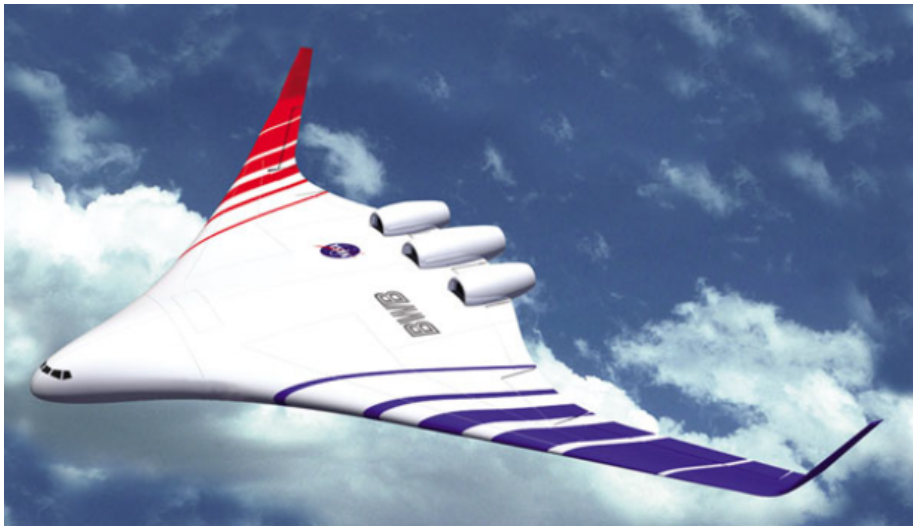


Figure 4.3: *Artist concept for a BWB*

In this case, as in the previous ones, two comparable configurations are studied, one more traditional with no BLI and the other more innovative with BLI. Based on [15], a reduction in fuel burn of around 10 % is achieved, and thus, at the same time, a reduction in pollutant emissions.

5 Mesh generation theory

A mesh is a network or grid of interconnected nodes or vertices, along with edges, lines, or surfaces connecting these nodes, forming a structure representing a geometric shape or region in space. In computational and engineering contexts, meshes are commonly used to discretize and represent complex geometries for numerical analysis, simulations, and computer graphics.

Meshes can be two-dimensional (2D) or three-dimensional (3D) and vary in complexity depending on the application. They are widely used in various fields such as engineering, physics, computer graphics, and computational science.

In summary, a mesh is a fundamental structure used to represent and analyze geometric shapes or physical domains in computational and engineering contexts. The following theory underpins the main programs currently used, with particular reference to the programs used in our discussion.

5.1 Triangulations of Point Sets

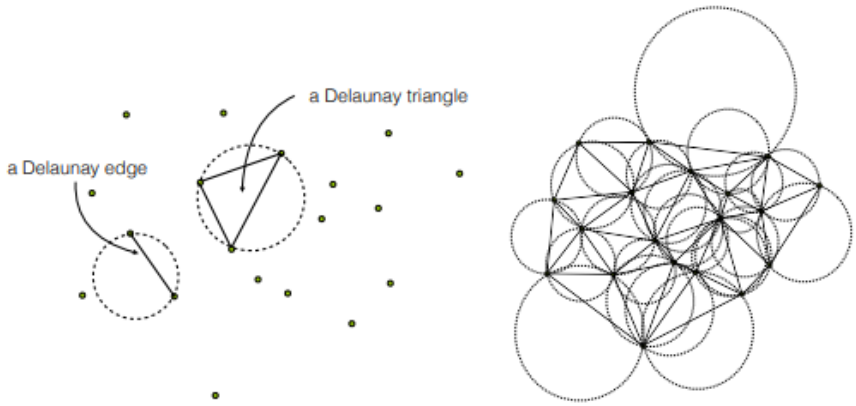


Figure 5.1: *Left: Every Delaunay simplex has an empty circumcircle. Right: The Delaunay triangulation of a 2d point set.*

Triangulations are basic geometric structures. A *triangulation* of a set V of points in $\mathbb{R}^d$ is a d-dimensional simplicial complex S whose vertex set is a subset of or equal to V. The underlying space of S is the convex hull of V. Given a point set, there are many triangulations. Among them, the Delaunay triangulation is the most interesting one. The dual of Delaunay triangulation is the Voronoi diagram of the point set. Delaunay triangulations and Voronoi diagrams have many nice mathematical properties [24, 30, 37], and they are extensively used in many applications.

5.2 Delaunay Triangulations, Voronoi Diagrams

Let V be a set of points in $\mathbb{R}^d$, σ be a k-simplex ($0 \leq k \leq d$) whose vertices are in V. A circumsphere of σ is a sphere that passes through all vertices of σ. If $k = d$, σ has a unique circumsphere, otherwise, there are infinitely many circumspheres of σ. We say that σ is Delaunay if there exists a circumsphere of σ such that no vertex of V lies inside it, see Figure 5.1.

A Delaunay triangulation D of V is a simplicial complex such that all simplices are Delaunay, and the underlying space of D is the convex hull of V [29]. Figure 5.1 right illustrates a 2d Delaunay triangulation. A 3d Delaunay triangulation is also called a Delaunay tetrahedralization.

We say that V is in general position if no $d+2$ points in V lie on a common sphere. Otherwise, we say that V is in a special position (or V is degenerate). A Delaunay triangulation of V is unique if V is in general position. Degeneracies can be removed by applying an arbitrary small perturbation onto the coordinates of points in V.

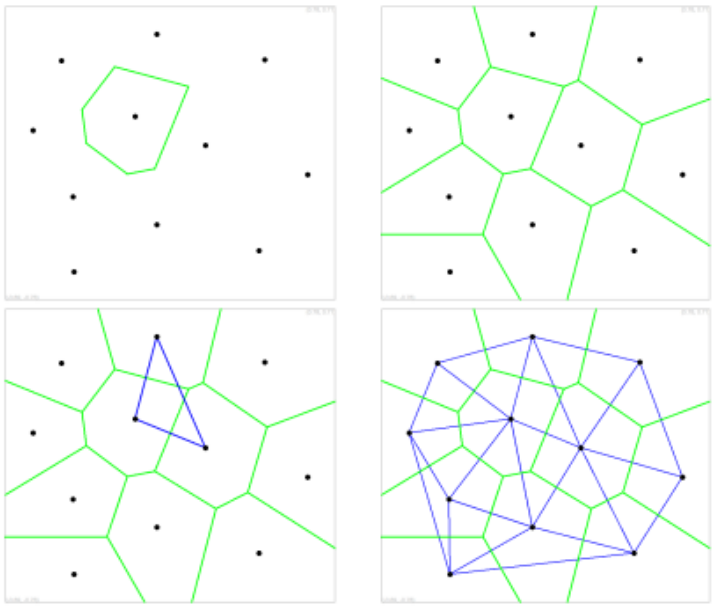


Figure 5.2: *A Voronoi region (Top-Left) and the Voronoi diagram (Top-Right) of a two-dimensional point set. Three dual edges of the Voronoi edges (Bottom-Left) and the dual diagram (Bottom-Right). If the point set is in general position, then this dual diagram is the Delaunay triangulation of this point set.*

The dual of the Delaunay triangulation is the Voronoi diagram (see Figure 5.2). For any vertex $p \in V$, the Voronoi cell of p is the set of points whose distance to p is not greater than to any other vertex of V, i.e., it is the set

$$\text{cell}(p) = \{x \in \mathbb{R}^d; \|x - p\| \leq \|x - q\|, \forall q \in V\},$$

where $\|\cdot\|$ stands for the Euclidean distance. The Voronoi diagram of V is a subdivision of $\mathbb{R}^d$ into Voronoi cells (some of which may be unbounded) and their faces

[47]. It is a d-dimensional polyhedral complex. If the point set V is in general position, there is a one-to-one correspondence between the k-simplices of the Delaunay triangulation and the $(d-k)$-polyhedra of the Voronoi diagram, where $0 \leq k \leq d$. In $\mathbb{R}^3$, the vertices of the Voronoi diagram are the circumcenters of the tetrahedra of the Delaunay tetrahedralization.

There is a nice relation between a Delaunay triangulation in $\mathbb{R}^d$ and a convex hull in $\mathbb{R}^{d+1}$. For any point $p = (p_0, p_1, \ldots, p_{d-1}) \in \mathbb{R}^d$, define its lifted point $p^+ = (p_0, p_1, \ldots, p_{d-1}, p_d) \in \mathbb{R}^{d+1}$, where $p_d = p_0^2 + \ldots + p_{d-1}^2$.

For any point set $V \subset \mathbb{R}^d$, define $V^+ = \{p^+; p \in V\} \subset \mathbb{R}^{d+1}$ be the lifted point set of V. All points in V^+ lie on a paraboloid in $\mathbb{R}^{d+1}$ (see Figure 3 left). The convex hull of V^+ is a $(d+1)$-dimensional convex polytope P. A lower face of P is a face of P which is on the downside of P (visible by points in V). The Delaunay triangulation of V is the projection of the set of

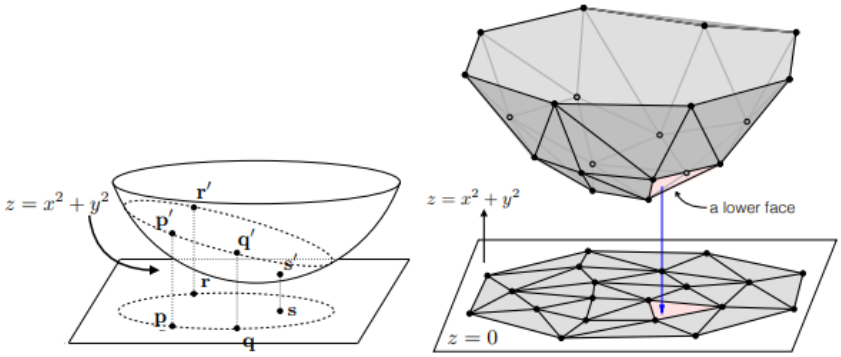


Figure 5.3: *The relation between Delaunay triangulation in $\mathbb{R}^d$ and convex hull in $\mathbb{R}^{d+1}$ (here $d = 2$). Left: Some 2d points and their corresponding 3d lifted points. Right: The Delaunay triangulation of a set of 2d points and the lower convex hull of its 3d lifted points.*

Lower faces of P onto d dimensions. Figure 5.3 right illustrates the relationship when $d = 2$. A simplex σ is a Delaunay simplex if and only if there exists a hyperplane in $\mathbb{R}^{d+1}$ passing through the lifted vertices of σ such that no other lifted vertices in V^+ lies below it. Similarly, the Voronoi diagram of V is the projection of the lower faces of a convex polytope $Q \subset \mathbb{R}^{d+1}$ such that P and Q are polar to each other [49].

5.3 Weighted Delaunay Triangulations, Power Diagrams

Weighted Delaunay triangulations are generalizations of Delaunay triangulations by replacing the Euclidean distance by "weighted distance". We associate each point $p \in \mathbb{R}^d$ a weight (a real value) $\omega_p \in \mathbb{R}$. A point with a weight is called a weighted point in $\mathbb{R}^d \times \mathbb{R}$. In particular, points in $\mathbb{R}^d$ can be considered weighted points with zero weight.

It is convenient to write the weight of a point as the square of a non-negative real, $\omega_p = \pm p^2$, where $p \geq 0$. A weighted point $p_0 = (p, \pm p^2)$ can be interpreted as a sphere centered at p with a radius p. Figure 5.4 shows a set of weighted points in

$\mathbb{R}^2$.

The weighted distance between two weighted points p_0 and z_0 is $\pi_{p_0,z_0} = \|p - z\|^2 - (p^2 + z^2)$, see Figure 5.4 left for an example.

Two weighted points p_0, z_0 are orthogonal if their weighted distance is zero, i.e., $\|p - z\|^2 = (p^2 + z^2)$.

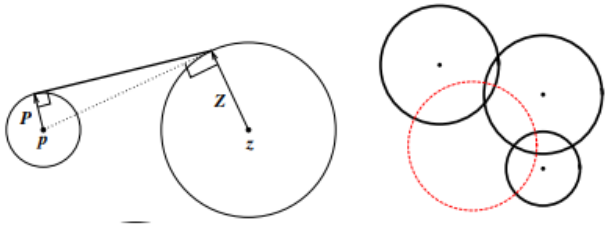


Figure 5.4: *Left: The weighted distance (left) of two weighted points (p, p^2) and (z, z^2). Right: The orthosphere of three weighted points. (Figures are taken from Damrong Guoy's PhD thesis.)*

We say that two weighted points are farther than orthogonal when their weighted distance is positive, and closer than orthogonal when the distance becomes an imaginary number.

In general, $d + 1$ points in $\mathbb{R}^d$ define a unique circumsphere passing through them. Similarly, $d + 1$ weighted points in $\mathbb{R}^d$ define a unique common orthosphere. When all points have zero weights, their orthosphere is just their circumsphere.

5.4 (right) gives an example of the orthosphere of three weighted points in two dimensions.

Let $V' \subset \mathbb{R}^d \times \mathbb{R}$ be a finite set of weighted points. We say a sphere is empty if all weighted points in V' are farther than orthogonal of it. The weighted Delaunay triangulation of V' is a simplicial complex D' such that every simplex has an orthosphere which is empty, and the underlying space of D' is the convex hull of V'. Obviously, if all the points have the same weight, the weighted Delaunay triangulation is the same as the usual Delaunay triangulation. Note that a weighted Delaunay triangulation does not necessarily contain all points in V'.

The dual of a weighted Delaunay triangulation is a weighted Voronoi diagram, also called the power diagram [24, 32] of the weighted point set V_0. Power diagrams can be similarly defined as the Voronoi diagram by using the weighted distance instead of the Euclidean distance. If no $d+2$ weighted points of V_0 share a common orthosphere, i.e., it is in general position, then the simplices of the weighted Delaunay triangulation and the cells of the power diagram have a one-to-one correspondence. In $\mathbb{R}^3$, the vertices of the power diagram are the orthocenters of the tetrahedra of the weighted Delaunay tetrahedralization.

A weighted Delaunay triangulation of $V \subset \mathbb{R}^d$ is also the projection of the set of lower faces of a convex polytope $P \subset \mathbb{R}^{d+1}$. Any point $p = \{p_0, \ldots, p_{d-1}\} \in V$ is lifted to a point $p_0 = \{p_0, \ldots, p_{d-1}, p_d\} \in \mathbb{R}^{d+1}$, where $p_d = p_0^2 + \ldots + p_{d-1}^2 - p^2$ (where p is the weight of p). For $p \neq 0$, p_0 does not lie on a paraboloid in $\mathbb{R}^{d+1}$, but

is moved vertically downward by p^2. A simplex belongs to the weighted Delaunay triangulation of V (i.e., it has an empty orthosphere) if and only if there exists a hyperplane passing through the lifted weighted points of this simplex and no lifted weighted point of V lies below the hyperplane.

Both weighted Delaunay triangulations and power diagrams are called regular subdivisions of point sets [49]. Regular subdivisions have nice combinatorial structures. They are one of the important objects studied in higher-dimensional convex polytopes [49, 28].

5.4 Algorithms

Algorithms for generating Delaunay (and weighted Delaunay) tetrahedralizations are well studied in computational geometry [30]. The TetGen software used in this project implemented two algorithms, the Bowyer-Watson algorithm [26, 48], and the incremental flip algorithm [32]. Both algorithms are incremental, i.e., they insert points one after another. Both have the worst-case runtime $O(n^2)$. In most practical applications, they are usually very efficient. For example, if points are uniformly distributed, the expected run time is $O(n \log n)$.

The speed of incremental algorithms is very much affected by the cost of point location. TetGen uses a spatial sorting scheme [25] to improve the point location. The idea is to sort the points such that nearby points in space have nearby indices. The points are first randomly sorted into different groups, and then points in each group are sorted along the Hilbert curve. Inserting points in this order, each point location can be done in nearly constant time.

TetGen uses Shewchuk's exact geometric predicates [38] for performing the Orient3D, InSphere, and Orient4D tests. These suffice to guarantee the numerical robustness of generating Delaunay and weighted Delaunay tetrahedralizations. TetGen uses a simplified symbolic perturbation scheme [31] to remove the degeneracies.

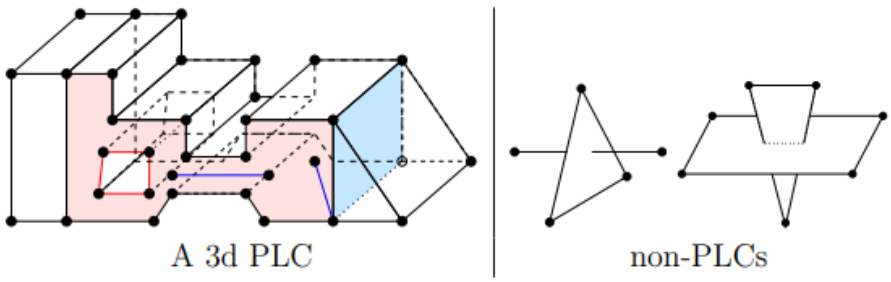


Figure 5.5: *Left: A 3d piecewise linear complex. The left-shaded area shows a facet, which is non-convex and has a hole in it. It has also edges and vertices floating in it. The right-shaded area shows an interior facet separating two sub-domains. Right: Configurations which are not PLCs.)*

5.5 Tetrahedral Meshes of 3d Spaces

A tetrahedral mesh is a 3d simplicial complex that is a discrete representation of a 3d continuous space (domain), both in its topology and geometry. Note that a Delaunay tetrahedralization is a tetrahedral mesh of the convex hull of its vertex set. In general, a geometric domain may not be convex and may have arbitrarily complex boundaries. The input domain is modelled by a piecewise linear complex (Section 1.2.1). The focuses are the representation of the geometry (the boundary) and the quality of the mesh. Programs such as TetGen or Gmsh generate several types of tetrahedral meshes to achieve these goals. They are explained in the following subsections.

5.6 Piecewise Linear Complexes (PLCs)

At first, we need a model to represent a 3D domain. A 3D piecewise linear complex (PLC) [11] X is a set of cells that satisfies the following properties:

1. The boundary of each cell in X is a union of cells in X.
2. If two distinct cells $f, g \in X$ intersect, their intersection is a union of cells in X.

Figure 5 left shows an example.

The boundary of a 3D PLC is the set of cells whose dimensions are less than or equal to 2. A 0-dimensional cell is a vertex. In particular, we call a 1-dimensional cell (an edge) a segment, and a 2-dimensional cell a facet. Each facet of a PLC is a 2D PLC. It may contain holes, segments, and vertices in its interior, see Figure 5 left for an example.

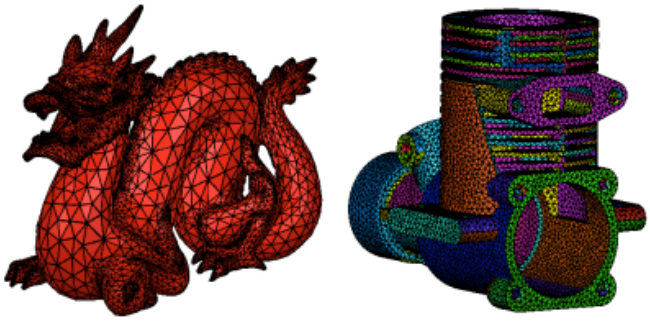


Figure 5.6: *Examples of surface meshes of PLCs.*

PLCs are flexible in describing 3D geometric features. For instance, they permit facets, segments, and vertices to float in a domain, or segments and vertices to float in the facet. One purpose of these floating cells is to constrain how the PLC can be meshed therefore that boundary conditions may be applied to those cells.

The definition of a PLC disallows illegal intersections of its cells, see

5.6 right for examples. Two segments can only intersect at a common vertex that is also in X. Two facets of X may intersect only at a union of vertices and segments which are also in X.

The underlying space of a PLC X, denoted $|X|$, is $\bigcup_{f \in X} f$, which is the domain to be triangulated. A tetrahedral mesh of X is a 3D simplicial complex T such that:

1. X and T have the same vertices,
2. every cell in X is a union of simplices in T, and
3. $|T| = |X|$.

Let T be a tetrahedral mesh of a 3D PLC X. The boundary of X is respected by the elements of T, i.e., each segment of X is represented by a union of edges in T, and each facet of X is represented a union of triangles in T. To distinguish those edges and triangles of T which are on segments and facets of X, we call them boundary edges and boundary faces.

TetGen uses a simple boundary representation (a surface mesh) to represent a 3D PLC. It is explained in Section 5.1.1 and in the file formats .poly and .smesh of TetGen. Figure 6 shows two typical surface meshes of 3D PLCs. The following points are useful to know:

- TetGen does not generate the input surface mesh of the PLC.
- TetGen can modify the surface mesh by further subdividing them. It is necessary to conform to the constrained Delaunay property and improve the mesh quality. This is the default choice.

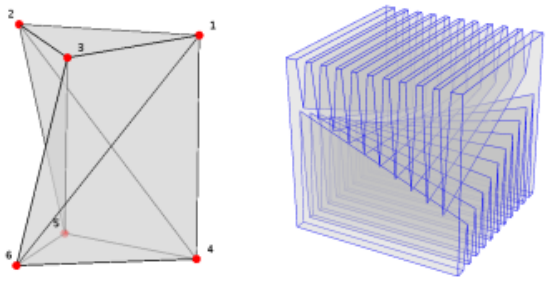


Figure 5.7: *Polyhedra which can not be tetrahedralized without Steiner points. Left: The Schonhardt polyhedron [36]. Right: The Chazelle's polyhedron [27].*

- can preserve the surface mesh (does not subdivide it) when the switch -Y is applied.
- If the input surface mesh contains self-intersections, TetGen will detect them and stop the meshing process automatically.
- If the input surface mesh does not enclose the 3d volume, i.e., it is not watertight, TetGen will finish the meshing process, but it returns an empty 3d tetrahedral mesh unless the -c switch (to keep the convex hull of the mesh) is used.

Limitation of PLCs. A PLC only gives a piecewise linear approximation of a 3d domain. It does not take the curvature of the surfaces into account. When TetGen modifies the surface mesh, it only modifies the linear edges and facets. This is unfortunately a limitation of using PLC.

5.7 Steiner Points

There are 3D polyhedra which may not be triangulated with only their vertices. Two typical examples are shown in Figure 5.7. Such polyhedra can only be triangulated if additional vertices, called Steiner points, are added to them.

A Steiner tetrahedralization of a PLC X is a tetrahedralization of $X \cup S$, where S is a finite set of Steiner points (not vertices of the PLC X). TetGen generates Steiner tetrahedralizations of PLCs. Two types of Steiner points are used in TetGen:

- The first type of Steiner points is used in creating an initial tetrahedralization of PLC. These Steiner points are mandatory to create a valid tetrahedralization.
- The second type of Steiner points is used in creating quality tetrahedral meshes of PLCs. These Steiner points are optional, but they may be necessary to improve the mesh quality or to conform to the size of mesh elements.

In both cases, TetGen tries to generate the Steiner points efficiently and uses a small amount of Steiner points. The optimal locations and the optimal amount of Steiner points is still a topic of research.

5.8 Boundary Conformity

A fundamental problem in mesh generation is how to generate a mesh that contains a set of input constraints (edges and triangles). These constraints usually represent special requirements, such as the boundary complex of a PLC. It is generally referred to as the boundary conformity or boundary recovery problem. Boundary conformity in 2d is very easy. One can enforce any edge (which does not intersect any boundary) into a triangulation. Moreover, it does not need any Steiner point. However, it isn't easy in 3d since it is not always possible to enforce an edge or a triangle into a 3d triangulation without using Steiner points. Different types of (Steinerte) trahedralizations can be generated such that input segments and facets of a PLC are respected.

- A conforming Delaunay tetrahedralization. It is a subcomplex of a Delaunay tetrahedralization, i.e., every tetrahedron is a Delaunay tetrahedron. It may contain Steiner points. Some Steiner points may lie on the PLC boundary, i.e., a boundary segment or facet of the PLC may be represented by a union of edges and triangles of the tetrahedralization.
- A constrained Delaunay tetrahedralization (CDT). Each of its tetrahedra satisfies a constrained Delaunay criterion. A CDT has many properties similar to

those of a Delaunay tetrahedralization. It is further explained in Section 1.2.4. A CDT may contain Steiner points. Moreover, most of the Steiner points lie on the segments of the PLC.

- A constrained tetrahedralization. It is a tetrahedralization that preserves the input surface mesh of the PLC. It may contain Steiner points, which may lie on the boundary or in the PLC's interior. This type of tetrahedralization may be neither Delaunay nor constrained Delaunay.

These different types of tetrahedralizations produced may find use in different situations. For instance, conforming DTs are desired for

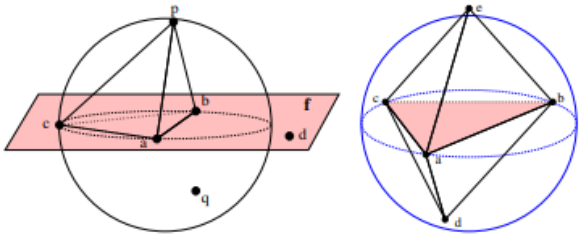


Figure 5.8: *Left: The tetrahedron abcd is constrained Delaunay. Right: The triangle abc is locally Delaunay.*

applications that need the Delaunay property. While a conforming DT might need a large amount of Steiner points. CDTs require fewer Steiner points than conforming DTs. Constrained tetrahedralizations are useful in many engineering applications for which the input domain boundaries need to be preserved.

5.8.1 Constrained Delaunay Tetrahedralizations

A constrained Delaunay tetrahedralization (CDT) is a variation of a Delaunay tetrahedralization that is constrained to respect the edges and facets of X. CDTs in the plane were introduced by Lee and Lin [33]. Shewchuk [42, 39] generalized them into three or higher dimensions.

In the following, we give two equivalent definitions of constrained Delaunay tetrahedralizations.

The visibility between two vertices $p, q \in |X|$ is called occluded if there is a facet $f \in X$ such that p and q lie on opposite sides of the plane that includes f, and the line segment pq intersects this facet (see Figure 5.8). A tetrahedron t whose vertices are in X is constrained Delaunay if its circumsphere encloses no vertex of X, which is visible from any point in the relative interior of t (see Figure 5.8 Left).

A tetrahedralization T is a constrained Delaunay tetrahedralization of X if it is a tetrahedralization of X and every tetrahedron of T is constrained Delaunay.

Let s be a triangle in a tetrahedralization T of X. s is said to be locally Delaunay if either it belongs to only one tetrahedron of T, or it is a face of exactly two tetrahedra t_1 and t_2 and it has a circumsphere which does not enclose any vertex of t_1 and t_2.

Equivalently, the circumsphere of t_1 encloses no vertex of t_2 and vice versa (see Figure 5.8 Right). A tetrahedralization T of P is a CDT of P if every triangle in T not included in any facet of P is locally Delaunay.

The definitions of Delaunay tetrahedralization and constrained Delaunay

Tetrahedralizations are almost the same except that, for the CDT, we free the requirement of being locally Delaunay for triangles in the facet. Hence CDTs retain many properties of those of Delaunay tetrahedralizations, see [42, 44]. Note that simplices (tetrahedra, triangles, and edges) in a CDT are not always Delaunay.

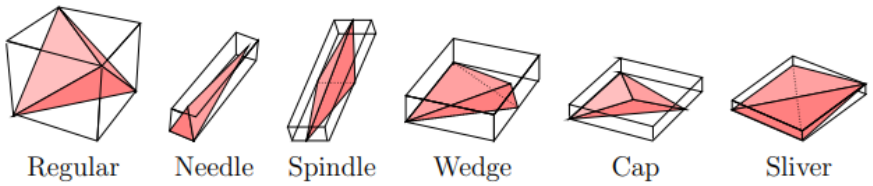


Figure 5.9: *Tetrahedra of different shapes.*

A CDT of an arbitrary PLC X may not exist [42]. Steiner points are necessary to ensure the existence of a CDT. A Steiner CDT of X is a CDT of $X \cup S$, where $S \subset |X|$ is a set of Steiner points.

Compared to conforming Delaunay tetrahedralizations, (Steiner) CDTs usually require much fewer Steiner points.

6 Meshing programs

6.1 Gmsh

Gmsh is a three-dimensional 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 flexible visualization capabilities.

Gmsh is built around four modules (geometry, mesh, solver and post-processing), which can be controlled with the graphical user interface (GUI; see Gmsh graphical user interface), from the command line (see Gmsh command-line interface), using text files written in Gmsh's own scripting language (.geo files; see Gmsh scripting language), or through the C++, C, Python, Julia and Fortran application programming interface (API; see Gmsh application programming interface). A brief description of the four modules is given hereafter, before an overview of what Gmsh does best, and some practical information on how to install and run Gmsh on your computer.

- Geometry module.
- Mesh module.
- Solver module.
- Post-processing module

6.1.1 Mesh Module

A finite element mesh of a model is a tessellation of its geometry by simple geometrical elements of various shapes (in Gmsh: lines, triangles, quadrangles, tetrahedra, prisms, hexahedra and pyramids), arranged in such a way that if two of them intersect, they do therefore along a face, an edge or a node, and never otherwise. This defines a so-called conformal mesh. The mesh module implements several algorithms to generate such meshes automatically. By default, meshes produced by Gmsh are considered unstructured, even if they were generated in a structured way (e.g., by extrusion). This implies that the mesh elements are completely defined simply by an ordered list of their nodes and that no predefined ordering relation is assumed between any two elements.

In order to guarantee the conformity of the mesh, mesh generation is performed in a bottom-up flow: curves are discretized first; the mesh of the curves is then used to mesh the surfaces; then the mesh of the surfaces is used to mesh the volumes. In this process, the mesh of an entity is only constrained by the mesh of its boundary, unless entities of lower dimensions are explicitly embedded in entities of higher dimension. For example, in three dimensions, the triangles discretizing a surface will be forced to be faces of tetrahedra in the final 3D mesh only if the surface is part of the boundary of a volume, or if that surface has been explicitly embedded in the volume. This automatically ensures the conformity of the mesh when, for example, two volumes

share a common surface. Mesh elements are oriented according to the geometrical orientation of the underlying entity. Every meshing step is constrained by a mesh size field, which prescribes the desired size of the elements in the mesh. This size field can be uniform, specified by values associated with points in the geometry, or defined by general mesh size fields (for example related to the distance to some boundary, to an arbitrary scalar field defined on another mesh, etc.): see Gmsh mesh size fields. For each meshing step, all structured mesh directives are executed first and serve as additional constraints for the unstructured parts. (The generation and handling of conformal meshes has important consequences on how meshes are stored internally in Gmsh, and how they are accessed through the API: see Gmsh application programming interface.)

Gmsh's mesh module regroups several 1D, 2D and 3D meshing algorithms:

The 2D unstructured algorithms generate triangles and/or quadrangles (when recombination commands or options are used). The 3D unstructured algorithms generate tetrahedra, or tetrahedra and pyramids (when the boundary mesh contains quadrangles).
The 2D structured algorithms (transfinite and extrusion) generate triangles by default, but quadrangles can be obtained by using the recombination commands or options. The 3D structured algorithms generate tetrahedra, hexahedra, prisms, and pyramids, depending on the type of surface meshes they are based on.

Gmsh provides a choice between several 2D and 3D unstructured algorithms. Each algorithm has its advantages and disadvantages.

For all 2D unstructured algorithms, a Delaunay mesh that contains all the points of the 1D mesh is initially constructed using a divide-and-conquer algorithm3. Missing edges are recovered using edge swaps3. After this initial step, several algorithms can be applied to generate the final mesh:

The "MeshAdapt" algorithm4 is based on local mesh modifications. This technique makes use of edge swaps, splits, and collapses: long edges are split, short edges are collapsed, and edges are swapped if a better geometrical configuration is obtained. The "Delaunay" algorithm is inspired by the work of the GAMMA team at INRIA5. New points are inserted sequentially at the circumcenter of the element that has the largest adimensional circumradius. The mesh is then reconnected using an anisotropic Delaunay criterion. The "Frontal-Delaunay" algorithm is inspired by the work of S. Rebay6. Other experimental algorithms with specific features are also available. In particular, "Frontal-Delaunay for Quads"7 is a variant of the "Frontal-Delaunay" algorithm aiming at generating right-angle triangles suitable for recombination; and "BAMG"8 allows to generate anisotropic triangulations. For very complex curved surfaces the "MeshAdapt" algorithm is the most robust. When high element quality is important, the "Frontal-Delaunay" algorithm should be tried. For very large meshes of plane surfaces the "Delaunay" algorithm is the fastest; it usually also handles complex mesh size fields better than the "Frontal-Delaunay". When the "Delaunay" or "Frontal-Delaunay" algorithms fail, "MeshAdapt" is automatically triggered. The "Automatic" algorithm uses "Delaunay" for plane surfaces

and “MeshAdapt” for all other surfaces.

Several 3D unstructured algorithms are also available:

The “Delaunay” algorithm is split into three separate steps. First, an initial mesh of the union of all the volumes in the model is performed, without inserting points in the volume. The surface mesh is then recovered using H. Si’s boundary recovery algorithm Tetgen/BR. Then a three-dimensional version of the 2D Delaunay algorithm described above is applied to insert points in the volume to respect the mesh size constraints. The “Frontal” algorithm uses J. Schoeberl’s Netgen algorithm 9. The “HXT” algorithm10 is a new efficient and parallel implementation of the Delaunay algorithm. Other experimental algorithms with specific features are also available. In particular, “MMG3D”11 allows to generate of anisotropic tetrahedral cations. The “Delaunay” algorithm is currently the most robust and is the only one that supports the automatic generation of hybrid meshes with pyramids. Embedded model entities and general mesh size fields (see Specifying mesh element sizes) are currently only supported by the “Delaunay” and “HXT” algorithms.

When Gmsh is configured with OpenMP support (see Compiling the source code), most of the meshing steps can be performed in parallel:

1D and 2D meshing is parallelized using a coarse-grained approach, i.e. curves (resp. surfaces) are each meshed sequentially, but several curves (resp. surfaces) can be meshed at the same time. 3D meshing using HXT is parallelized using a fine-grained approach, i.e. the actual meshing procedure for a single volume is done in parallel. The number of threads can be controlled with the -nt flag on the command line (see Gmsh command-line interface), or with the General.NumThreads, Mesh.MaxNumThreads1D, Mesh.MaxNumThreads2D and Mesh.MaxNumThreads3D options (see General options and Mesh options).

6.2 TetGen

TetGen is a tetrahedral mesh generator. It creates 3d triangulations of polyhedral domains. It generates meshes with well-shaped elements whose sizes are adapted to the geometric features or user-provided sizing functions. It has applications in various applications in scientific computing, such as computer graphics (CG), computer-aided design (CAD), geometry processing (parametrizations and computer animation), and physical simulations (finite element analysis). For a set of 3d (weighted) points, TetGen generates the Delaunay and weighted Delaunay tetrahedral cations as well as their duals, the Voronoi diagram, and the power diagram. For a 3d polyhedral domain, TetGen generates the constrained Delaunay tetrahedral cation and an isotropic adaptive tetrahedral mesh of it. Domain boundaries (edges and faces) are respected and can be preserved in the resulting mesh. The shapes of resulting tetrahedra can be provably good for a large class of inputs. One of its main applications is to simulate physical phenomena by numerical methods, such as finite element and finite volume methods. A good quality mesh is essential to achieve high accuracy and efficiency of the simulations. The algorithms of TetGen are Delaunay-based. They can preserve arbitrary complex geometry and topology. TetGen uses a constrained Delaunay refinement algorithm which guarantees termination and good mesh quality. The robustness of TetGen is enhanced by using advanced technologies developed in computational geometry. TetGen is an outcome of a long-term research project supported by the Weierstrass Institute for Applied Analysis and Stochastics (WIAS). It is continuously developed and improved. In some CFD workflows, tools such as SUMO are used to generate surface meshes, which are then processed by volume mesh generators such as TetGen. However, in this work, the mesh generation process is based on Gmsh, TetGen, and Pentagrow.

6.2.1 Algorithms

Shewchuk first considered algorithms for constructing 3d CDTs. In [39], a sufficient condition for the existence of a CDT of a PLC (or a polyhedron) is given. Based on this condition, several algorithms for constructing Steiner CDTs are proposed [38, 40, 43, 45]. TetGen's CDT algorithm is from Si and Gartner [45, 43]. The basic algorithm for generating quality tetrahedral meshes is the Delaunay refinement algorithm from Ruppert [35] and Shewchuk [38]. This algorithm generates a quality mesh of Delaunay tetrahedra with no tetrahedra having a radius-edge ratio greater than 2.0 (equivalently, no face angle less than 14.5grad). The sizes of tetrahedra are graded from small to large over a short distance. TetGen implemented this algorithm for improving the mesh quality of a CDT of a PLC. In practice, the algorithm generally surpasses- in the theoretical bounds and eliminating tetrahedra with small or large dihedral angles efficiently. There are two theoretical problems with the basic Delaunay refinement algorithm. First, it does not remove slivers due to using a radius-edge ratio as the sole tetrahedral shape measure. Second, it may not terminate if the input PLC X contains sharp features, i.e., there are two edges of X meeting at an acute angle, or two facets of X meeting at an acute dihedral angle. TetGen uses the minimal dihedral angle of a tetrahedron as a second shape measure. The Delaunay refinement algorithm can find and remove slivers. Since TetGen works

with CDTs, it can detect all the sharp features in the CDT in advance. Tetrahedra at the sharp features are never removed. The modified algorithm in TetGen always terminates. However, some badly shaped tetrahedra near the sharp features may survive. If a user-defined mesh sizing function is given, TetGen will generate an adaptive tetrahedral mesh according to the input mesh sizing function. It uses a constrained Delaunay refinement algorithm [43].

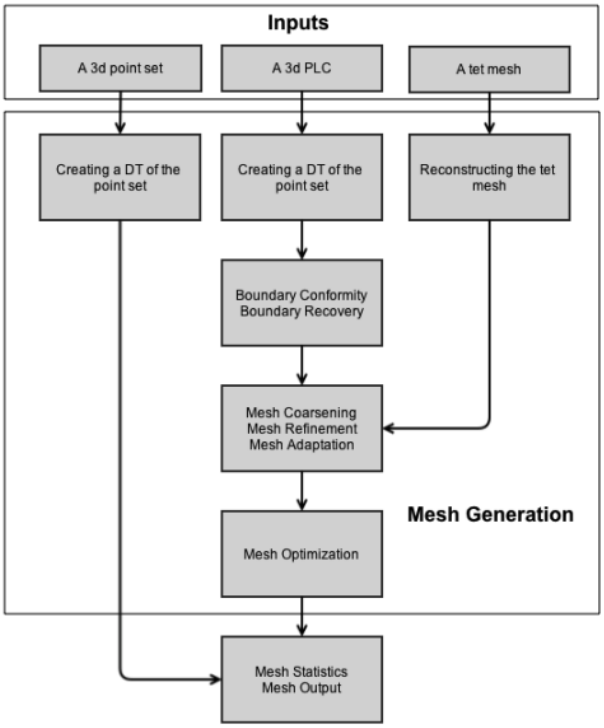


Figure 6.1: *The flowchart of the mesh generation process of TetGen.*

6.2.2 Description of the Meshing Process

Figure 6.1 shows a graphic flowchart of the meshing process in TetGen. Here are the general steps of TetGen to create a (quality) tetrahedral mesh. Many of these steps can be skipped, depending on the command line switches.

1. Initialize constants and parse the command line.
2. Read the vertices from a (.node) file and either
   - create the corresponding Delaunay tetrahedralization (DT) (no -r), or
   - read an existing tetrahedral mesh from (.ele, .face, .edge) files and reconstruct it (-r).
3. Read the boundary information (segments and facets) from (.poly or .smesh, .edge) files and triangulate them (-p).
4. Read the background mesh from (.b.node, .b.ele, .b.mtr ...) files (if it is provided) and interpolate the mesh element size from the background mesh to the current mesh (-m).

5. Insert the boundary segments and facets into the DT (-p) by either
   - constructing a constrained Delaunay tetrahedralization (CDT) in which the segments and facets may be split into smaller pieces (no -Y), or
   - recovering the segments and facets in a tetrahedralization which contains them (-Y).
6. Read the holes (-p), regional attributes (-pA), and regional volume constraints (-pa), and either
7. Insert the boundary segments and facets into the DT (-p) by either
   - remove the exterior tetrahedra (in the holes and concavities) (no -c), or
   - mark the exterior tetrahedra (-c), and spread the regional attributes and volume constraints.
8. Coarsen the mesh (-R) by removing vertices which are either marked or inconsistent according to a mesh sizing function (-m).
9. Read the list of additional vertices from a (.a.node) file (if it is provided) and insert them into the current mesh (-i)
10. Read the background mesh from (.b.node, .b.ele, .b.mtr ...) files (if it is provided) and interpolate the mesh element size from the background mesh to the current mesh (-m).
11. Enforce the constraints on minimum quality bound (-q) and maximum, volume (-a), and mesh sizing function (-m).
12. Optimize the mesh for the optimization scheme (-O) based on specified quality measures (-o).
13. Write the output files and print the statistics.
14. Check the consistency of the mesh (-C).

6.2.3 Pentagrow

Pentagrow is a command-line interface to the hybrid mesh generation features in libsurf. This interface allows the generation of hybrid pentahedral/tetrahedral meshes around existing surface meshes. The program is controlled by employing a configuration file where some execution options can be set. The final mesh can be viewed (and converted to other formats) using the program wfscope.

Internally, Pentagrow makes use of the same functions used by sumo when starting a hybrid mesh generation pass on a surface mesh in the graphical user interface. The difference is that Pentagrow can be used to generate a hybrid volume mesh around a surface mesh which does not originate from sumo.

- **Generate envelope**: As a first step, the envelope surface of the prismatic layer is determined from the wall mesh and some of the parameters given in

the configuration file. This step is usually completed in a few seconds.

- **Tetrahedral region**: Envelope and far-field boundaries are passed to the external program Tetgen, which generates the tetrahedral mesh between these two surfaces and removes elements from any marked cavities. In most cases, this step is the most time-consuming and may take as long as 20 minutes for envelope meshes of a million triangles. Note that the combination of a surface mesh of extremely low quality and very high tet quality requirements can result in excessively long refinement times because Tetgen is forced to repeatedly split envelope surface triangles to achieve the desired tet shape quality measure.
- **Prismatic layer**: Once the tetrahedral region has been filled with elements, the envelope surface and wall mesh are adapted to the inner boundary of the tet mesh and pentahedra are created starting from the wall. Although this step usually results in most of the mesh nodes, it tends to be completed rather more quickly than the more complicated Tetgen stage.

7 CEASIOMpy

CEASIOMpy is a Conceptual Aircraft design tool that is freely available to the public as open-source software downloadable from GitHub. It has been developed by CFS Engineering in collaboration with Airinnova, to provide a comprehensive and user-friendly platform for the design of new aircraft.
CEASIOMpy was initially developed within the frame of the SimSAC European project from 2006 to 2009 (at that time called CEASIOM and written in MatLab) and later improved as a part of the European project AGILE from 2015 to 2018 [19]. The project aimed to reduce the time-to-market and development costs of new aircraft, which are two critical objectives for the aeronautical industry.
One of the key features of CEASIOMpy is its use of Python as a programming language. Python is becoming one of the most popular languages in the scientific and engineering communities due to its simplicity and versatility. With its modular design, CEASIOMpy allows users to easily customize and extend the functionality of the software to meet their specific needs.

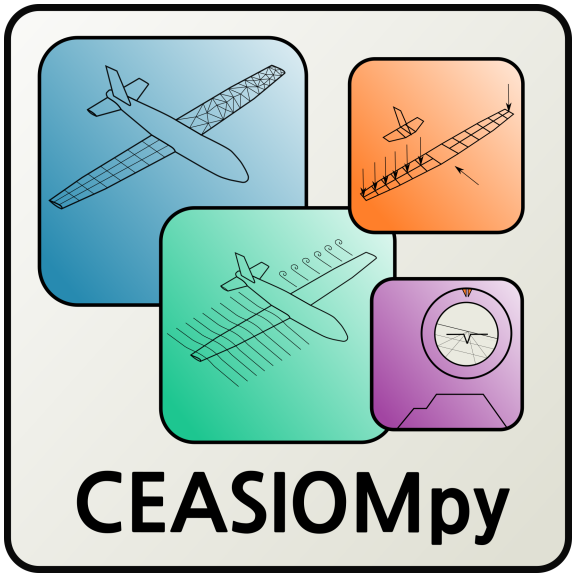


Figure 7.1: *CEASIOMpy logo*

CEASIOMpy provides a wide range of tools for the design and optimization of various aircraft components such as wings, fuselage, and propulsion systems. Various methods are included for aerodynamic and structural analysis, optimization, and sizing. The software also includes a graphical user interface (GUI) that allows users to interact with the software easily and efficiently.
Overall, CEASIOMpy represents a significant step forward in the field of aircraft design and optimization. Its open-source and collaborative nature, combined with its powerful features and user-friendly interface, make it a valuable resource for aeronautical engineers and researchers alike.

7.1 Modules

Over the years, a variety of tools have been implemented in CEASIOMpy to design and optimize workflows. These tools include general modules, as well as more specific ones for different components of the aircraft design process. The main modules in CEASIOMpy are:

- **General modules**:
  These modules provide the core functionality of CEASIOMpy, including the ability to import and export data, manage projects and configure settings. Here, the "ThermoData" module has been added to obtain the engine boundary conditions.

- **Geometry and Mesh module**:
  This module contains the "CPACSCreator" tool, which is a CAD tool used to create and modify CPACS files. CPACS, or Common Parametric Aircraft Configuration Scheme [18], is a data definition model for air transport systems that is based on the TiGL language, developed by the German Aerospace Center (DLR) and the European aircraft manufacturer Airbus. The Geometry and Mesh modules allow users to generate a 3D model of an aircraft from a CPACS file, which can be used for further analysis and optimization. The important feature of the CPACS files is the possibility for engineers to exchange information between their tools. Inside this module are also present the "CPACS2GMSH", "CPACS2SUMO" and "SUMOAutoMesh" for the automatic unstructured mesh generation starting from the geometry defined in the CPACS file.

- **Aerodynamics module**:
  This module provides a set of tools for aerodynamic analysis, including computational fluid dynamics (CFD) simulations. The CFD tool can be used to simulate the airflow around an aircraft, providing valuable data for the design and optimization of the aircraft's shape and components. Is also available a vortex lattice calculation tool called "PyTornado" for low-fidelity aerodynamic analysis of wings. The calculations are Euler and are performed with the open-source code SU2 for CFD analysis.

- **Weight and Balance**:
  This module provides tools to estimate the weight and balance of an aircraft. This is a critical step in the design process, as it affects the performance and safety of the aircraft.

- **Mission Analysis**:
  This module is under development, it allows users to simulate the flight of an aircraft, taking into account factors such as fuel consumption, altitude, and weather conditions. This information can be used to optimize the aircraft's design for a specific mission or route.

- **Structure**:
  This module is not developed yet, but in the future, it will provide structural analysis tools that will allow users to evaluate the strength and durability of

the aircraft's components. This is essential to ensure that the aircraft is safe and reliable.

Together, these modules provide a comprehensive and integrated environment for aircraft design and optimisation. By using CEASIOMpy, engineers and researchers can streamline their workflow and optimise the design process, resulting in more efficient and cost-effective aircraft development.

7.2 How to run a case

CEASIOMpy can be run using a GUI interface to build the complete workflow step by step, or from the command line to directly define the CPACS files and the modules you want to use.

```
> conda activate ceasiompy
> cd src/streamlit
> streamlit run CEASIOMpy.py
```

Figure 7.2: *Running CEASIOMpy by using the GUI*

Once the GUI is open there are fixed steps to follow to perform a calculation:

- Choose the CPACS file that is going to be used, with the possibility of visualizing the geometry as shown in the figure below.

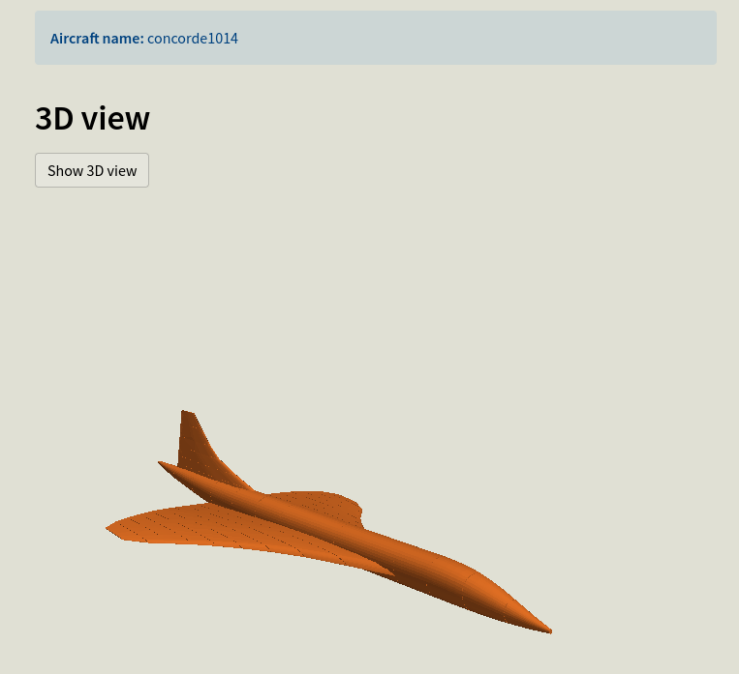


Figure 7.3: *Choosing geometry*

- Select the workflow or add the single modules. To do a CFD calculation, the workflow at least has to consist of a mesh module and the SU2Run module to perform the calculation.

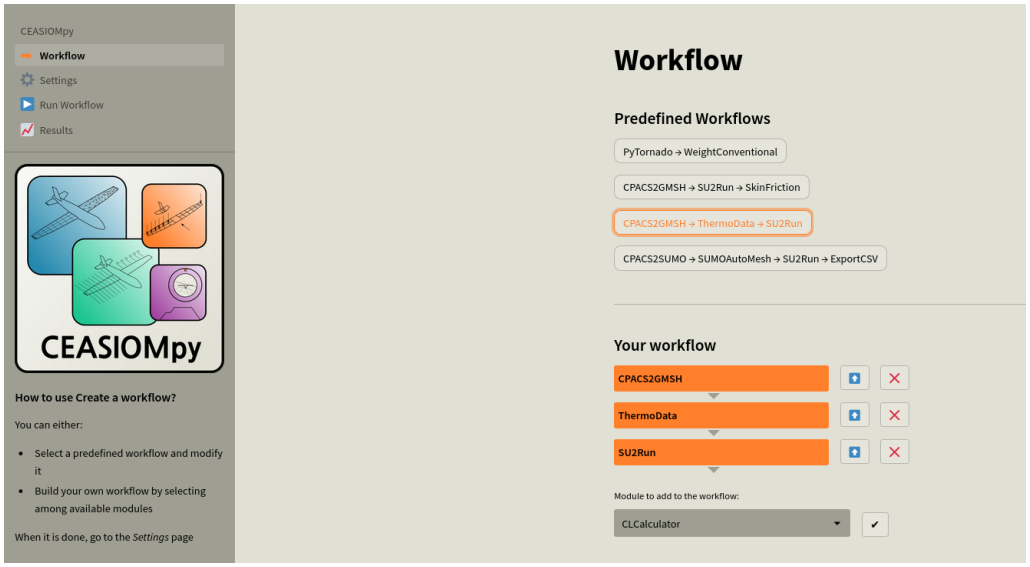


Figure 7.4: *Select the Workflow*

- Set the input variables for each module. Almost every module needs the usage of an *aeromap* that contains at least *Altitude*, *Mach number* and *angle of attack*, this values are then used to automatically calculate the air density, thanks to a package developed by Airinnova *Ambiance*, and the freestream velocity. It is also possible to choose a different *Angle of sweep*.

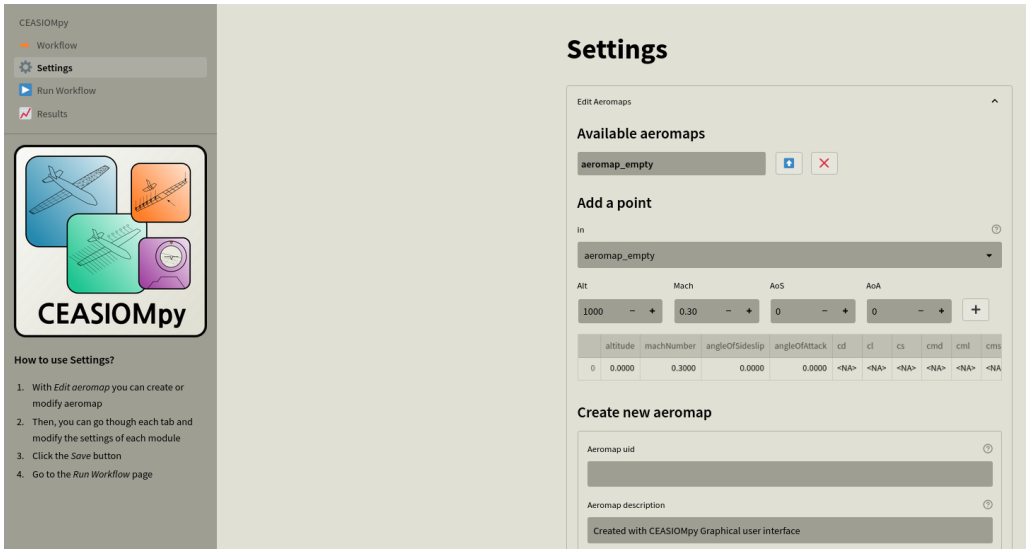


Figure 7.5: *Add the input values*

Each module has its input parameters that can be modified or the default parameters can be used. For example, the options of the "ThermoData" module are presented in Figure 7.6.

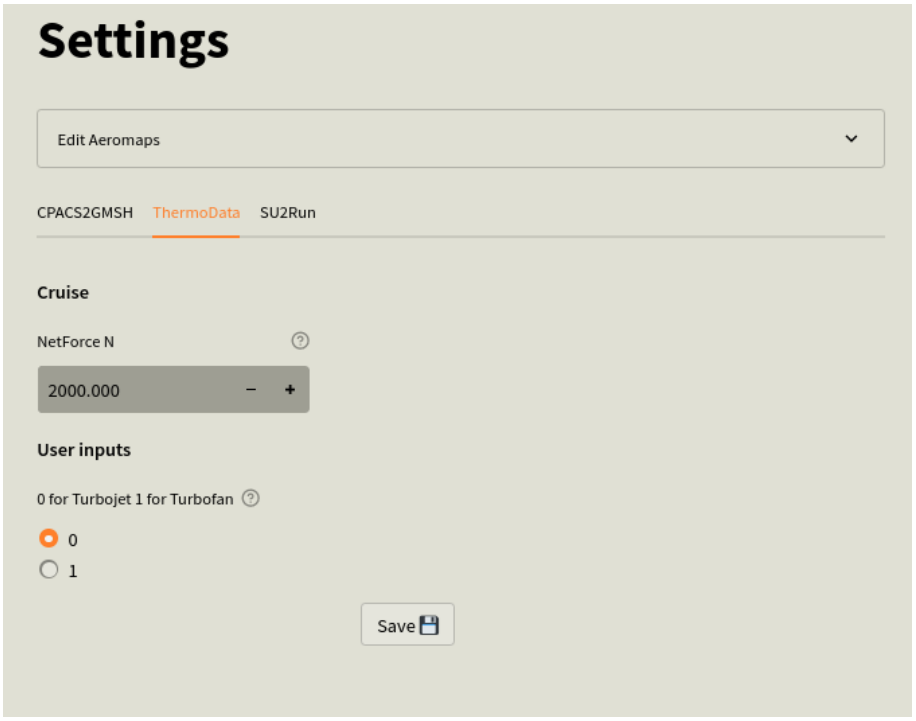


Figure 7.6: *ThermoData options*

- Once all the options are set it is possible to run the workflow. During the simulation, the residuals can be monitored to track their evolution. Upon completion, the results are automatically stored in the *"result repository"* within a folder named *Workflow00**. As each calculation is performed, the folder changes to create a record of all the actions taken. Each module is executed following the order given by the workflow and provides the results necessary for the next module by saving them within the CPACS file (.xml).

8 Meshing

8.1 Hybrid mesh

A hybrid mesh is a type of computational mesh used in numerical simulations, particularly in finite element analysis and computational fluid dynamics. It combines different types of elements, such as tetrahedra, hexahedra, pyramids, and prisms, within the same mesh.

The motivation behind using a hybrid mesh is to take advantage of the benefits of different element types in different regions of the geometry being modelled. For example, tetrahedral elements are flexible and can represent complex geometries well, while hexahedral elements are more computationally efficient and accurate for regular or structured geometries. By combining these element types judiciously, a hybrid mesh can offer a good compromise between accuracy and computational cost.

Hybrid meshes are often used in simulations where there are regions of varying geometric complexity or where certain physical phenomena need to be captured with higher accuracy. They are also employed in cases where automatic mesh generation algorithms generate meshes of mixed element types.

- **Types of Elements:** Hybrid meshes typically include a mix of tetrahedral, hexahedral, prismatic, and pyramidal elements. Each element type has its own advantages and disadvantages in terms of computational efficiency, accuracy, and ease of mesh generation.
- **Mesh Generation:** Hybrid meshes can be generated using a combination of automatic mesh generation algorithms and manual meshing techniques. Automated mesh generation tools often generate meshes with predominantly tetrahedral elements, while manual techniques may be used to refine the mesh in certain regions or to introduce specific element types where needed.
- **Mesh Adaptation:** In some cases, hybrid meshes may be dynamically adapted during the simulation to improve accuracy or computational efficiency. Mesh adaptation techniques involve refining or coarsening elements based on solution metrics such as error indicators or gradient estimates.
- **Applications:** Hybrid meshes are commonly used in simulations of complex geometries or flow configurations where different element types are needed to capture the physics of the problem accurately. They are employed in various fields, including aerospace engineering, automotive design, biomedical modelling, and environmental simulations.
- **Challenges:** While hybrid meshes offer flexibility and versatility, they also present challenges regarding mesh quality, element connectivity, and computational overhead. Managing the transition between different element types and ensuring a smooth mesh transition can be complex, particularly in regions of geometric complexity or high-flow gradients.

8.2 Hybrid mesh in Gmsh

We started with the idea of obtaining a fully hybrid mesh within the Gmsh environment. Indeed, this program offers some solutions for both the 2D and 3D parts, albeit only for very simple geometries for which only two examples are provided in the guide.

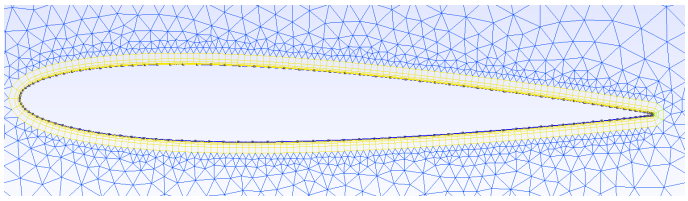


Figure 8.1: *2D NACA profile*

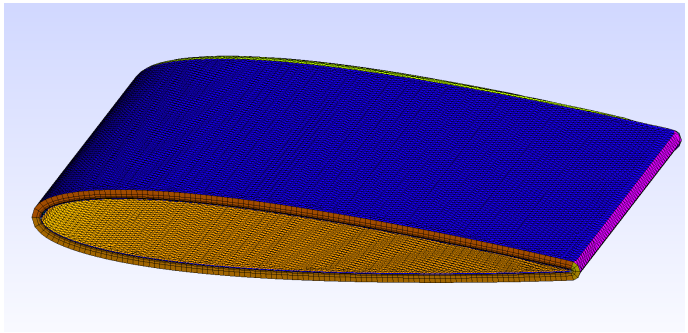


Figure 8.2: *3D NACA wing tip*

In these two examples, two methods are developed to obtain a structured boundary layer:

- By extrusion: a boundary layer can be created through extrusion using the built-in CAD kernel: this creates topological entities that will be filled with a discrete geometry (a mesh extruded along the boundary normals) during mesh generation
- By mesh constrain: in 2D, boundary layers can also be specified as a meshing constraint, through the BoundaryLayer field; this is quite a bit more general, as it handles intersections between several boundary layers, fans, etc.

As also reported by the developers themselves, these two methods are inefficient and unstable. After trying both methods on a 2D section and on a 3D geometry, obtaining poor results, we had to look for another way to achieve our goals. Consulting the limited bibliography available online regarding the open-source world, we decided to give the TetGen software a chance.

8.3 Hybrid mesh in TetGen

As previously mentioned, TetGen is a computational software tool used to generate 3D tetrahedral meshes. TetGen requires the input geometry, which can be provided as:

- A set of 3D points.
- A surface mesh defining the boundaries of the domain.
- Boundary conditions or constraints.

The input is typically supplied in formats like `.node` (node list), `.poly` (polygon format), or `.smesh` (surface mesh).

In the context of TetGen, the term "Pentagrow" refers to a specific algorithm or method employed to generate hybrid meshes. The typical process for hybrid mesh generation using Pentagrow involves:

- **Input Specification**: The input geometry defines regions where specific types of elements (tetrahedra, prisms, etc.) are required. Boundary surfaces and constraints are also defined.
- **Initial Tetrahedralization**: TetGen generates a tetrahedral mesh for the domain.
- **Prism/Hex Layer Generation**: Near surfaces or boundaries, Pentagrow grows structured layers of prism or hexahedral elements.
- **Transition Zones**: Pyramids or other transitional elements are generated to connect tetrahedral and prismatic/hexahedral regions.
- **Quality Refinement**: The entire hybrid mesh is refined to meet quality metrics, ensuring stability and accuracy in numerical simulations.

Advantages of Pentagrow for Hybrid Meshes:

- **Flexibility**: Handles arbitrary geometries and supports a mix of element types.
- **Boundary Layer Resolution**: Generates high-quality boundary layers, which are critical for simulations like computational fluid dynamics (CFD).
- **Smooth Transitions**: Ensures compatibility between different mesh regions, minimizing errors or artifacts in simulations.

Limitations:

- While Pentagrow and TetGen are powerful, generating hybrid meshes with complex transitions requires careful input preparation and post-processing to ensure compatibility.
- TetGen's capabilities for hybrid mesh generation may be limited compared to specialized tools specifically designed for hybrid meshing.

- For advanced hybrid meshing, users often integrate TetGen with other tools or frameworks to take advantage of its tetrahedralization capabilities while addressing any limitations in the features of hybrid meshing.

8.4 Our Hybrid Mesh Workflow

In light of various experiments and the resources at our disposal, we realized that the best route to follow was to opt for a tandem use of Gmesh and TetGen to achieve a result that could be concrete experimentally and consistently applicable. We can thus summarize the functioning of this tandem in the following steps:

1. Starting from the characteristic points of the geometry, Gmesh identifies and categorizes each piece of information according to the aircraft component, creating sub-volumes. To generate a mesh usable by the subsequent software, we had to proceed by cutting and merging these small volumes to obtain an overall watertight volume.

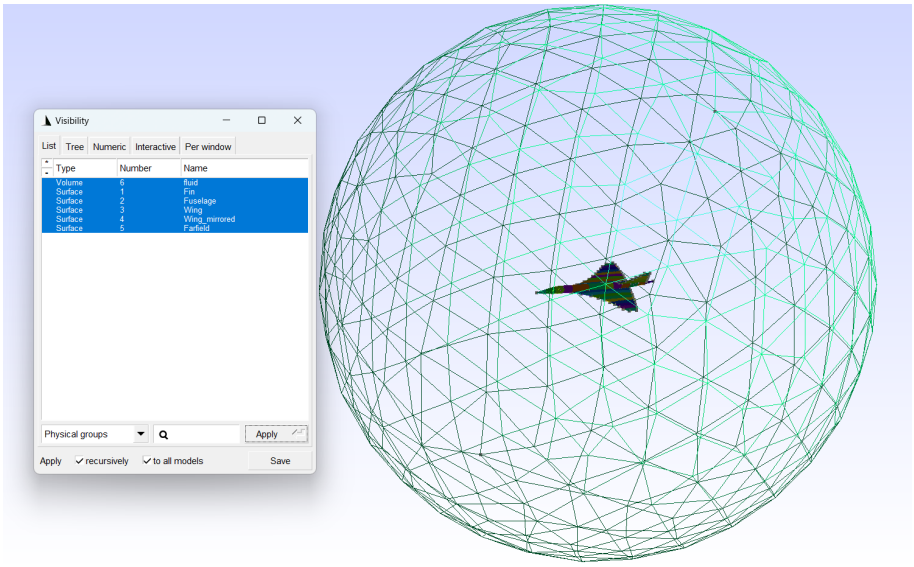


Figure 8.3: *Aircraft component meshed by Gmesh*

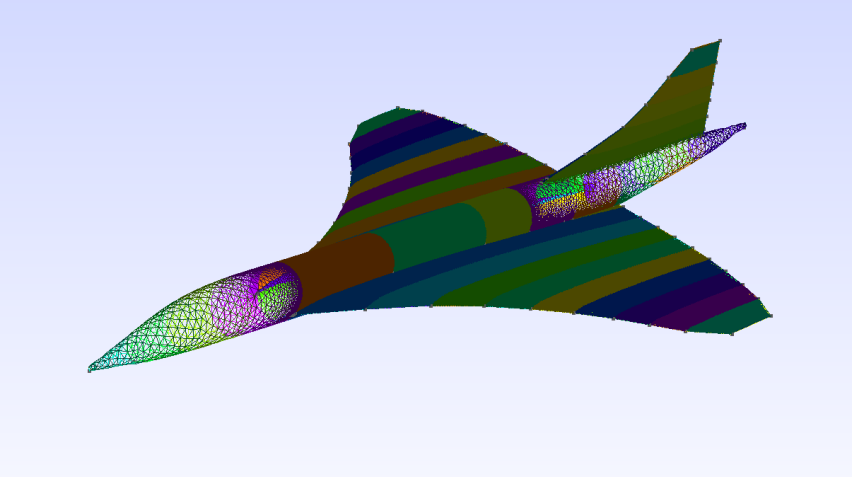


Figure 8.4: *Mesh of the entire aircraft extracted from Gmsh*

2. This mesh is processed by TetGen, which extracts the boundaries within a volume of the spherical domain, for which the operator provides the radius and center.

Figure 8.5: *Visualization and processing of the mesh using TetGen*

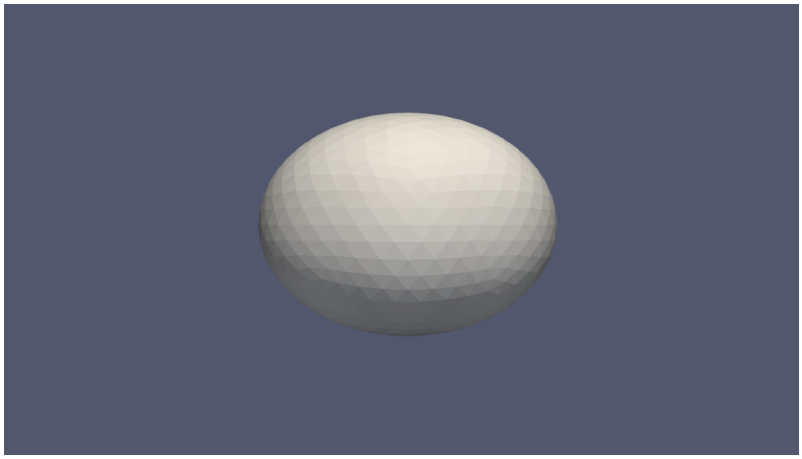


Figure 8.6: *Creation of the computational domain*

3. Pentagrow begins to construct the structured mesh along the edges of the geometry.

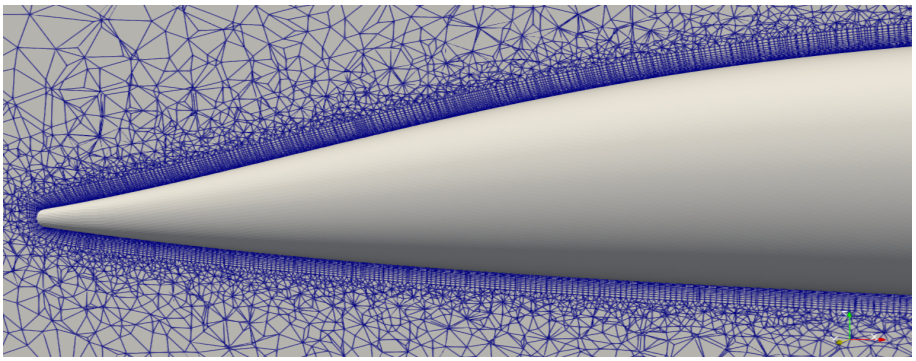


Figure 8.7: *Structured mesh construction using Pentagrow*

4. The remaining volume is filled by TetGen with an unstructured mesh.

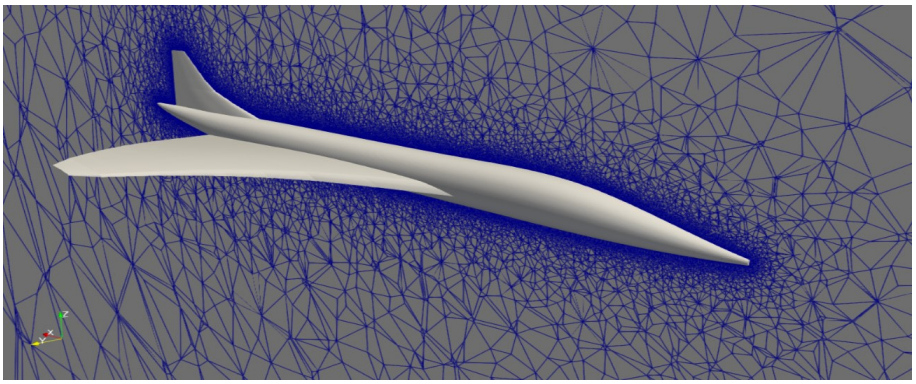


Figure 8.8: *Complete hybrid mesh*

Once the robustness of this workflow was tested, the labor-intensive part was integrating this procedure into the Ceasiompy modules and making it as user-friendly

as possible by automating most of the process. The subsequent implementation of this part involves enabling the program to recognize the various components of the aircraft (fuselage, wings, tail, etc.) to automatically customize the mesh sizing. This would allow refinement in areas of critical interest (e.g., trailing edges and leading edges of the wings).

9 PyCycle

Thermodynamics is fundamental to the analysis of a gas turbine engine. The main objective of the analysis is to determine the performance parameters, particularly thrust and specific fuel consumption. To perform the calculations, it is necessary to use constraints, which in the case of aircraft engines are usually given by the maximum turbine inlet temperature and the Mach number for the cruise phase.
Analysis of the thermodynamic cycle is therefore one of the first steps in carrying out engine design and after a more precise analysis of the individual components, this is again given as feedback to the thermodynamic analysis to verify that the desired performance values are maintained, thus creating an iterative process.
To be able to perform this type of calculation within Ceasiompy, the open-source code PyCycle was added to CEASIOMpy's environment.
The primary goal for the development of PyCycle was therefore to build a cycle analysis tool that would be efficient to apply in a vehicle-level multidisciplinary optimization context, while still maintaining modularity and flexibility [16].
This module is therefore beneficial to add within CEASIOMpy therefore that analyses can be carried out including the boundary conditions given by the aircraft engines to be analyzed.

9.1 PyCycle structure

The basic structure of PyCycle can be represented through the use of an XDSM (eXtended Design Structure Matrix) diagram, which is a tool used to visualize MDO (Multi-disciplinary design optimization) processes, visible in the figure below.

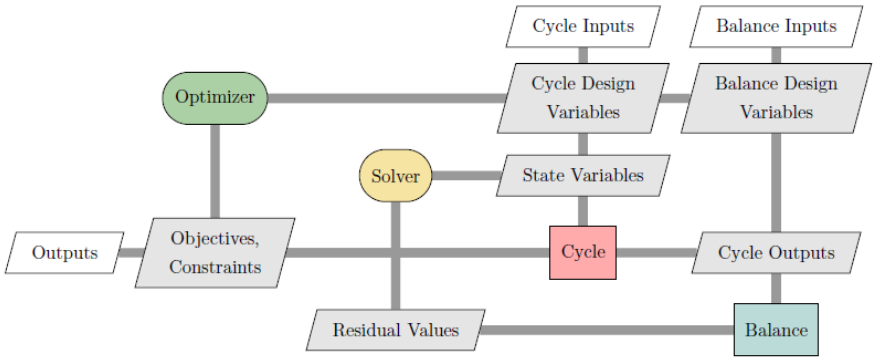


Figure 9.1: *Basic structure of PyCycle [16]*

Along the diagonal, the four main blocks are easily identifiable: Optimiser, Solver, Cycle, and Balance. These blocks exchange information with each other as visible through the grey lines

- **Optimiser:**
  This block contains the physical dependencies and the design rules, as a set of implicit state variables and associated non-linear residual equations.
- **Solver:**
  The task of this block is to converge the non-linear equations contained in the optimizer.
- **Cycle:**
  Inside this block, there are all the thermodynamic equations needed to model the engine, therefore it is the most important one. It's easy to note that it is the one with more exchanges of information with the other's blocks.
- **Balance:**
  It's necessary to find the values of the design variables that satisfy the constraints and minimize the objective specified for the problem.

It should also be noted that this code is written using the openMDAO framework [17]. PyCycle differentiates itself from other similar software in computing thermodynamic properties because, unlike other programs that present as the smallest part on which to act to modify the engine structure, the macrostructures of the compressor, turbine, and so on, with PyCycle, elements are themselves composed of several nested layers of smaller computational components that capture the different types of calculations. The automatic total derivative functionality of the OpenMDAO framework is then relied on to combine these simpler partial derivatives to obtain the derivatives for the overall elements and cycle model [16].

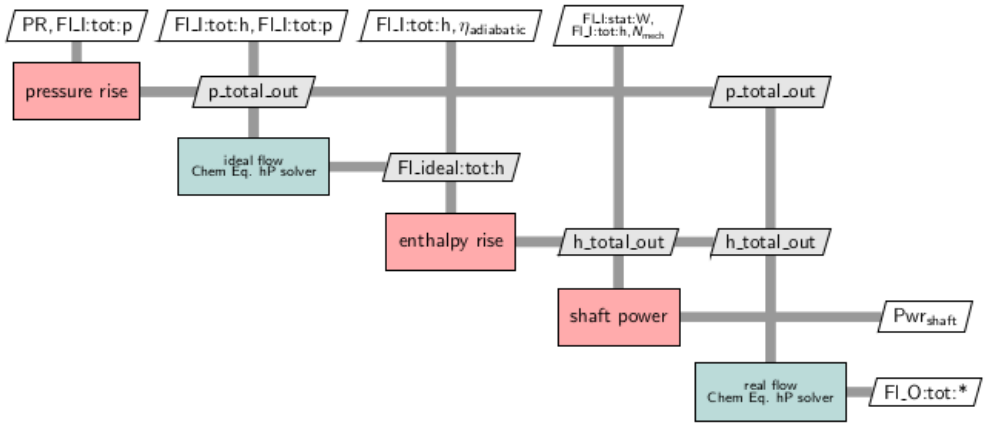


Figure 9.2: *XDSM diagram for the PyCycle Compressor element. Engineering calculations are shown as red boxes. Chemical equilibrium flow solutions are shown in green. [16]*

Moreover, thanks to the predefined libraries, it is possible not to have to act on

individual blocks by putting them together but to use predefined templates for the compressor, the turbine and all the various parts of the engine, which can be assembled to compose the engine to study by using the *Balance* and the *Connect* blocks.

9.2 PyCycle implementation ThermoData module

To add PyCycle inside the Ceasiompy environment it was necessary to adapt the original code for the merging and add some features. The addition of this module is an important improvement within CEASIOMpy, allowing not only the aerodynamic part to be involved in the simulation but also the thermodynamic part with the boundary conditions given to the engines.

9.2.1 Functions added

The two main functions added to the CEASIOMpy ThermoData module, are *Turbojet_func* and *Turbofan_func*. With these functions it is possible to perform automatic calculations for the simulated engine, obtaining the boundary conditions for a CFD simulation or only the values to evaluate the performances of the engine.

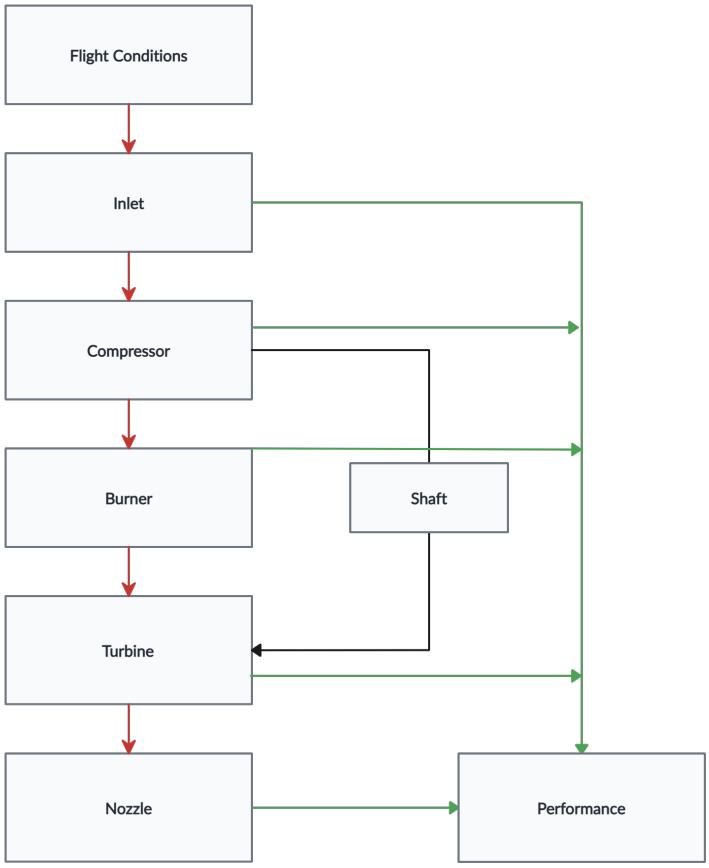


Figure 9.3: *Compressor scheme in PyCycle*

In Figure 9.3, it is possible to visualize the relationships between the different elements used to model a complete turbojet engine. The red lines represent flow connections, the black lines indicate physical shaft connections, and the green lines correspond to secondary data connections between elements. In this case, these connections provide information to the performance block, enabling the required calculations.

This modular and easily customizable structure is well suited for the design and analysis of different engine configurations.

Once the ThermoData module is executed, it generates an output file called *EngineBC.dat*. This file contains the parameters considered most relevant for the analysis of engine performance. The specific values included depend on whether a turbojet or a turbofan engine is selected. The results obtained are presented below.

- **Turbojet**
  1. `T_stat_out= Nozzle static temperature outlet[K]`
  2. `T_stat_out= Nozzle static temperature outlet[K]`
  3. `P_tot_out= Nozzle total pressure outlet [Pa]`
  4. `P_stat_out= Nozzle static pressure outlet [Pa]`
  5. `V_stat_out= Nozzle static velocity outlet[m/s]`
  6. `MN_out= Nozzle Mach number outlet [adim]`
  7. `massflow_stat_out= Nozzle massflow outlet [Kg/s]`
- **Turbofan**
  1. `T_tot_out_core = Total temperature outlet from the core [K].`
  2. `V_stat_out_core = Static velocity outlet from the core [m/s].`
  3. `MN_out_core = Mach number outlet from the core [adim].`
  4. `P_tot_out_core = Total pressure outlet from the core [Pa].`
  5. `massflow_out_core = Mass flow outlet from the core [kg/s].`
  6. `T_stat_out_core = Static temperature outlet from the core [K].`
  7. `T_tot_out_byp = Total temperature outlet from the bypass [K].`
  8. `V_stat_out_byp = Static velocity outlet from the bypass [m/s].`
  9. `MN_out_byp = Mach number outlet from the bypass [adim].`
  10. `P_tot_out_byp = Total pressure outlet from the bypass [Pa].`
  11. `massflow_stat_out_byp = Mass flow outlet from the bypass [kg/s].`
  12. `T_stat_out_byp = Static temperature outlet from the bypass [K].`

Once the ThermoData module has been executed, it can be left as a stand-alone block or connected to other modules in CEASIOMpy, creating a Workflow.
A typical example of a Workflow is presented in figure 9.4.

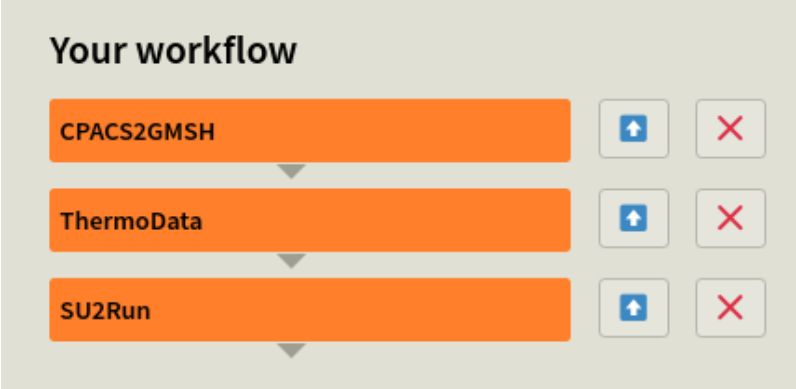


Figure 9.4: *ThermoData Workflow*

With this Workflow, the module is connected to two other important modules: CPACS2GMSH and SU2RUN. By combining these three modules, the aim is to obtain an automated simulation of an aircraft using engine boundary conditions.
Interactions through the modules are carried out using the CPACS file to which each module adds information necessary for the execution of the next.

```
<BC>
  <temperatureOutlet mapType="vector">870.184</temperatureOutlet>
  <pressureOutlet mapType="vector">176981</pressureOutlet>
</BC>
```

Figure 9.5: *Values exchanged between ThermoData and SU2Run*

For example, the ThermoData module stores the pressure and temperature at the engine outlet in the CPACS file, as shown in the figure 9.5, these are then taken from the SU2Run module to be added to the SU2 configuration file and then simulation is run.
To make the correct calculations, it is also necessary to know the inlet conditions for the engine. These are calculated from the altitude using the *Ambiance* module, which is a full implementation of the ICAO 1993 atmosphere standard written in Python. This module was developed by Airinnova and implemented in CEASIOMpy.
The lines added to the Su2 configuration file can be seen in the following figure.

```
INLET_TYPE = TOTAL_CONDITIONS
MARKER_INLET = (INLET_ENGINE, 288.15, 101325.0, 1,0,0, OUTLET_ENGINE,870.184,176981.0, 1,0,0)
```

Figure 9.6: *Added lines to SU2 configuration file*

As can be seen from figure 9.6, for the inlet it is necessary to write the total temperature [K] and pressure [Pa] and the same for the outlet, it is also required to give the flow direction unity vector in the example = (1, 0, 0), that corresponds to the x-direction.

9.2.2 RANS configuration template

Another major contribution of this thesis to CEASIOMpy is the implementation of the capability to perform RANS simulations. This development was achieved through improvements in two main areas.

First, a new meshing strategy was introduced, based on the methods described in Chapter 8. This approach enables the generation of hybrid meshes capable of resolving the boundary layer and correctly applying the no-slip condition on solid surfaces.

Second, the simulation workflow was automated by extending the SU2Run module with a new function called *su2config_rans.py*. This function is designed to automatically generate a SU2 configuration file by extracting the necessary information from the CPACS file and adapting it to the selected geometry.

The SU2 configuration file (.cfg) contains all the parameters required for the CFD simulation. It is read by the SU2 solver to define key aspects of the computation, including the mesh, flow conditions, turbulence models, and numerical schemes. Therefore, it represents a critical component for obtaining accurate and reliable simulation results.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
% SU2 configuration file                                                       %
% Case description: _____________________________________________________     %
% Author: ________________________________________________________________     %
% Institution: ___________________________________________________________     %
% Date: __________                                                             %
% File Version 8.0.0 "Harrier"                                                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ------------ DIRECT, ADJOINT, AND LINEARIZED PROBLEM DEFINITION ------------%
%
% Solver type (EULER, NAVIER_STOKES, RANS,
%              INC_EULER, INC_NAVIER_STOKES, INC_RANS,
%              NEMO_EULER, NEMO_NAVIER_STOKES,
%              FEM_EULER, FEM_NAVIER_STOKES, FEM_RANS, FEM_LES,
%              HEAT_EQUATION_FVM, ELASTICITY)
SOLVER= EULER
%
% Specify turbulence model (NONE, SA, SST)
KIND_TURB_MODEL= NONE
%
% Specify versions/corrections of the SST model (V2003m, V1994m, VORTICITY, KATO_LAUNDER, UQ, SUSTAINING)
SST_OPTIONS= NONE
%
% Specify versions/corrections of the SA model (NEGATIVE, EDWARDS, WITHFT2, QCR2000, COMPRESSIBILITY, ROTATION, BCM, EXPERIMENTAL)
SA_OPTIONS= NONE
%
% Transition model (NONE, LM)
KIND_TRANS_MODEL= NONE
% Specify versions/correlations of the LM model (LM2015, MALAN, SULUKSNA, KRAUSE, KRAUSE_HYPER, MEDIDA, MEDIDA_BAEDER, MENTER_LANGTRY)
LM_OPTIONS= NONE
%
% Specify subgrid scale model(NONE, IMPLICIT_LES, SMAGORINSKY, WALE, VREMAN)
KIND_SGS_MODEL= NONE
```

Figure 9.7: *Configuration file example*

The new function was developed starting from the existing *su2config.py*, which was

extended by introducing a dedicated template for RANS simulations. The number of configurable options was reduced in order to improve control over the parameters influencing convergence. Some inputs are fixed, while others are either set through the graphical user interface or automatically extracted from the CPACS file.

An important feature of this implementation is the automatic computation of the Reynolds number. This value is determined based on the aircraft geometry, a selected characteristic length, and the aeromap parameters (altitude and Mach number), and is then directly written into the SU2 configuration file.

In typical aircraft simulations, the Reynolds number is very high, usually on the order of 10^6–10^7, indicating turbulent flow conditions. However, its estimation is not straightforward, as it requires the definition of a suitable reference length. In most cases, the mean aerodynamic chord C of the wing is used, although the appropriate choice may depend on the specific part of the aircraft being analyzed.

In the present work, the mean aerodynamic chord was selected as the reference length, as it represents a suitable characteristic dimension when analyzing the overall aerodynamic performance of the aircraft, particularly in terms of lift and drag.

9.2.3 Limitations

The current implementation presents several limitations that could be addressed to improve the integration between CEASIOMpy and PyCycle.

The main limitation concerns the simulation of different engine types without requiring modifications to the existing functions. At present, only two engine configurations are supported: turbofan and turbojet. Their main characteristics are predefined and cannot be modified by the user through the CEASIOMpy graphical interface.

Future improvements could include the introduction of predefined configurations for a wider range of engine types, as well as greater flexibility in modifying key parameters such as component efficiencies and imposed constraints. However, it should be noted that implementing such features would require a significant development effort and is therefore beyond the scope of the present work.

Another limitation is related to the computational time required to obtain results, particularly for the turbofan case. While convergence is achieved within a few seconds for simpler configurations such as the turbojet, the turbofan simulations may require several minutes to converge.

One possible improvement would be to refine the initial guess values used by the solver, adapting them to the input conditions. This could significantly accelerate the convergence process.

Additional improvements could also be made in the generation of the configuration file for RANS simulations. Currently, many parameters are automatically derived from the geometry, while others remain fixed across all simulations. One of the most influential parameters for convergence is the CFL number, which at present must be manually adjusted by the user.

Automating the estimation of the CFL number would improve convergence behavior and further enhance the level of automation of the simulation process, which represents one of the primary objectives of CEASIOMpy.

10 Results

10.1 LabAR

At first, we decided to validate the new meshing generation method and the RANS simulation implementation comparing the obtained results to the Euler simulation already implemented in CEASIOMpy. After some test cases, we moved to the first 3D full geometry of the LabAR. The LabAR is a wind tunnel model with a simple geometry, perfect for an easy and rapid convergence test case.

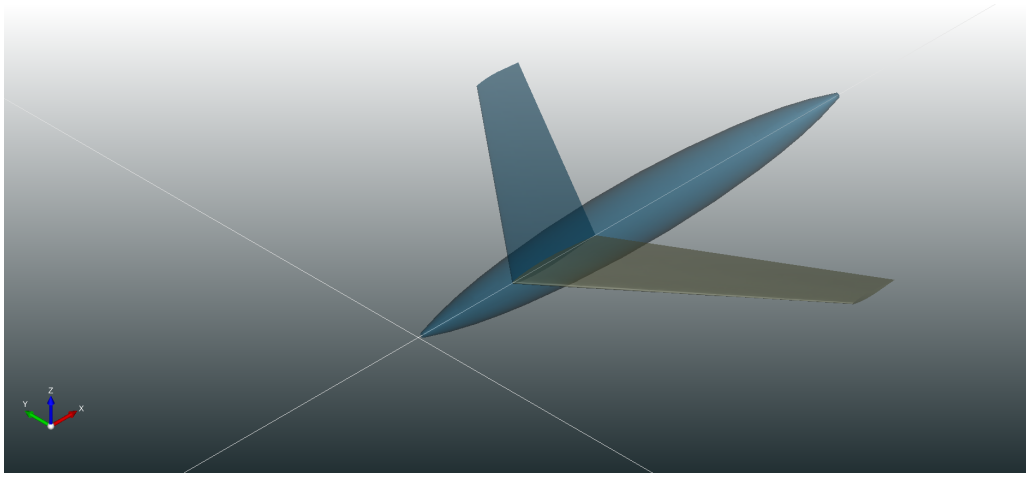


Figure 10.1: *LabAR geometry visualized with CPACSCreator*

The first simulation carried out was for the Eulerian case, using the features already present in CEASIOMpy before the developments discussed in this thesis. We then proceeded with an unstructured mesh automatically generated by the CEASIOMpy CPACS2GMSH module and then performed the simulation with the SU2RUN module. Figure 10.2 shows the mesh used for the Eulerian simulation while Figure 10.3 shows the hybrid mesh used for the RANS simulation.
It is also important to show the enlargement, see figure 10.4, which shows more clearly the development of the structured part of the hybrid mesh and the interface with the unstructured part.

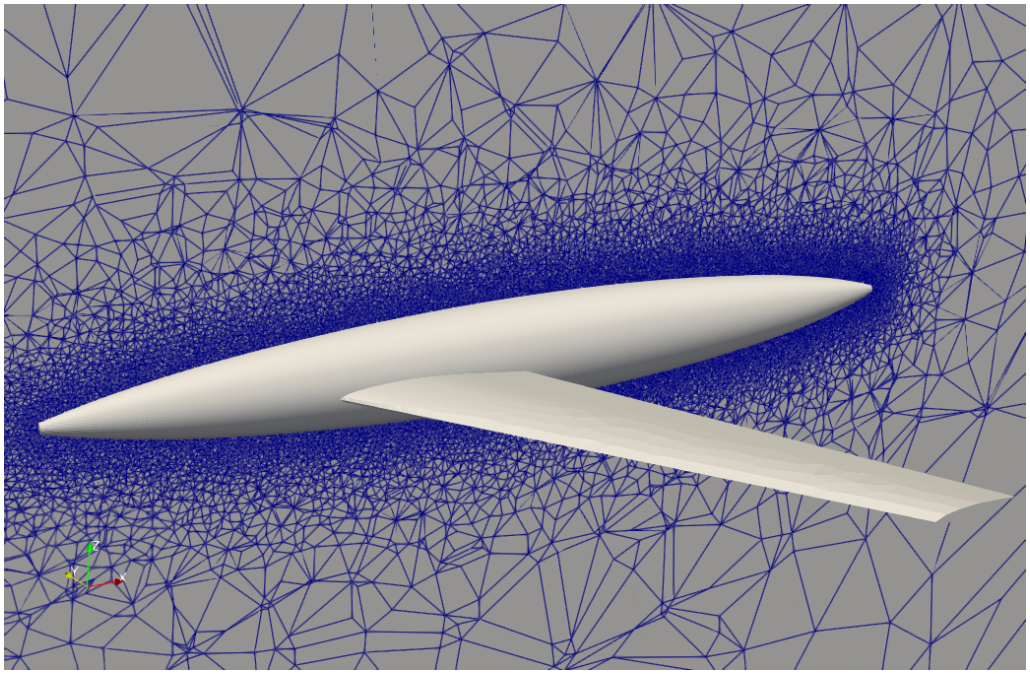


Figure 10.2: *LabAR unstructured mesh*

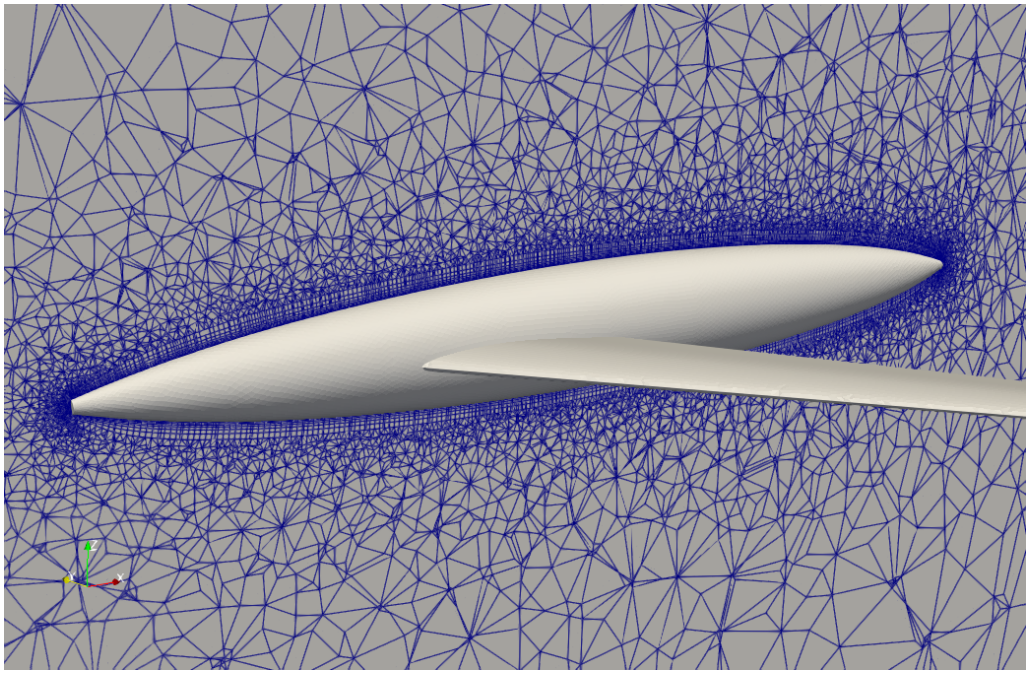


Figure 10.3: *LabAR hybrid mesh*

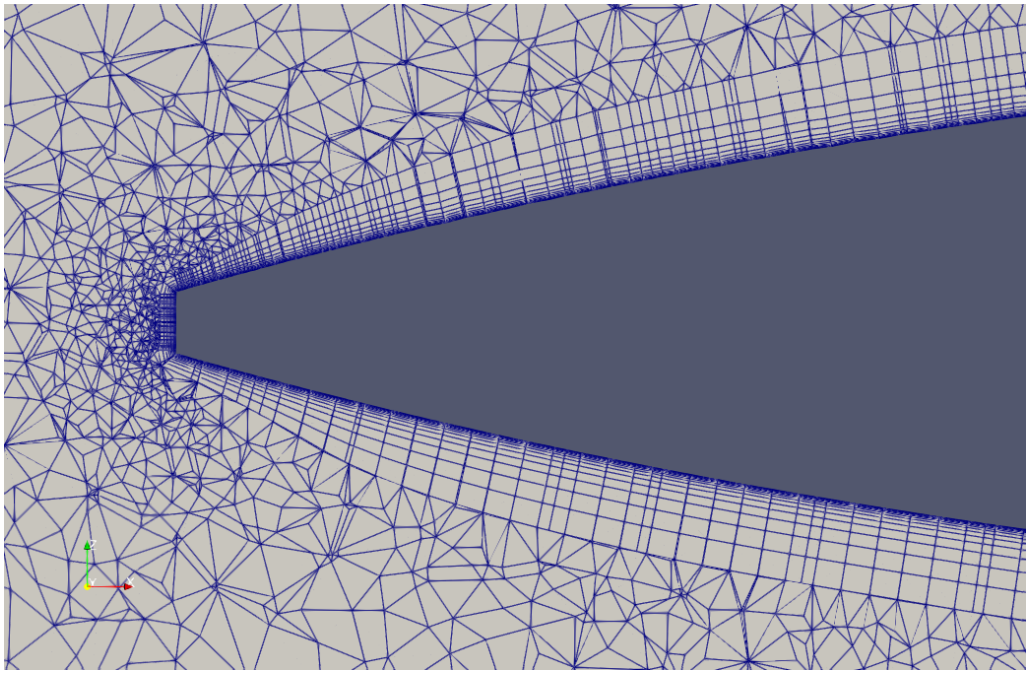


Figure 10.4: *LabAR hybrid mesh enlargement*

Simulation Parameters	
Altitude	0 [m]
Mach	0.3
AOA	0°

Table 1: *Operating conditions LabAR*

After mesh generation, we continued by deciding on the operating conditions for the CFD simulation, which are summarized in the table above.
After deciding on the operating conditions, it was only necessary to start the simulation in CEASIOMpy which automatically created the config file for SU2 in both configurations and added the necessary parameters for the RANS case such as the Reynolds number ≈ 1000000.

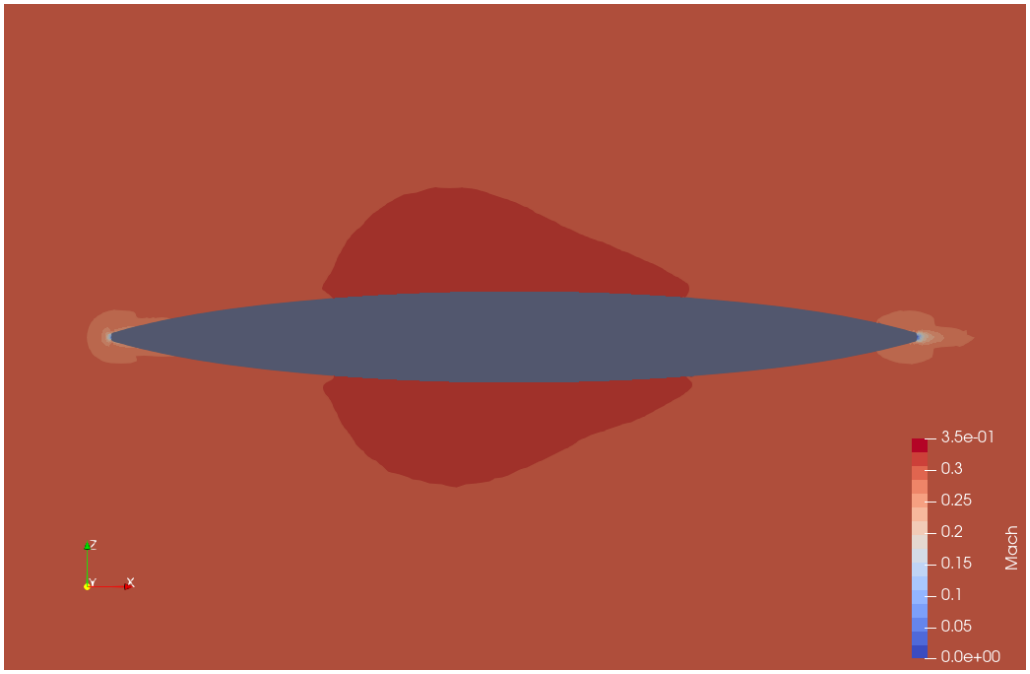


Figure 10.5: *LabAR Euler Mach distribution*

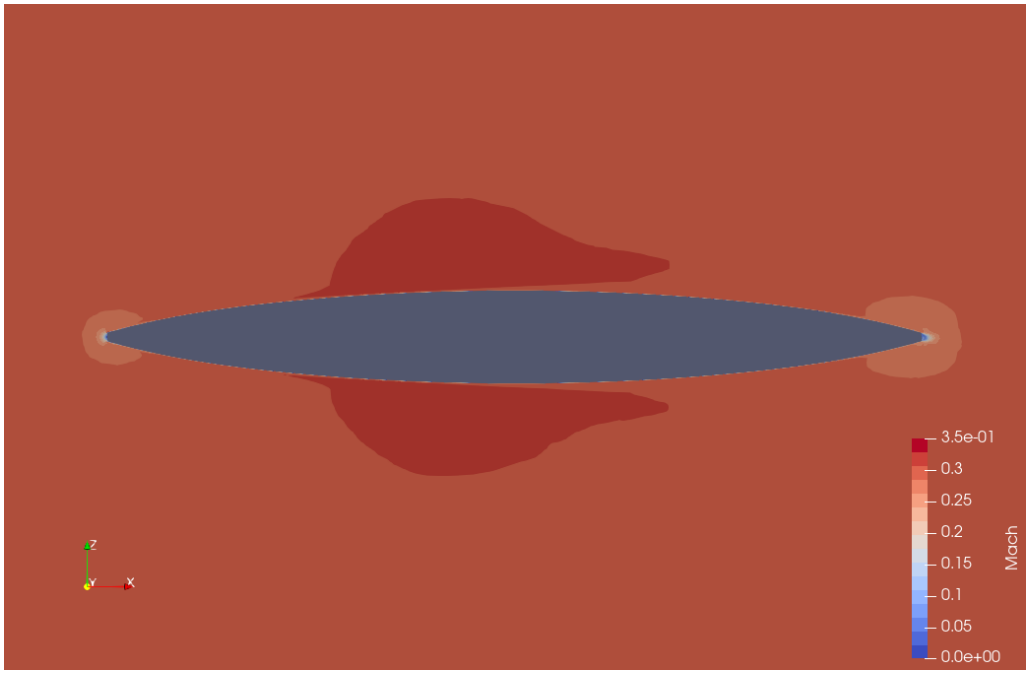


Figure 10.6: *LabAR RANS Mach distribution*

The figures above 10.6 10.5, shows the distribution of Mach number over the aircraft's surface and were obtained by sectioning with a plane passing from the nose to the aircraft's tail. As can be seen, the distributions are very similar, although the RANS calculation captures very clearly the viscous interactions near the aircraft surface.

From this first analysis it can be seen that the contribution of the new mesh generation system for performing RANS calculations has added a fundamental feature to CEASIOMpy.
To complete this work, we added some other contours just for the RANS simulations, which can be seen in the following figures.

Figure 10.7: *LabAR RANS cp distribution*

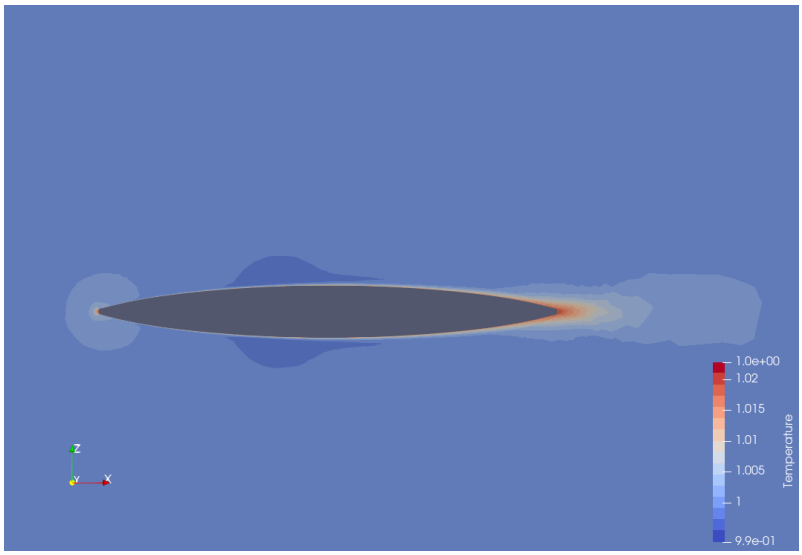


Figure 10.8: *LabAR RANS temperature distribution*

Figure 10.9: *LabAR RANS skin friction coeff. distribution*

To best visualize how the mesh captures the presence of the boundary layer, the Mach number distribution from the surface of the plane to the undisturbed part is shown in the figure below 10.10 .
The distribution was obtained by plotting the Mach number values on a line normal to the aircraft's surface. As can be seen, the Mach number goes from 0 on the surface to 0.3 at the undisturbed part of the flow. The fact that $Mach = 0$ on the surface shows that the no-slip condition is respected.

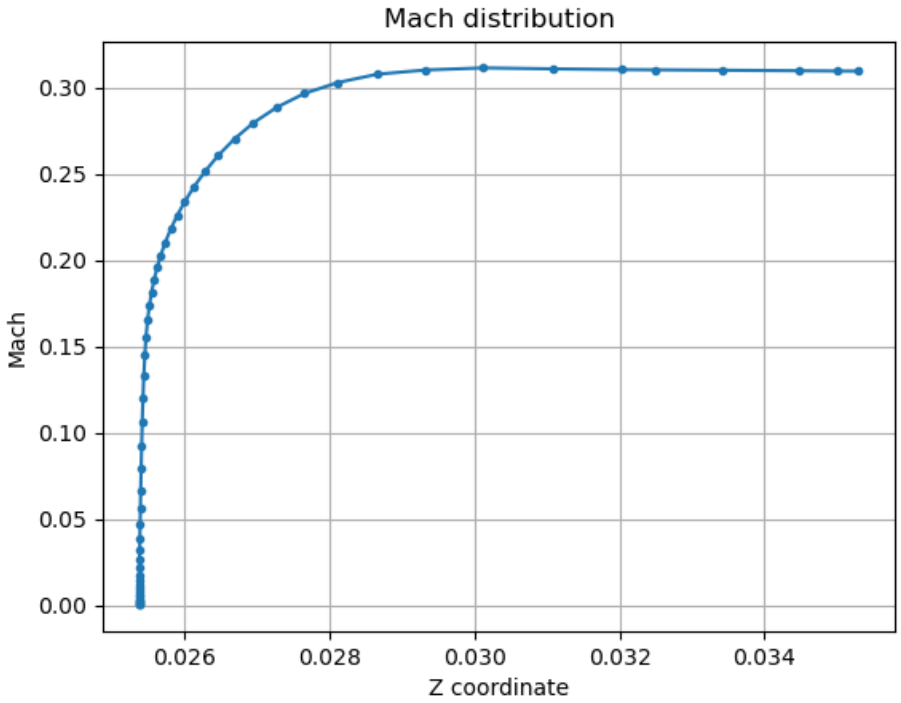


Figure 10.10: *LabAR RANS Mach number*

To complete the discussion on LabAR, the evolution of the residuals is shown below, which, as can be easily seen, converge very quickly, in less than 2000 iterations.

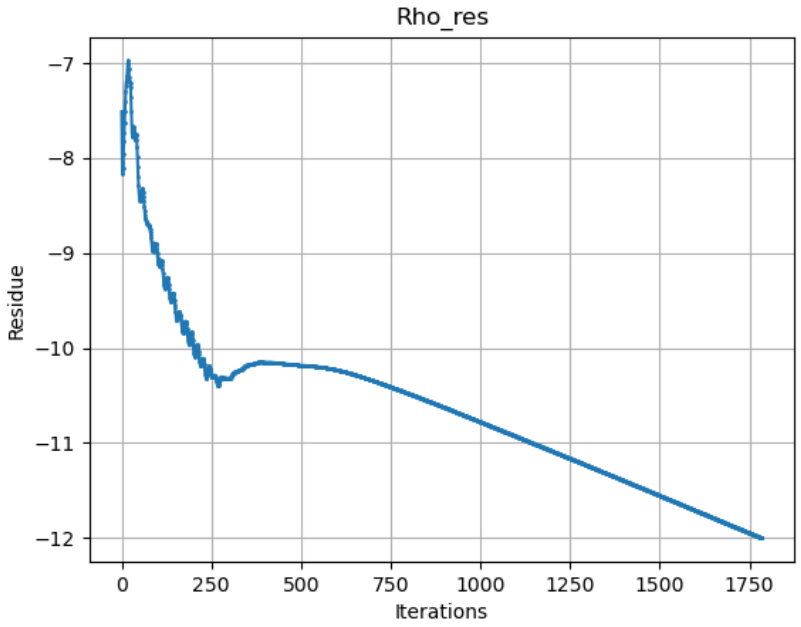


Figure 10.11: *LabAR residue convergence*

10.2 J28

After running the test simulation with LabAR and seeing that the newly implemented workflow worked correctly, we continued by running further simulations with more complex geometries. The geometry presented now is again taken from a CPACS file and was provided by KTH for simple tests on CEASIOMpy. The aircraft shown below is called a J28, it is a small aircraft with the special feature of having a rear wing.

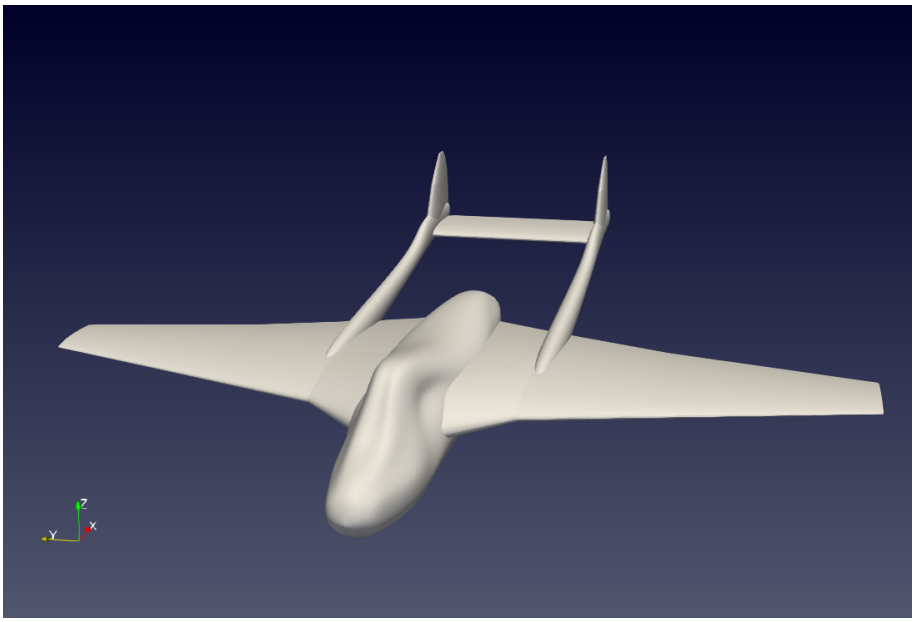


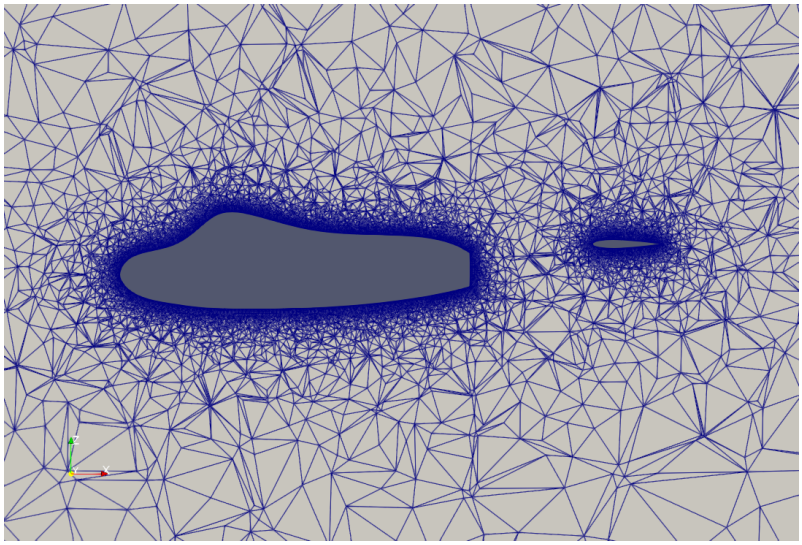
Figure 10.12: *J28 geometry*

Two different simulations were carried out for this aircraft with the same conditions except for the variation of the aircraft's angle of attack. By varying the angle of attack, an attempt was made to simulate different flight conditions and see the differences in the skin friction coefficient.

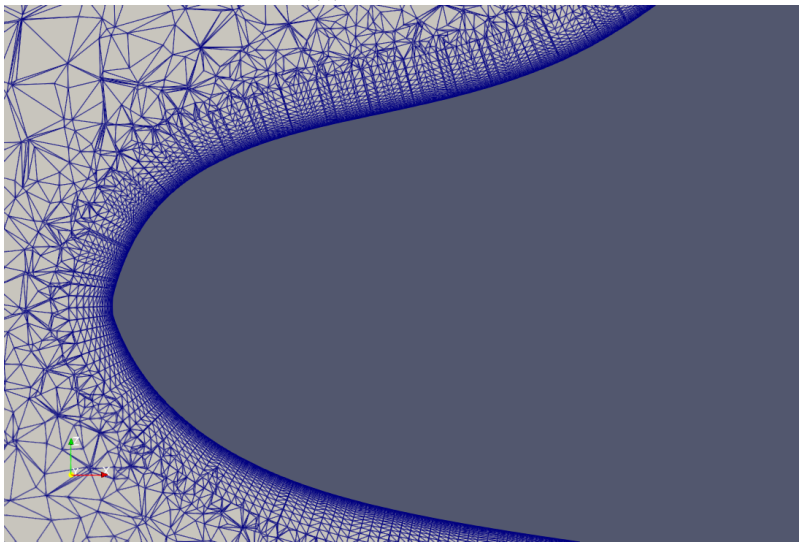
Simulation Parameters	
Altitude	0 [m]
Mach	0.3
AOA	0°and 7°

Table 2: *Operating conditions J28*

The mesh used for the study is shown below in figure 10.13. The first layer is 0.0003 m and there are 35 layers with a growth factor of 1.25 up to the triangular mesh area.



(a) *Full mesh J28*



(b) *Zoom on the nose for the J28 mesh*

Figure 10.13: *Mesh generated for J28 simulation*

The results obtained in the two simulations are now presented, considering that for both angles of attack, the no-slip condition is respected as visible in the figure 10.14 where the number of mach on the aircraft surface is presented.

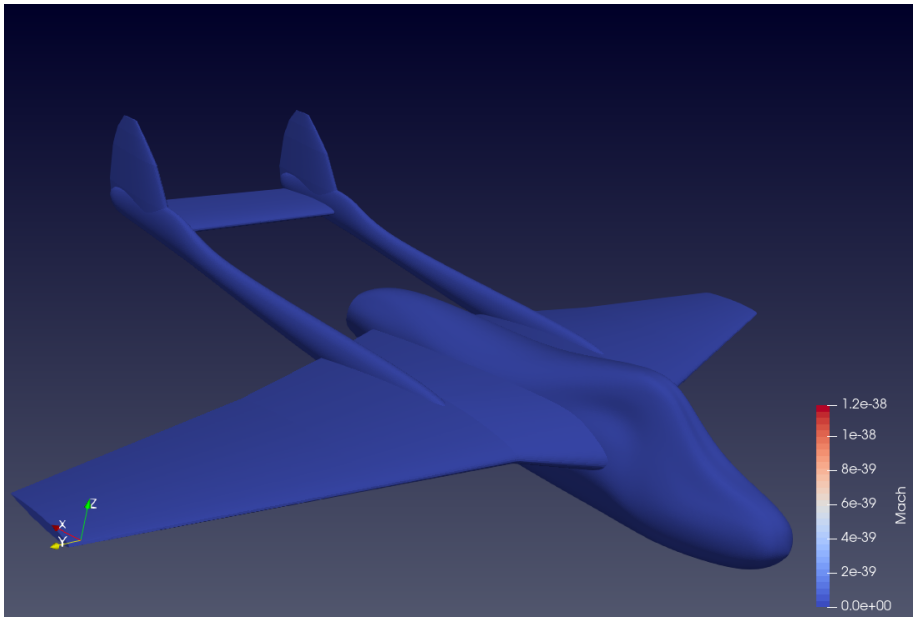
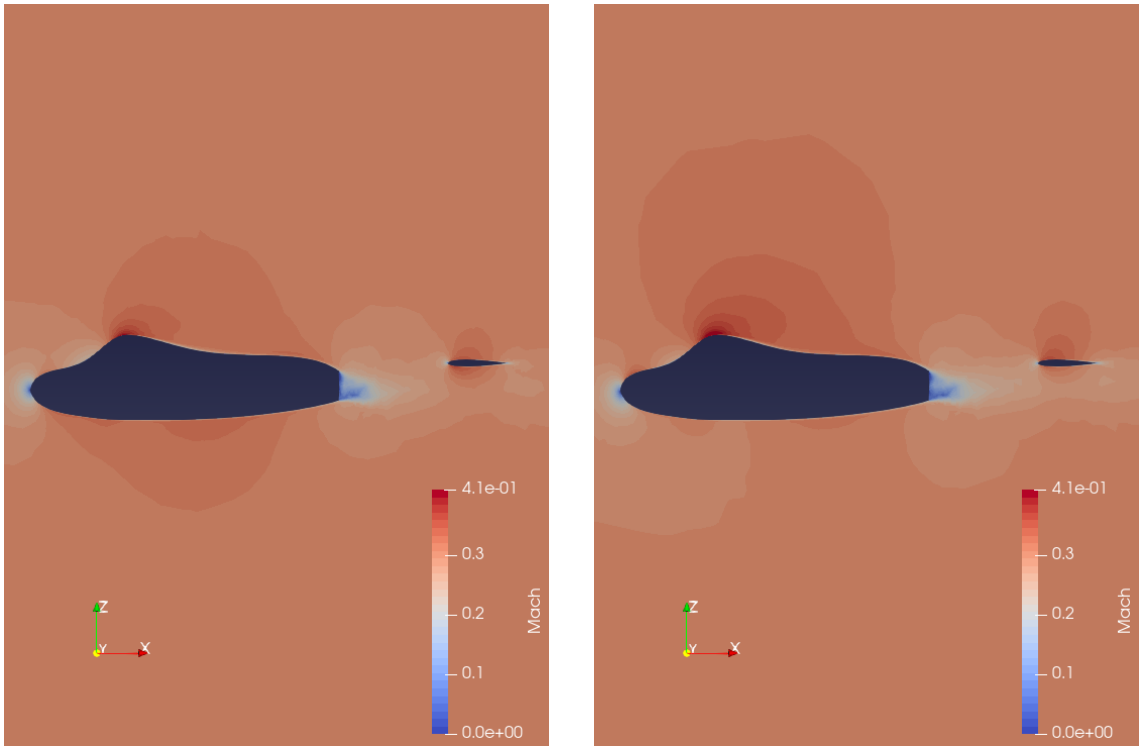


Figure 10.14: *J28 mach distribution on the surface*

The results obtained with the two angles of attack are now compared. The scaling used is the same therefore the differences in the two cases under analysis are more visible.

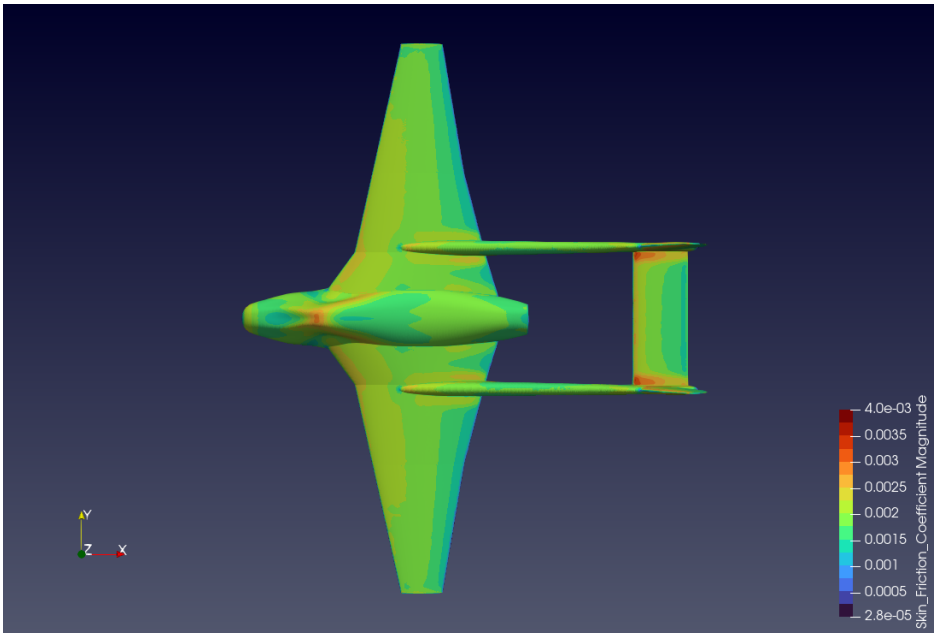


(a) *J28 mach contour AOA 0°* **(b)** *J28 mach contour AOA 7°*

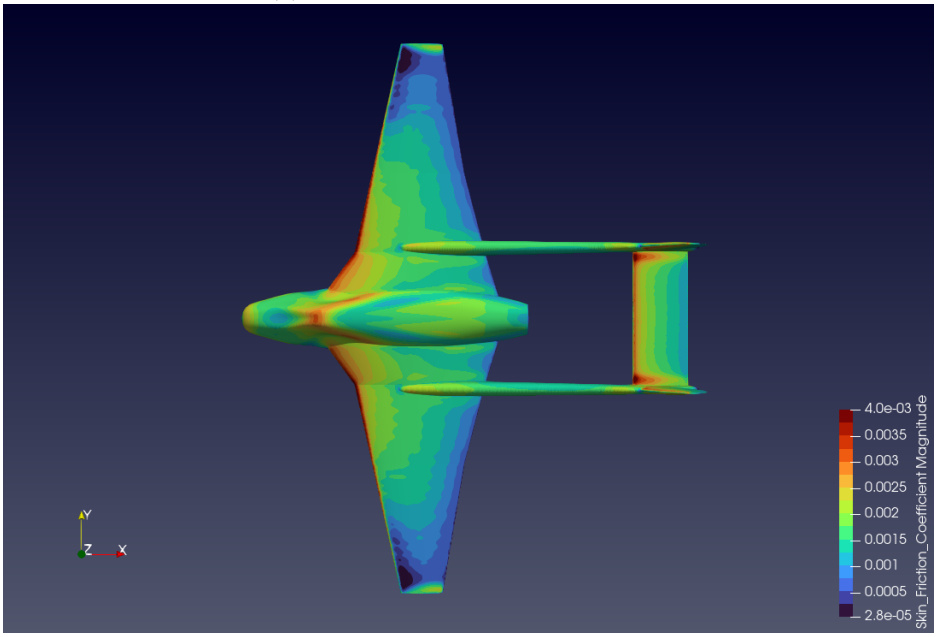
Figure 10.15: *J28 mach contour*

A significant result regarding a RANS simulation is now reported: the contour on the aircraft surface of the skin friction coefficient, whose formula is given here.

$$C_f = \frac{\tau}{\frac{1}{2}\rho V^2} \tag{10.1}$$



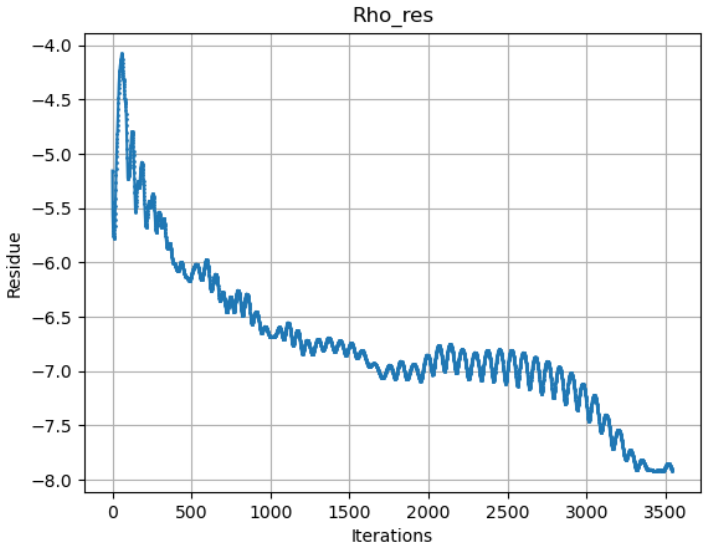
(a) *J28 skin friction coefficient AOA 0°*



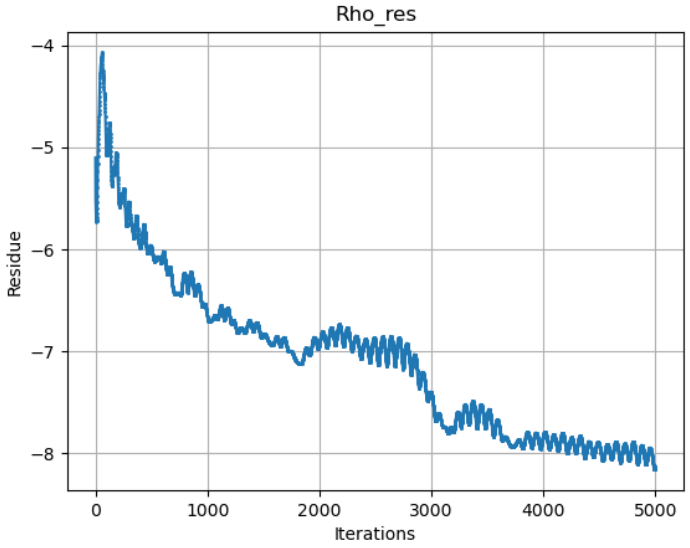
(b) *J28 skin friction coefficient AOA 7°*

Figure 10.16: *J28 skin friction coefficient*

As shown in the figure above, there is a substantial difference in the skin friction coefficient as the angle of attack varies. In fact, in the case of AOA=0°, it can be seen that the distribution is almost uniform over the entire upper part of the wings. When the angle of attack rises to 7 °, we can see a sudden drop in the coefficient caused by a wing stall in the tip part.
To complete the presentation of this case study, the residuals are shown in figure 10.17. As can be seen there is a good convergence, further demonstrating the quality of the mesh and the parameters used in the configuration file.



(a) *J28 residues AOA 0°*



(b) *J28 residues AOA 7°*

Figure 10.17: *J28 residues*

10.3 BLI 3D

After studying the previous cases, we attempted to perform CFD simulations to study an aircraft with a BLI engine. As explained in Chapter 3, there are two different types of BLI. The first has engines that receive an asymmetric airflow, as in the case of wing-mounted engines or a BWB. The second case, which was chosen in the study of this thesis, where the engine receives symmetrical airflow and is mounted at the rear of the fuselage.
In the present case, the geometry is elementary to carry out a qualitative study of the capabilities added to CEASIOMpy. A vertical stabilizer, normally used in this type of aircraft for manoeuvrability, has been avoided as it would have caused excessive asymmetry in the speed profile. The geometry used is presented in the figure below.

Figure 10.18: *BLI 3D geometry*

10.3.1 Structured mesh simulation

To study an engine with BLI, we first decided to develop a structured mesh using the *ICEM* software. This licensed software allows the development of structured meshes, i.e. meshes in which the position of each cell is unique and obtainable as if it were an element of an array. Structured meshes differ from unstructured meshes in which the position of a cell is determined by its neighbouring cells.
This mesh was made as light and performant as possible with two axes of symmetry, one on the y-axis and the other on the z-axis, thus allowing a quarter of it to be studied more precisely and with denser cells.

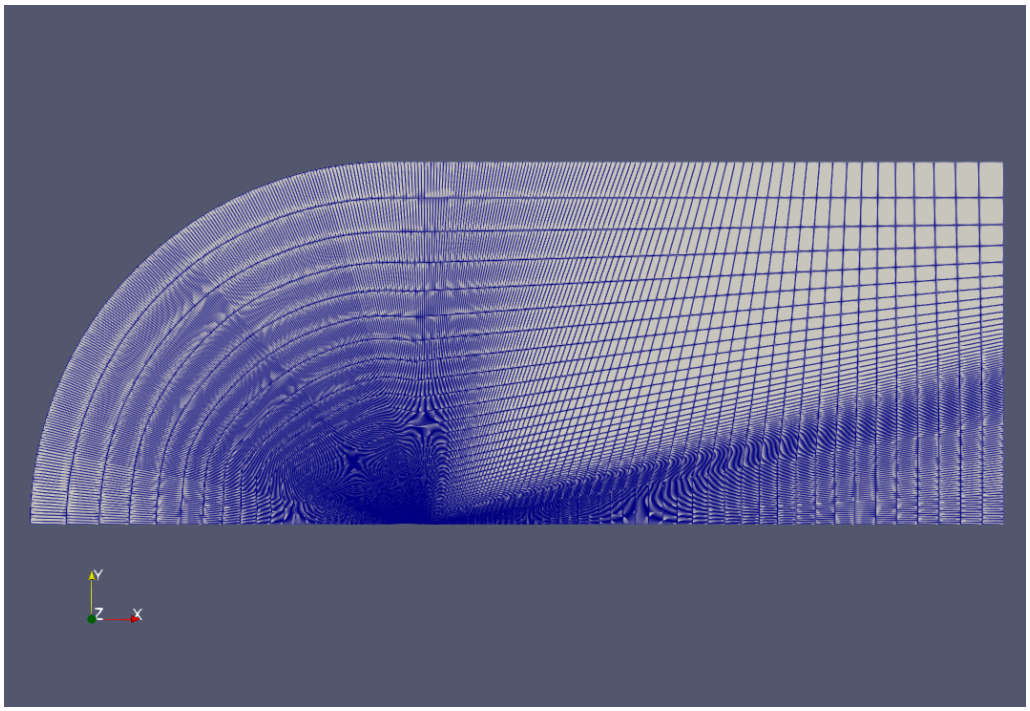


Figure 10.19: *BLI 3D full mesh*

As the figure above shows, the Farfield is far from the body, therefore there is no risk of affecting the results in the central part where the relevant geometry is present.
The figures below show two enlargements of the mesh created in two particularly critical zones for this geometry. The first complex zone concerning the creation of the mesh is the one where the engine is present; here one must be able to accurately grasp the presence of the boundary layer that is fundamental for a simulation with BLI. Another point where the geometry under consideration was more complex is that of the aircraft nose, which can be seen in figure 10.21

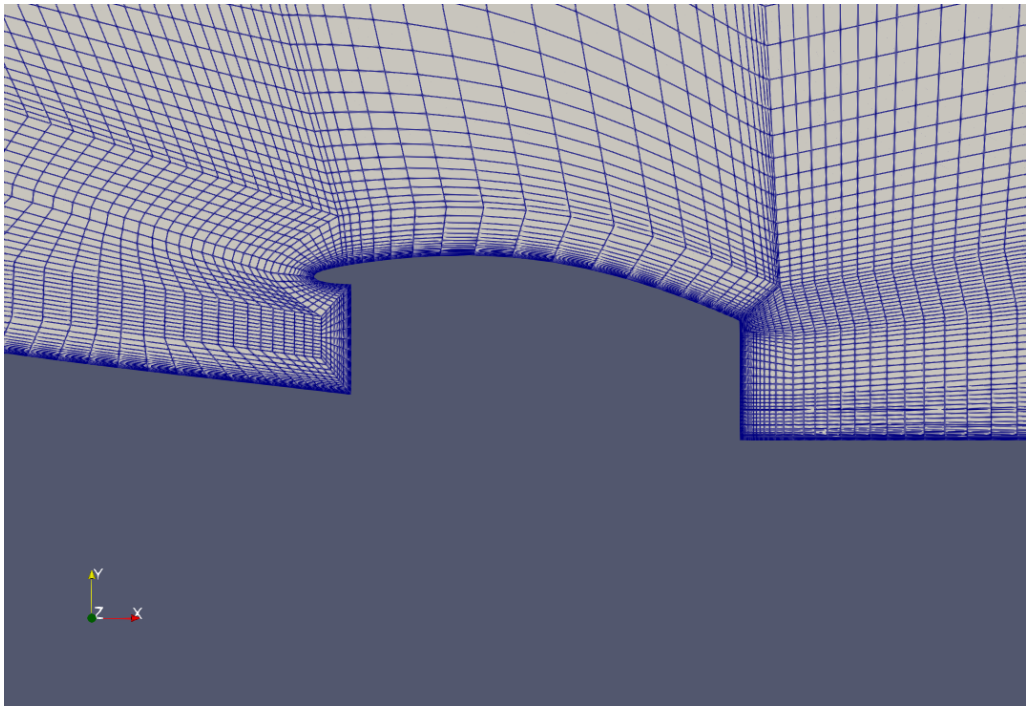


Figure 10.20: *BLI 3D mesh for the engine part*

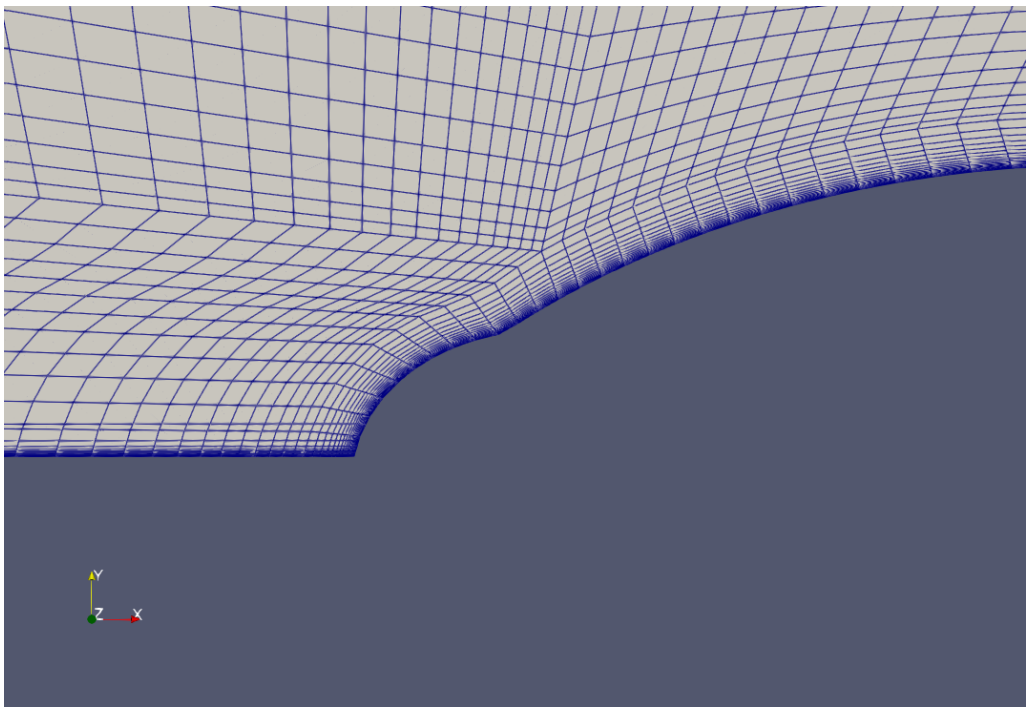


Figure 10.21: *BLI 3D mesh for the nose of the aeroplane*

The table below shows the values given to the boundary conditions supplied to ICEM using a configuration file. These values were obtained by using the Thermo-Data module implemented in CEASIOMpy, using a turbojet as the engine type.

Boundary cond.	**Temperature[K]**	**Pressure[Pa]**	**Velocity vector**
Farfield	288	101000	1,0,0
Inlet	Valore 10	Valore 11	1,0,0
Outlet	Valore 14	Valore 15	1,0,0
Surface		No-slip condition	

Table 3: *Boundary conditions*

The simulation results are then shown in the figures below, with particular attention to the Mach number at the motor output.

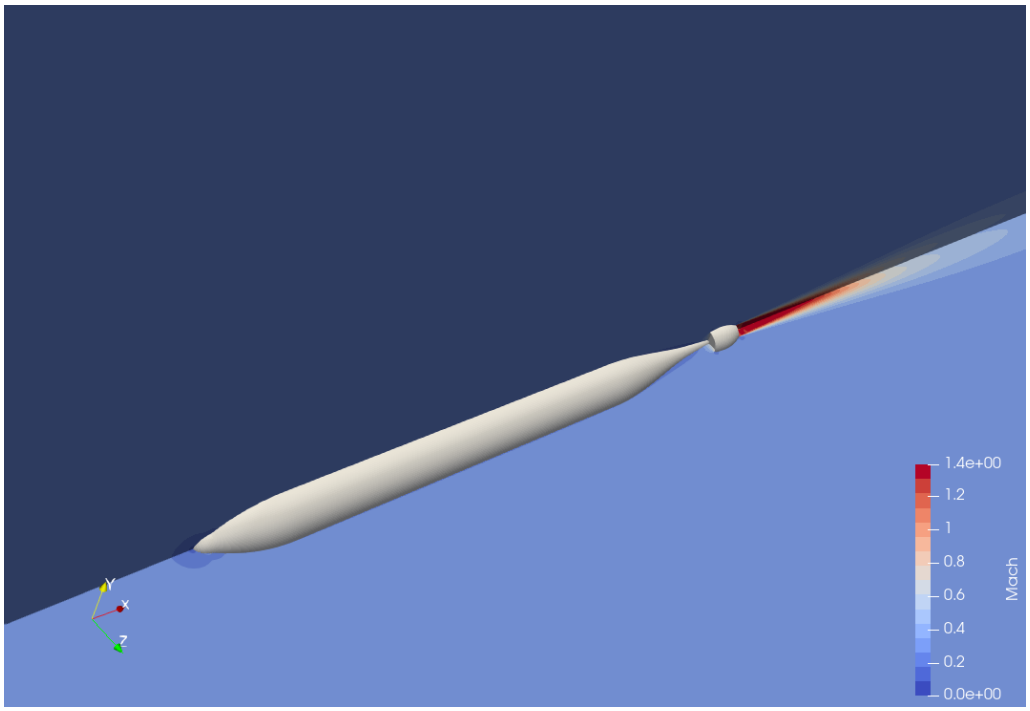


Figure 10.22: *BLI 3D Mach distribution*

In the enlargement below you can clearly see how the flow is accelerated at the motor outlet.

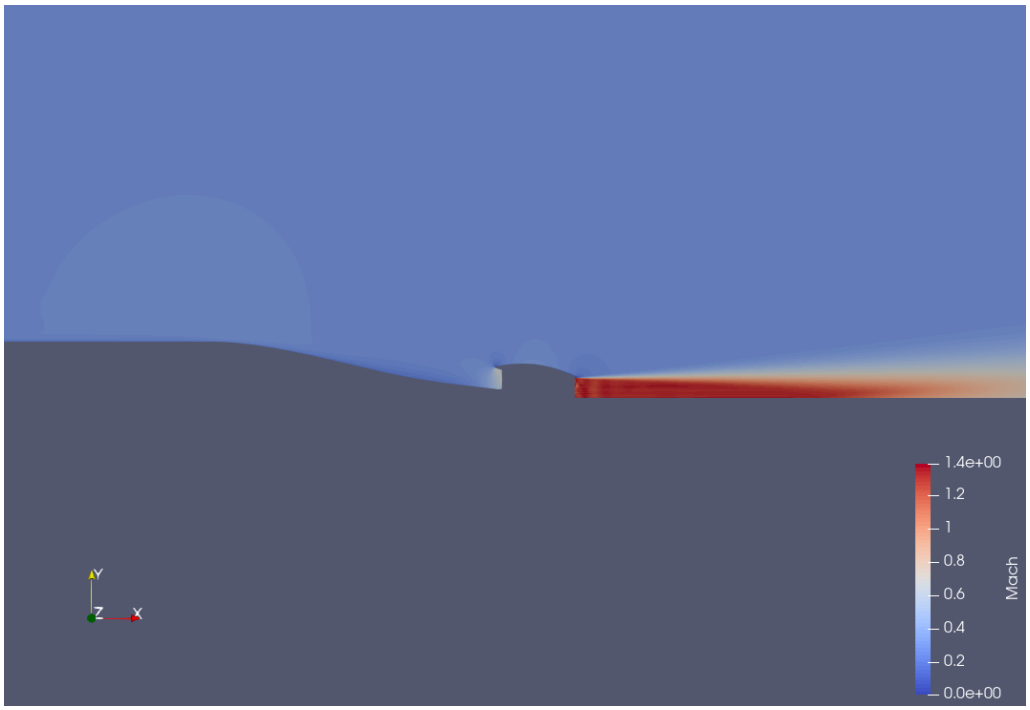


Figure 10.23: *BLI 3D Mach distribution zoom for the engine part*

The last two figures show, respectively, the number of Mach number plotted on a line perpendicular to the fuselage in the middle of the fuselage and the trend of the residuals. As can be seen in figure 10.24, the zeroing of the Mach number at the fuselage (no-slip condition) was well captured with this mesh. While in figure 10.25 we can see how good convergence is achieved despite having a rather complex geometry and boundary conditions.

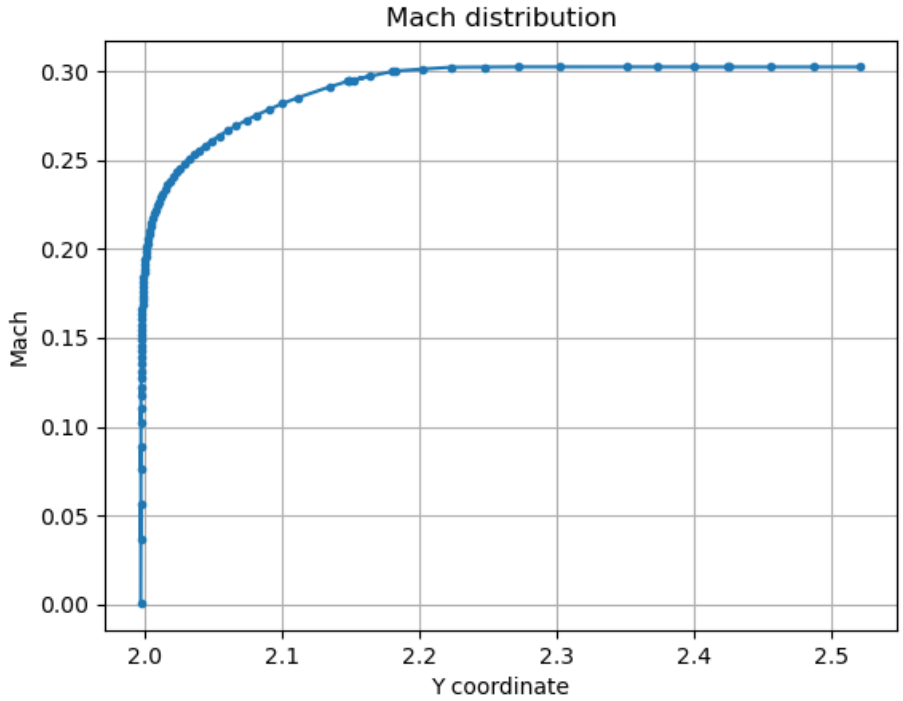


Figure 10.24: *BLI 3D mach distribution on a line perpendicular to the fuselage*

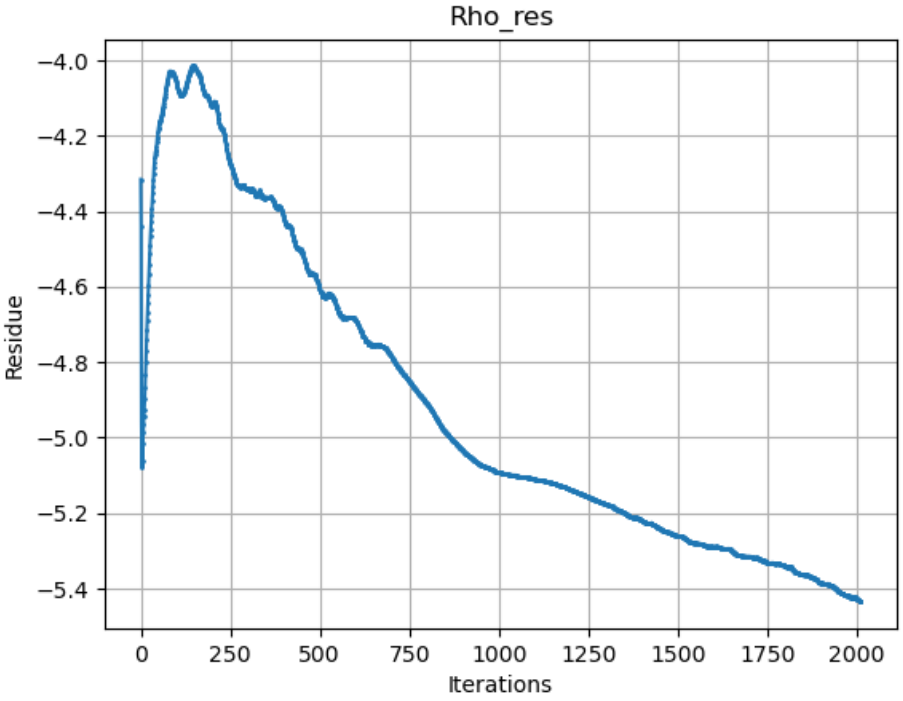


Figure 10.25: *BLI 3D residues*

Unfortunately, we were not able to carry out the same simulation using the automatic hybrid meshing RANS system implemented in CEASIOMpy. The geometry was too complex near the engine, preventing convergence of the simulation due to a mesh of not good enough quality. The capability of this automatic meshing system will need to be improved in future developments.

11 Conclusions

The thesis successfully integrated the PyCycle code into the CEASIOMpy software, enabling the calculation of boundary conditions at the aircraft engine outlet. This integration allows for a more comprehensive simulation that combines both the aerodynamic and thermodynamic aspects of aircraft design.

Furthermore, the thesis incorporated a functionality within CEASIOMpy for the automatic generation of hybrid meshes, which are essential for performing RANS calculations. This was achieved through the combination of three open-source codes: Gmsh, Pentagrow, and TetGen. Hybrid mesh generation allows for the resolution of the boundary layer, which is crucial for accurate CFD simulations.

The implementation in CEASIOMpy was validated using computational fluid dynamics (CFD) simulations of various configurations, including LabAR, J28, and a BLI configuration. These simulations demonstrated the effectiveness of the new meshing method and the implementation of the RANS simulation, showing results comparable to existing Euler simulations and capturing viscous interactions near the aircraft surface. The possibility to simulate BLI configurations is a significant advancement, as this technology can lead to improved fuel efficiency and reduced emissions.

Key achievements of this thesis include:

- **Integration of Pycycle**: The successful implementation of the PyCycle code within CEASIOMpy allows for a more complete analysis of aircraft engines, including the thermodynamic parameters
- **Hybrid Mesh Generation** The development of an automatic hybrid mesh generation process using Gmsh, Pentagrow, and TetGen, enables RANS calculations, accurately capturing the boundary layer and no-slip conditions.
- **RANS Simulation Capability** The addition of the ability to perform RANS simulations, which is a critical step for a more accurate aerodynamic analysis by resolving viscous effects.

The work provides a valuable contribution to CEASIOMpy software, expanding its capabilities for aircraft design and analysis. The implemented modules can be used for further research and development, especially in the field of boundary layer ingestion, and for a more comprehensive analysis of the complete aircraft system. The thesis lays the groundwork for several potential future developments, which can be categorized into improvements in software capabilities, expansion of analysis scope, and further research areas.

Specific Areas for Future Development:

- **Enhanced Engine Modeling:** The current implementation of the ThermoData module in CEASIOMpy includes only two engine types: turbofan and turbojet. A key area for improvement is to expand the library of engine types, including presets for different engine configurations and allowing users to modify individual component parameters such as efficiencies and constraints.
- **Improved Convergence:** The turbofan simulations currently require more time to converge compared to the turbojet simulations. To improve this, the initial values used by the software could be differentiated based on input conditions, leading to faster convergence.
- **Automation of RANS Simulations:** The RANS simulation setup could be improved by automating more parameters. A critical parameter to be automatically calculated is the CFL number, which is currently manually set by the user and should be automatically calculated from the mesh to improve convergence.
- **Integration of More Complex Geometries:** Further simulations should include increasingly complex geometries to validate the results and ensure the potentiality of the new implementation.
- **Advanced Meshing Techniques:** The hybrid meshing process could be further refined by adding more control over parameters or refining the quality of the mesh. The automatic generation of a mesh with a structured boundary layer and unstructured elements outside of it is a key feature of this work that can be improved for future works.
- **Experimental Validation:** The results of the simulations could be validated with experimental data from wind tunnel tests. This would help to ensure the accuracy of the results and also make sure that the implementation is working correctly.
- **User Interface Enhancements:** The user interface could be improved to allow easier input of the parameters by the user, and also with real-time visualization of the mesh to check the accuracy before performing the calculation.

References

[1] Suvanjan Bhattacharyya, John P. Abraham, Lijing Cheng, John Gorman, 2021, *Applications of Computational Fluid Dynamics Simulation and Modeling*

[2] H. Schlichting, 1979, *Boundary Layer theory*, McGraw-Hill Book Company, seventh edition.

[3] https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/airsim.html.

[4] John D. Anderson Jr., 2017, *Fundamentals of Aerodynamics*, McGraw-Hill Education, sixth edition.

[5] Frank M. White, 2016, *Fluid mechanics* McGraw-Hill, 8th edition

[6] R.H. Barnard, D. R. Philpott, 2010, *Aircraft Flight A description of the physical principles of aircraft flight*, Pearson Education Limited, fourth edition.

[7] Doug McLean, 2013, *Understanding aerodynamics: arguing from the real physics.*

[8] Angelique Plas, 2006, *Performance of a Boundary Layer Ingesting Propulsion System*, master thesis.

[9] David K. Hall, Arthur C. Huang, Alejandra Uranga, Edward M. Greitzer, Mark Drela, Sho Sato, 2020, *Boundary Layer Ingestion Propulsion Benefit for Transport Aircraft.*

[10] Nils Budziszewski, Jens Friedrich, 2018, *Modelling of A Boundary Layer Ingesting Propulsor.*

[11] Anil Yildirim, Justin S. Gray, Charles A. Mader, Joaquim R. R. A. Martins, 2022, *Boundary-Layer Ingestion Benefit for the STARC-ABL Concept.*

[12] Alejandra Uranga, Mark Drela, Edward M. Greitzer, David K. Hall, Neil A. Titchener, Michael K. Lieu, Nina M. Siu, Cécile Casses, Arthur C. Huang, Gregory M. Gatlin, Judith A. Hannon, 2017, *Boundary Layer Ingestion Benefit of the D8 Transport Aircraft.*

[13] Annalisa Tresoldi, 2020, *CFD Modeling and Analysis of a Boundary Layer Ingesting S-shaped Intake*, master thesis.

[14] Brian Yutko, Neil Titchener, Christopher Courtin, Michael Lieu, Larry Wirsing, David Hall, John Tylko, Jeffrey Chambers, Thomas Roberts, Clint Church, 2018, *Design and development of the D8 commercial and transport concept*

[15] Ronald T. Kawai, Douglas M. Friedman, Leonel Serrano, 2006, *Blended Wing Body (BWB) Boundary Layer Ingestion (BLI) Inlet Configuration and System Studies.*

[16] Eric S. Hendricks, Justin S. Gray, 2019, *pyCycle: A Tool for Efficient Optimization of Gas Turbine Engine Cycles*

[17] Gray J.S., Hwang J.T., Martins J.R.R.A., Moore K.T., Naylor B.A., 2019, *OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization.*

[18] "CPACS, Common Parametric Aircraft Configuration Schema" `https://www.cpacs.de/`

[19] "Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts" `https://www.agile-project.eu/project/`

[20] "Engineering and CFD" `https://www.gauss-centre.eu/results/computational-and-scientific-engineering/dns-of-airfoil-acoustics`

[21] Gray J.S., 2018, *Design Optimization of a Boundary Layer Ingestion Propulsor Using a Coupled Aeropropulsive Model*

[22] Leroy H. Smith Jr., 1993, *Wake Ingestion Propulsion Benefit*, Journal of propulsion and power, Vol. 9, No. 1

[23] Greitzer, E., Bonnefoy, P., la Rosa Blanco, E. D., Dorbian, C., Drela, M., Hall, D., Hansman, R., Hileman, J., Liebeck, R., Lovegren, J., Mody, P., Pertuze, J., Sato, S., Spakovszky, Z., Tan, C., Hollman, J., Duda, J., Fitzgerald, N., Houghton, J., Kerrebrock, J., Kiwada, G., Kordonowy, D., Parrish, J., Tylko, J., Wen, E., and Lord, W., 2010, *N+3 Aircraft Concept Designs and Trade Studies, Final Report*

[24] F. Aurenhammer. Voronoi diagrams, 1991, *"A study of fundamental geometric data structure"*. ACM Comput. Surveys, 23:345–405.

[25] J.-D. Boissonnat, O. Devillers, and S. Hornus, 2009, *"Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension"*, In Proc. 25th Annual Symposium on Computational Geometry.

[26] A. Bowyer, 1987, *"Computing Dirichlet tessellations"*, Comp. Journal, 24(2):162–166.

[27] B. Chazelle, 1984, *" Convex partition of polyhedra: a lower bound and worst case optimal algorithm"*, SIAM Journal on Computing, 13(3):488–507.

[28] J. A. de Loera, J. Rambau, and F. Santos, 2010, *"Triangulations, Structures for Algorithms and Applications, volume 25 of Algorithms and Computation in Mathematics."*, Springer Verlag Berlin Heidelburg, 1 edition.

[29] B. N. Delaunay, 1934, *"Sur la sphère vide"*, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 7:793–800.

[30] H. Edelsbrunner, 2001, *"Geometry and topology for mesh generation"*, Cambridge University Press, England.

[31] H. Edelsbrunner and M.P. Mucke, 1990, *" Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithm."*,ACM Transactions on Graphics, 9(1):66–104.

[32] H. Edelsbrunner and N. R. Shah, 1934, *"Incremental topological flipping works for regular triangulations."*,s. Algorithmica, 15:223–241.

[33] D. T. Lee and A. K. Lin, 1986, *"Generalized Delaunay triangulations for planar graphs. "*,Discrete and Computational Geometry, 1:201–217.

[34] G. L. Miller, D. Talmor, S.-H. Teng, N. J. Walkington, and H. Wang, 1996, *"Control volume meshes using sphere packing: Generation, refinement and coarsening."*, In Proc. 5th Intl. Meshing Roundtable.

[35] J. Ruppert, 1995, *"A Delaunay refinement algorithm for quality 2-dimensional mesh generation. "*,Journal of Algorithms, 18(3):548–585.

[36] E. Schonhardt, 1928, *"Uber die zerlegung von dreieckspolyedern in tetraeder. "*,Mathematische Annalen, 98:309–312.

[37] V. T. Rajan, 1994, *"Optimality of the Delaunay triangulation in* $\mathbb{R}^d$ *."*,Discrete and Computational Geometry, 12:189–202.

[38] J. R. Shewchuk, 1997, *Adaptive precision floating-point arithmetic and fast robust geometric predicates. "*,Discrete and Computational Geometry, 18:305–363.

[39] J. R. Shewchuk, 1998, *A condition guaranteeing the existence of higherdimensional constrained Delaunay triangulations. "*,s. In Proc. 14th Ann. Symp. on Comput. Geom., pages 76–85.

[40] J. R. Shewchuk, 2002, *"Constrained Delaunay tetrahedralizations and provably good boundary recovery. "*, In Proc. 11th International Meshing Roundtable, pages 193–204. Sandia National Laboratories.

[41] 20 J. R. Shewchuk, 2003, *"Updating and constructing constrained Delaunay and constrained regular triangulations by flips. "*, In Proc. 19th Ann. Symp. on Comput. Geom., pages 86–95.

[42] J. R. Shewchuk, 2008, *"General-dimensional constrained Delaunay and constrained regular triangulations, i: combinatorial properties."*, Discrete and Computational Geometry, 39:580–637.

[43] H. Si, 2003, *"Adaptive tetrahedral mesh generation by constrained delaunay refinement. "*, International Journal for Numerical Methods in Engineering, 75(7):856–880.

[44] 23 H. Si, 2008, *"Three dimensional boundary conforming Delaunay mesh generation."*, PhD thesis, Institut fur Mathematik, Technische Universitat Berlin, Strasse des 17. Juni 136, D-10623, Berlin, Germany.

[45] H. Si and K Gaertner, 2011, *" 3d boundary recovery by constrained Delaunay tetrahedralization. "*,International Journal for Numerical Methods in Engineering, 85:1341–1364.

[46] H. Si and K. Gartner, 2005, *" Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. "*,In Proc. 14th International Meshing Rountable, pages 147–163.

[47] G. Voronoi, 1907, *"Nouvelles applications des parametrès continusà la théorie de formas quadratiques. "*,Reine Angew. Math, 133:97–178.

[48] D. F. Watson, 1907, *" Computing the n-dimensional Delaunay tessellations with application to Voronoi polytopes."*,Comput. Journal, 24(2):167– 172.

[49] G. M. Ziegler, 1907, *"Lectures on Polytopes, volume 152 of Graduate Texts in Mathematics."*,Springer-Verlag, New York, second edition edition.

[50] C. Geuzaine and J.-F. Remacle,2009, *"Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities."*, International Journal for Numerical Methods in Engineering, Volume 79, Issue 11, pages 1309-1331.