MEng Final Year Project

Imperial College London

Department of Aeronautics

Aerodynamic Shape Optimization of Wings with Compliant Winglets

Author:

Jopaul Jobi

CID:

02023052

Supervisor:

Prof. Rafael Palacios

Second Marker:

Prof. Robert Hewson

02/06/2025

Abstract

This project presents a fully open-source workflow for aerodynamic shape optimization of wings and winglets using Gmsh and SU2. A parametric study was first conducted on a family of wing–winglet configurations generated using OpenVSP, with sweep and cant angles varied to analyze their effects on lift and drag. Results from inviscid simulations highlighted expected aerodynamic trends with lift but also revealed challenges such as negative drag coefficients, attributed to mesh quality limitations in Gmsh. To explore optimization capability, SU2's adjoint solver was applied to transonic 2D airfoil cases, achieving up to 74% drag reduction while maintaining specified lift. These results demonstrated the solver's ability to reshape airfoils in line with transonic aerodynamic theory. Although initial attempts at 3D wing optimization were hindered by boundary layer meshing issues and HPC instability, the project successfully established a modular and reproducible workflow. Runtime performance was also assessed, showing over a $3\times$ speedup with MPI and up to 36% reduction using multigrid methods. Overall, this work illustrates the potential and current limitations of using open-source tools for high-fidelity aerodynamic optimization, laying the groundwork for further development in accessible and scalable design pipelines.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Rafael Palacios, for his invaluable support and guidance throughout this project. I also wish to thank my friends and family for their unwavering encouragement over the years at Imperial.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The design of aerodynamic components is crucial for the implementation of the Net Zero strategy outlined by the UK government [1]. In order to be carbon neutral by 2050, significant reductions in greenhouse gas emissions are required in all sectors, but especially in transport. It is estimated that 113.2 MtCO2e (million tonnes of carbon dioxide equivalent) came from the transport sector, with 29.6 MtCO2e coming from domestic and international aviation in 2022 [2]. Leveraging optimized aerodynamic geometries is a significant step toward reducing these emissions by allowing for more sustainable and cost-effective vehicles for transportation.

Within the aviation industry, wings and winglets are particularly critical due to their direct impact on lift-to-drag ratio and fuel efficiency. The design of winglets, originally inspired by the wingtip feather adjustments of birds, helps to reduce the lift-induced drag on an airplane by reducing the strength of wingtip vortices and their associated downwash on the wing [3]. Modern winglets are estimated to reduce fuel burn and associated CO2 emissions by up to 6 percent over the full flight envelope [4][5].

Optimizing components such as winglets usually requires advanced computational tools which have traditionally only been available with expensive proprietary software. This usually limits accessibility and reproducibility for research and development purposes. However, open-source software aims to address this problem as it allows the user to use, study, and change the software and its source code for any purpose [6][7]. This allows users to bypass constraints such as limited licenses, enabling faster development cycles and customisation of workflows and setups, and significantly reduces costs. In recent years, high-performance open-source tools have been released and updated to support complex aerodynamic design, offering advanced insights and analysis capabilities comparable to those of commercial software.

This thesis addresses these challenges by investigating a fully automated workflow for the aerodynamic shape optimization of wings and winglets, built entirely on open-source tools. The primary simulation engine used in this work is SU2 (Stanford University Unstructured), an open-source computational fluid dynamics (CFD) software package that offers adjoint-based optimization capabilities and a rich set of tools for aerodynamic analysis [8]. SU2 requires a computational mesh as input, which in this work will be generated using the open-source meshing tool Gmsh. This work develops a modular and reproducible framework that integrates geometry parametrization, mesh generation, CFD simulation, and optimization—all using open-source components.

1.2 Aims and Objectives

The main goal of this project is to demonstrate that open-source software can offer a competitive, transparent, and extensible alternative to commercial solutions for high-fidelity aerodynamic optimization, while also highlighting its current limitations. By focusing on wing and winglet configurations, this work aims to contribute toward the broader objective of lowering emissions in the transport sector. Through the integration of SU2 and Gmsh, the project showcases the potential for open-source tools to deliver advanced aerodynamic analysis and optimization workflows that are not only cost-effective but also accessible and reproducible within academic and industrial settings. The primary aims of this project can be broken down and summarised as follows:

1. Evaluate the capabilities of Gmsh in generating high-quality meshes for complex wing-winglet 3D geometries and their compatibility with SU2 simulations.
2. Investigate the performance and feasibility of viscous aerodynamic shape optimization in SU2 using meshes generated from Gmsh.
3. Explore strategies for reducing simulation time in Gmsh and SU2 through optimization of solver settings and computational efficiency.

2 Literature Review

To establish the foundation for this project, a review of the relevant aerodynamic principles and optimization techniques was conducted. This section outlines the theoretical background and prior work that has implemented similar optimization frameworks using SU2 to highlight current methodologies and opportunities for improvement that this project seeks to address.

2.1 Lift-Induced Drag

Finite-span wings generate lift through a pressure difference between the upper and lower surfaces. This imbalance causes air to flow spanwise near the wingtips, forming strong wingtip trailing vortices, as illustrated in Figure 1. These vortices induce a downward airflow called downwash, which reduces the effective angle of attack and tilts the lift vector rearward. The rearward component of lift is known as lift-induced drag, which can contribute up to 25% of total drag at cruise [9].

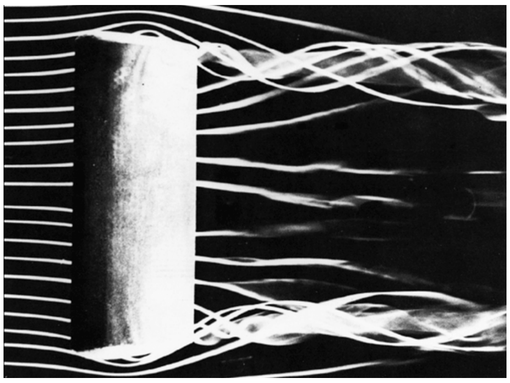


Figure 1. Wing-tip vortices for a finite span wing [9]

The induced drag coefficient can be expressed in terms of the lift coefficient, Oswald efficiency e, and aspect ratio AR. Therefore, one way to reduce lift-induced drag is by increasing the wing's aspect ratio, as shown in equation 1 below.

$$C_{D_i} = \frac{C_L^2}{\pi e AR} \tag{1}$$

However, longer wings increase structural loads and the total weight of the aircraft. Furthermore, wingspan affects an aircraft's ICAO Reference Code, with larger spans assigned higher code letters (D, E, F) [10]. This classification impacts airport compatibility as larger wings may restrict gate availability and increase parking or infrastructure costs.

2.2 Winglets

Winglets offer a more compact alternative by redirecting wingtip vortices, reducing their strength and the associated downwash, as illustrated in Figure 2. In well-designed cases, winglets can also generate a small forward thrust component, improving overall aerodynamic efficiency and decreasing fuel consumption, which in turn lowers environmental impact.

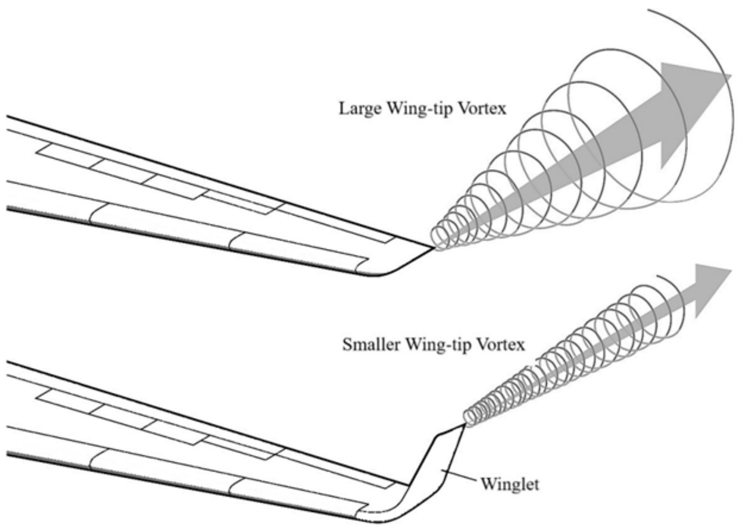


Figure 2. Winglet flow effects [11]

Winglets were initially developed by NASA engineer Richard T. Whitcomb in the 1970s as part of efforts to improve fuel efficiency. His research demonstrated that winglets could increase fuel savings by around 6–7% [12]. Following successful tests, winglets were adopted by major manufacturers, notably appearing on Boeing aircraft such as the 747-400 and later on models like the 737 and 767 [13] with advanced blended designs, as well as Airbus, which developed its own equivalent, known as sharklets.

Today, winglets are standard features on many modern aircraft, and their development reflects decades of aerodynamic research, experimental validation, and engineering refinement aimed at making air travel more efficient and sustainable.

2.3 Aerodynamic Shape optimization

Optimization is a mathematical process used to find the best possible solution to a problem within a defined set of constraints and objectives, it involves adjusting input variables to minimise an objective function. A general optimization problem can be formulated as:

$$\min_{\mathbf{x}} \quad J(\mathbf{x}) \tag{2}$$

$$\text{subject to} \quad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \tag{3}$$

$$h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, p \tag{4}$$

Here, x is the vector of design variables, $J(x)$ is the objective function to be minimised, $g_i(x)$ are inequality constraints, and $h_j(x)$ are equality constraints. For example, the objective function might represent drag to be minimised, with the design variable being the shape, while the constraints could enforce a constant lift coefficient and a minimum airfoil thickness.

Aerodynamic shape optimization (ASO) is the process of refining the geometry of an object to improve its performance in a fluid environment [14]. The overall design process, illustrated in Figure 3, follows an iterative framework in which CFD is used to evaluate the aerodynamic performance of a given design. Based on these results, an optimization algorithm determines how the design variables should be adjusted within the design space to improve a specific objective in the subsequent iteration. SU2 uses a gradient-based approach to guide the optimization. Specifically, its adjoint solver enables efficient calculation of these gradients by solving an additional set of adjoint equations, which relate the sensitivity of the objective function to perturbations in the flow field, allowing all gradients with respect to the design variables to be obtained from a single adjoint solution.

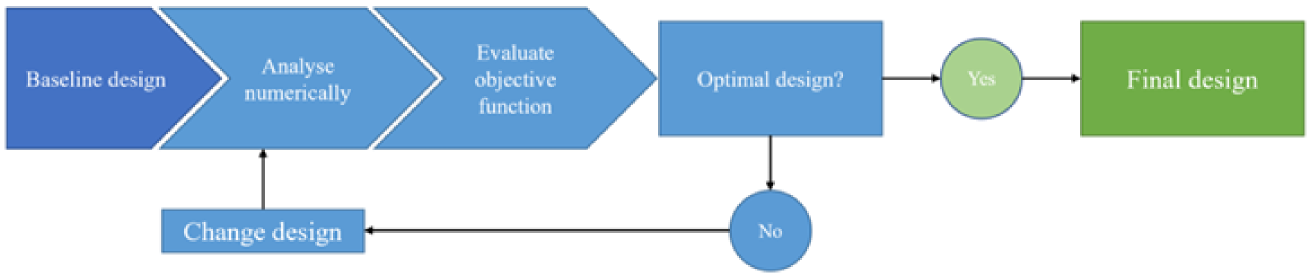


Figure 3. Numerical optimization process [15]

2.4 Open-Source Software

Open-source software is favored in scientific research due to its increased transparency, which allows for result validation and promotes greater trust and acceptance within the scientific community [16]. There are some challenges with open-source, such as a steep learning curve with the specific scripting and configurations used, as well as stability and maintenance concerns [17]. However, it also offers many benefits, such as attracting external contributors who can bring new perspectives and problem-solving approaches to the project. Furthermore, there is also a major cost benefit as it was estimated that the value of open-source software to businesses was $8.8 trillion in 2024, since firms would need to spend 3.5 times more without access to open-source solutions [18].

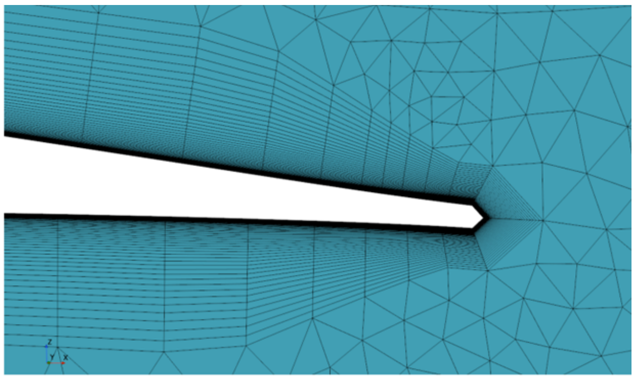
2.5 Previous Work

The process of automating aerodynamic shape optimization using SU2 has been explored with varying methodologies, notably by Herrera [19], King [20], and Loong [21].

Herrera and King used OpenVSP for the geometry generation stage while Loong utilised the Common Parametric Aircraft Configuration Schema (CPACS) [22]. Both of these software tools are open source, with the former being preferred due to its simplicity in defining parametric geometries.

All three used STAR-CCM+ for the meshing step before submitting the simulations to SU2. STAR-CCM+ is a commercial CFD tool developed by Siemens [23], and can be used for generating meshes for a wide range of geometries. Issues with cell skewness and deformed mesh cells were identified during the process, as illustrated in Figure 4. Furthermore, because STAR-CCM+ is a licensed commercial software, only one license was available per user, so the option to run multiple simulations with different configurations in parallel was not available.

Herrera was the only one who explored ASO, conducting both 1D and 2D optimizations on a winglet geometry under subsonic and inviscid flow conditions. While the results showed improved aerodynamic performance with reduced drag, it was identified that the use of a viscous solver would provide a more realistic representation of flow behavior and better capture practical effects for optimization outcomes.


(a) Trailing edge cell skewness [19]

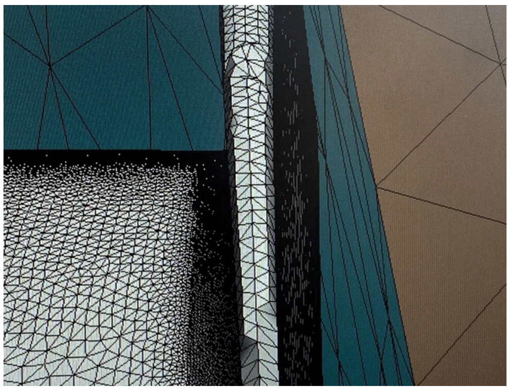

(b) Winglet mesh deformation [20]

Figure 4. Previous meshing issues

3 Methodology

3.1 Overview

Building on the motivation and previous research, this project developed and implemented a fully open-source workflow for aerodynamic shape optimization, with a focus on wings and winglets. The methodology integrates key stages of the aerodynamic design process, and the full workflow diagram is presented in Figure 5. The goal is to evaluate the viability, performance, and limitations of such a workflow when compared to traditional commercial tools. Each component of the pipeline is configured to run on the Imperial High Performance Computing (HPC) cluster, with bash scripts used for automation. The following sections describe the workflow in detail, including the tools used, parameter settings, mesh strategies, and solver configurations.

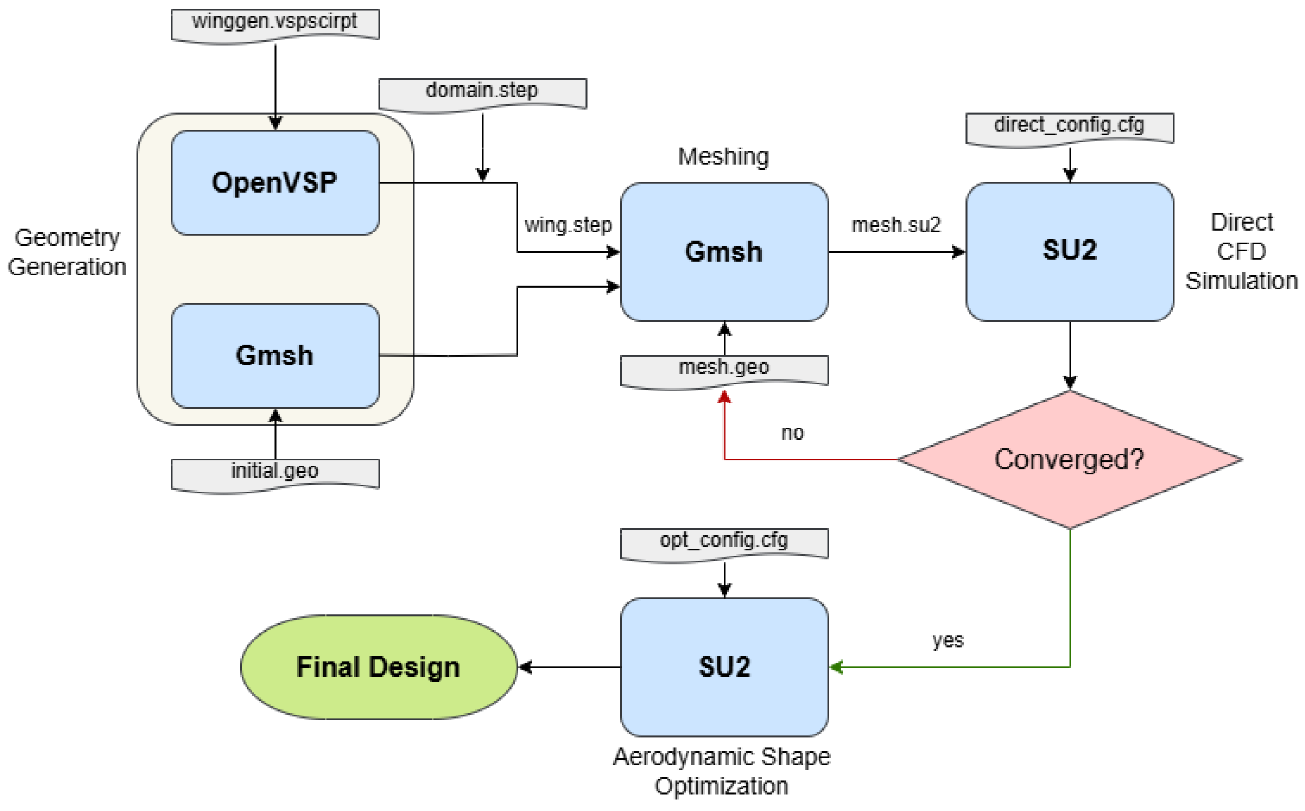


Figure 5. Workflow of project

3.2 Initial Geometry Generation

Geometry generation in this project was carried out using both OpenVSP and Gmsh. OpenVSP was used to create more complex 3D wing-winglet geometries that were then meshed using Gmsh. These meshes were then used in SU2 simulations and compared against results provided by Jamie, who used STAR-CCM+, to assess how the Gmsh-generated meshes performed relative to those from a commercial meshing tool. Gmsh was used to generate both 2D airfoil and 3D wing geometries, which were then meshed and exported to SU2. These meshes were then subjected to SU2's adjoint-based optimization process, enabling mesh deformation and the creation of aerodynamically enhanced shapes.

3.2.1 OpenVSP

The Open Vehicle Sketch Pad (OpenVSP) is an open-source parametric aircraft geometry modeling tool developed by NASA [24]. It allows users to create 3D aircraft models defined with variables and export them as `.step` files. This software has been extensively studied by Herrera and King; therefore, no further work was required. Instead, the methods they developed were adopted and applied within this project.

For the base wing geometry, the European Flutter Free Flight Envelope eXpansion for eCOnomical Performance (FLEXOP) aircraft was used and is illustrated in Figure 6.

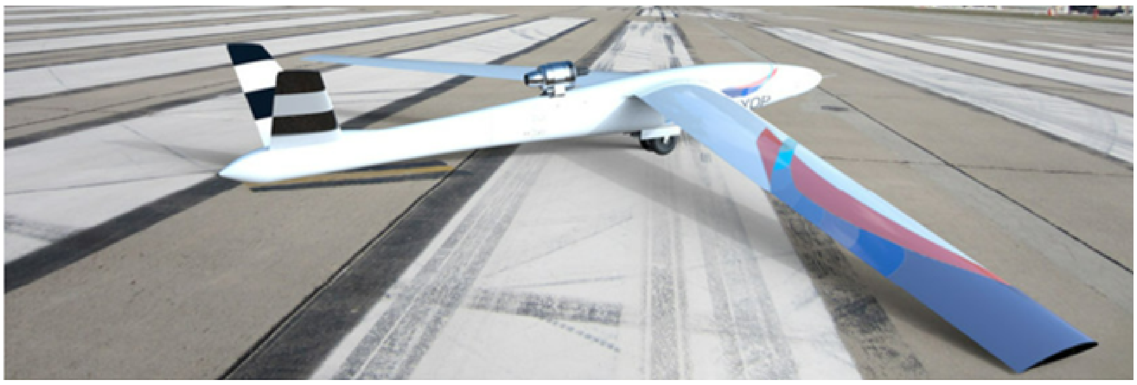


Figure 6. FLEXOP aircraft [25]

Winglets were then added into the wing geometry, incorporating a blended subsection to ensure a smooth transition between the wing and winglet. This process was automated using a `.vspscript`. The key input parameters were the cant, sweep, twist, and toe angles of the winglets. The parametrization of winglets is illustrated in Figure 7, where variations in these parameters allow for the generation of multiple wing–winglet configurations. A summary of all the wing and winglet parameters used is provided in Table 1.

airfoil	$t/c(\%)$	c_{root}(m)	λ_{wing}	b(m)	$A_{LE}(^\circ)$	AR	$b_{winglet}$ (m)	$\lambda_{winglet}$
E387	10	0.471	0.5	3.536	20	20	0.3536	0.2

Table 1. OpenVSP wing-winglet parameters [19]

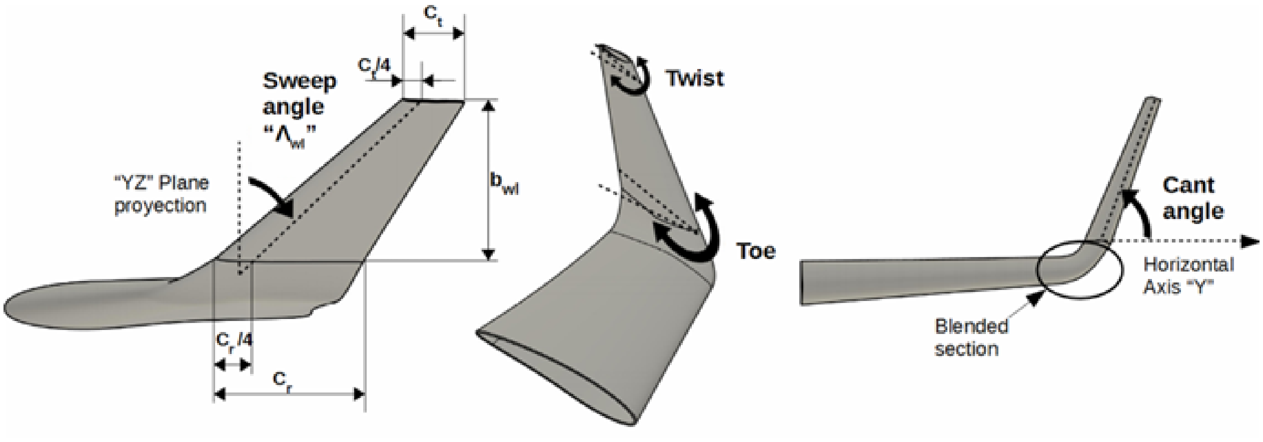


Figure 7. Winglet geometrical parameters [11]

3.2.2 Gmsh Geometry Generation

Gmsh provides a versatile set of geometry generation tools that support both simple and complex modeling tasks through its built-in scripting language and graphical user interface (GUI) [26]. It allows for the creation of points, lines, surfaces, and volumes using parametric definitions, enabling precise control over geometric configurations. The `.geo` scripting format was utilised to automate geometry definition.

NACA 4-digit airfoils were used as the baseline geometry for both 2D and 3D wing configurations. The 2D airfoil profiles were first generated using 150 coordinate points distributed along the surface using cosine spacing. This method concentrates points near the leading and trailing edges, where greater geometric resolution is needed to capture curvature and flow features accurately [27]. The cosine-spaced x-coordinates along the chord of length c are defined using the following equation:

$$x_i = \frac{c}{2}\left(1 - \cos\left(\frac{i\pi}{N-1}\right)\right), \quad \text{for } i = 0, 1, 2, \ldots, N-1 \tag{5}$$

Here, N is the total number of points, i is the index of the point, and c is the chord length; which is always normalised to 1 in this project. The airfoil coordinates were obtained from Airfoil Tools and exported in a standard `.dat` format, which contains the upper and lower surface point data [28]. A Python script was then used to parse this file and convert the coordinates into a Gmsh-compatible `.geo` script by defining each point in Gmsh's syntax. Once the points were defined, they were connected using splines to form the airfoil's closed-loop geometry. For 3D wings, the 2D profile was extruded along the spanwise direction.

3.3 Meshing with Gmsh

Gmsh is a 3D finite element mesh generator and is designed to provide a fast, light, and user-friendly meshing tool with parametric input and flexible visualisation capabilities [29]. While its geometry generation capabilities were discussed earlier, its primary strength lies in its meshing functionality. As mentioned previously, gmsh can be configured through its GUI, its native `.geo` scripting language, as well as APIs compatible with Python, C++, and Julia. For this project, `.geo` scripts were used for the meshing stage.

3.3.1 Meshing Algorithm

A finite element mesh in Gmsh consists of simple geometric elements (lines, triangles, quadrangles, tetrahedra, prisms, hexahedra and pyramids) arranged to form a conformal mesh, where elements intersect only at faces, edges, or nodes. Gmsh supports several automatic meshing algorithms and typically treats meshes as unstructured, meaning elements are defined solely by their node connectivity without any predefined ordering relation [29]. To create a full 3D volume mesh, Gmsh first creates a 1D mesh by dividing curves into segments based on either automatic sizing or user-defined parameters. Users can specify the exact number of points, and spacing between these points, along a curve using the `transfinite` setting.

The main way Gmsh creates a 2D mesh is based on Delaunay triangulation, which states that the unique circle that passes through all three of a triangles' vertices (the circumcircle) must contain no other points from the input set inside its interior [30]. This maximizes the minimum angle of all triangles and helps to avoid silver triangles, which are triangles with extremely acute angles or high aspect ratios which is undesirable in meshes for fluid simulations. Figure 8 illustrates a random set of points. On the left, the points are connected arbitrarily, resulting in several sliver triangles that are unsuitable for fluid simulations due to poor element quality. The arrangement on the right shows the same set of points connected using Delaunay triangulation, which produces a more uniform and well-shaped mesh.

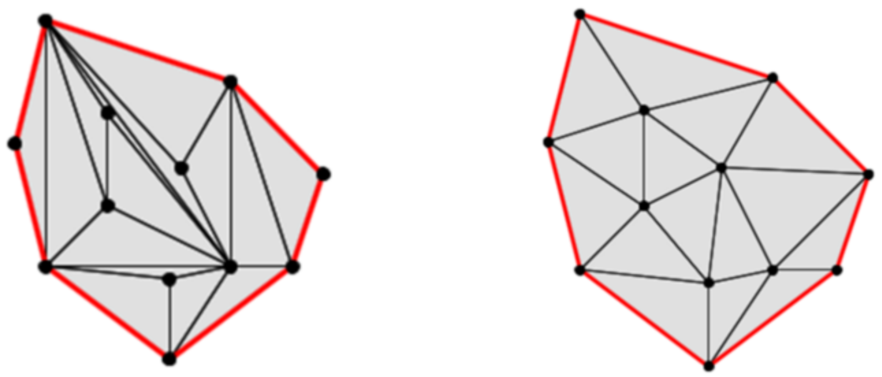
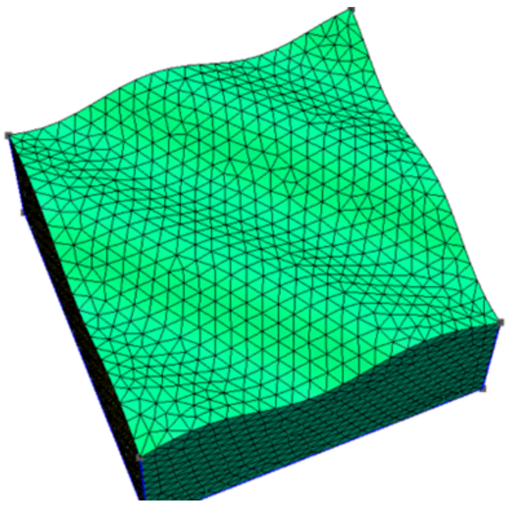


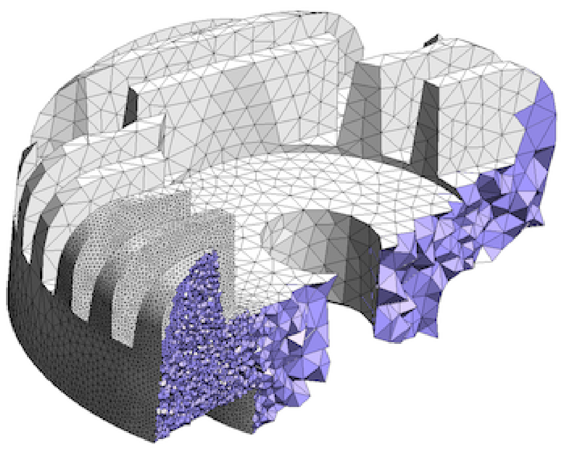
Figure 8. Delaunay triangulation example [31]

Gmsh initially creates a 2D surface mesh by combining the 1D mesh points along the curves and generates an unstructured grid by implementing a divide-and-conquer algorithm based on Dwyer [32] and recovering missing edges using edge swaps [33]. It can then apply one of its 2D meshing algorithms to finalise the mesh. The most appropriate algorithm was the Frontal-Delaunay algorithm [34] which grows the surface mesh from the curves and boundaries inward and places points in a way that maintains the Delaunay property. This creates high-quality, smooth, and graded meshes for curved geometries which makes it well-suited for airfoils and wings. An example of the surface mesh created on a curved surface from this algorithm can be seen in Figure 9a.

A 3D mesh is then generated by filling the enclosed volume with tetrahedral elements. This is done using the Delaunay tetrahedralisation method, which inserts points throughout the volume while respecting the predefined mesh constraints and boundary surfaces. To control mesh density and resolution, Gmsh allows the use of `Field` definitions. These fields enable local mesh refinement in regions of interest such as flow-sensitive regions. Figure 9b shows a volume mesh with finer resolution on the left side of the geometry.



(a) Surface mesh



(b) Ineternal volume mesh

Figure 9. Gmsh examples [29]

3.3.2 Boundary Layer Mesh

Accurate resolution of the boundary layer is essential to capture near-wall effects for viscous simulations, and Gmsh provides a dedicated tool for this through the use of its `BoundaryLayer` field. This allows users to automatically generate highly anisotropic, structured prismatic elements near walls and works by extruding layers of elements normal to the selected curves. It is important to note that this field currently functions only for 2D surface meshes. To create 3D boundary layer meshes, users must manually define extrusions using transfinite lines and surface/volume definitions or by scripting custom layer generation. Both 2D and 3D boundary layer generation require specification of the total layer thickness (δ_{99}), first cell height (y_1), and either the growth rate (r) or the total number of layers (N_{BL}). These parameters are computed using the methodology outlined below.

The Reynolds number was calculated first as:

$$Re = \frac{\rho U_\infty L}{\mu} \tag{6}$$

where ρ is the fluid density, U_∞ is the freestream velocity, μ is the dynamic viscosity, and L is the characteristic length which is either the chord length for an airfoil or the mean aerodynamic chord for a wing. An estimation for turbulent boundary layer thickness is given by using the empirical correlation [35]:

$$\delta_{99} = 0.37 \cdot Re^{-1/5} \tag{7}$$

Two different estimations of the skin friction coefficient were used: Equation 8, valid for Reynolds numbers between 5×10^5 and 10^7, was applied to subsonic flows, while Equation 9, suitable for Re $> 10^9$, was used for transonic conditions [35].

$$C_f = 0.0592 \cdot Re^{-1/5} \tag{8}$$

$$C_f = (2\log_{10}(Re) - 0.65)^{-2.3} \tag{9}$$

The wall shear stress τ_w could then be calculated as:

$$\tau_w = \frac{1}{2} C_f \cdot \rho U_\infty^2 \tag{10}$$

The corresponding friction velocity u_τ is:

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{11}$$

In turbulence modeling, y^+ is a dimensionless measure of the distance from the wall to the first mesh cell, used to assess whether the near-wall region is properly resolved. To ensure accurate capture of the viscous sublayer, the first cell height was calculated to achieve a target y^+ value of 1 [36] and is given by:

$$y_1 = \frac{y^+ \cdot \mu}{\rho u_\tau} \tag{12}$$

Finally, the total number of boundary layer mesh layers N_{BL}, given a total height δ_{99} and a growth rate r, was calculated by:

$$N_{BL} = \frac{\log\left(1 - (1 - r)\,\frac{\delta_{99}}{y_1}\right)}{\log(r)} \tag{13}$$

To maintain a smooth transition through the boundary layer, the mesh growth rate was kept between 1.1 and 1.3, with the total number of layers targeted to be around 30, ensuring adequate resolution from the wall to the outer edge of the boundary layer [37].

3.4 Direct Flow Simulations with SU2

SU2 is an open-source suite of tools for solving partial differential equations (PDEs), with the capacity for tackling CFD, ASO, and other multiphysics problems on general, unstructured meshes [8]. The framework can be used for an arbitrary set of governing equations; however, the main capabilities lie in its Reynolds-averaged Navier-Stokes (RANS) solver capable of simulating the compressible, turbulent flows which are useful for wing-winglet flow analysis. SU2 uses a native mesh format with the `.su2` file extension. Gmsh supports direct export to this format, allowing for the mesh to be used directly as input for SU2 simulations without the need of any intermediate conversions. SU2 also requires a `.cfg` configuration file that defines the simulation parameters, boundary conditions, physical models, numerical settings etc. For this project, SU2 version 8.2.0 'Harrier' was used for all simulations.

3.4.1 Code Structure

The core tools of the SU2 suite are modular C++ executables, each designed for a specific function within the simulation/optimization workflow. By distributing responsibilities across specialised executables, SU2 maintains a flexible and robust framework. A brief description of each of the C++ executables and their use cases is provided below [38]:

- `SU2_CFD`: Solves for steady or unsteady simulations, supporting compressible and incompressible flows and supports Euler, Navier–Stokes, RANS etc.
- `SU2_CFD_AD`: Executes the adjoint flow simulations using automatic differentiation. It is used for sensitivity analysis in shape optimization tasks.
- `SU2_DOT_AD`: Projects the adjoint surface sensitivities obtained from `SU2_CFD_AD` into the design space to compute the gradient.
- `SU2_DEF`: Handles mesh deformation with changes in the design variables during the shape optimization process.
- `SU2_GEO`: Evaluates or constrains a number of geometric properties of interest, such as volumes, section thicknesses, etc.
- `SU2_SOL`: Generates volume and surface solution files into desired format for analysis and visualisation.

3.4.2 Governing Equations

In this project, the aerodynamic behavior of compressible, turbulent flows is modeled using the RANS equations. These equations describe the conservation of mass, momentum, and energy in a flow domain $\Omega \subset \mathbb{R}^3$, bounded by two main surfaces: a far-field boundary Γ_∞ and a solid no-slip wall boundary S, which represents the surface of the aerodynamic body under consideration [39]. The RANS equations, in differential form, can be expressed as:

$$\begin{cases} \mathcal{R}(\boldsymbol{U}) = \dfrac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \vec{F}^c - \nabla \cdot \vec{F}^v - \boldsymbol{Q} = 0 & \text{in } \Omega, \\ \vec{v} = 0 & \text{on } S, \\ \partial_n T = 0 & \text{on } S, \\ (W)_+ = W_\infty & \text{on } \Gamma_\infty, \end{cases} \tag{14}$$

Here, the conserved variables are defined as $\boldsymbol{U} = \{\rho, \rho\vec{v}, \rho E\}^T$, where ρ is the fluid density, $\vec{v} \in \mathbb{R}^3$ is the velocity vector, and E is the total energy per unit mass. The convective and viscous fluxes, along with the source terms, are denoted by $\vec{F}^c$, $\vec{F}^v$, and $\boldsymbol{Q}$, respectively. In the arbitrary Lagrangian–Eulerian (ALE) framework, the convective fluxes are:

$$\vec{F}^c_{\text{ale}} = \begin{Bmatrix} \rho(\vec{v} - \vec{u}_\Omega) \\ \rho\vec{v} \otimes (\vec{v} - \vec{u}_\Omega) + \mathbb{I}p \\ \rho E(\vec{v} - \vec{u}_\Omega) + p\vec{v} \end{Bmatrix}, \tag{15}$$

where $\vec{u}_\Omega$ is the mesh velocity in the ALE context and p is the static pressure. The viscous flux terms are given by:

$$\vec{F}^v = \begin{Bmatrix} \cdot \\ \bar{\bar{\tau}} \\ \bar{\bar{\tau}} \cdot \vec{v} + \mu^*_{\text{tot}} c_p \nabla T \end{Bmatrix}, \tag{16}$$

and $\boldsymbol{Q} = \{q_\rho, \vec{q}_{\rho\vec{v}}, q_{\rho E}\}^T$, with the viscous stress tensor defined as:

$$\bar{\bar{\tau}} = \mu_{\text{tot}} \left(\nabla\vec{v} + \nabla\vec{v}^T - \frac{2}{3}\mathbb{I}(\nabla \cdot \vec{v}) \right). \tag{17}$$

The system assumes a perfect gas, where the pressure is related to the energy and velocity fields by:

$$p = (\gamma - 1)\rho \left(E - \frac{1}{2}\vec{v} \cdot \vec{v} \right), \tag{18}$$

and temperature is determined by:

$$T = \frac{p}{\rho R}, \quad \text{with} \quad c_p = \frac{\gamma R}{\gamma - 1}. \tag{19}$$

To account for turbulence, the Boussinesq approximation is employed [40], modeling turbulent effects through an increased effective viscosity. The total viscosity is thus split into a dynamic component μ_{dyn}, and a turbulent component μ_{tur}. The laminar viscosity is computed using Sutherland's law [41]:

$$\mu_{\text{dyn}}(T) = \mu_0 \left(\frac{T}{T_0} \right)^{3/2} \frac{T_0 + 110.4}{T + 110.4}, \tag{20}$$

and the turbulent viscosity is determined using the Spalart–Allmaras one-equation turbulence model [42]. The total viscosity and its effective counterpart for thermal diffusion are:

$$\mu_{\text{tot}} = \mu_{\text{dyn}} + \mu_{\text{tur}}, \quad \mu^*_{\text{tot}} = \frac{\mu_{\text{dyn}}}{Pr_d} + \frac{\mu_{\text{tur}}}{Pr_t}, \tag{21}$$

where Pr_d and Pr_t are the laminar and turbulent Prandtl numbers, respectively. Note that for inviscid flow simulations, the Euler equations are used by neglecting viscous effects and turbulent contributions. This is achieved by eliminating the viscous flux term $\vec{F}^v$ and setting the source term $\boldsymbol{Q}$ to zero. The resulting governing equation becomes:

$$\mathcal{R}(\boldsymbol{U}) = \frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \vec{F}^c = 0 \tag{22}$$

This formulation assumes the fluid is ideal and non-viscous, meaning no boundary layer formation or shear stress is modeled.

3.4.3 Numerical Implementation

In SU2, both the flow and adjoint problems are solved numerically using unstructured meshes organised via an edge-based data structure [43]. The discretizations follows the *method of lines*, which treats spatial and temporal discretizations separately. This separation provides flexibility in selecting numerical schemes suited to each part of the simulation. Space is discretized using the finite volume method (FVM), while time advancement is handled using the implicit Euler method.

Spatial Discretization

For spatial integration, the finite volume method is applied on a dual mesh constructed with a median-dual, vertex-based approach [8]. Control volumes are formed by connecting cell centroids and edge midpoints around a vertex. After applying the divergence theorem to the integral form of the governing equations, the semi-discrete form becomes:

$$\int_{\Omega_i} \frac{\partial \boldsymbol{U}}{\partial t}\, d\Omega + \sum_{j\in\mathcal{N}(i)} \left(\tilde{\boldsymbol{F}}_{c,ij} + \tilde{\boldsymbol{F}}_{v,ij}\right) \Delta S_{ij} - \boldsymbol{Q}|\Omega_i| = \int_{\Omega_i} \frac{\partial \boldsymbol{U}}{\partial t}\, d\Omega + R_i(\boldsymbol{U}) = 0, \tag{23}$$

Here, $\boldsymbol{U}$ is the state variable vector, $R_i(\boldsymbol{U})$ is the residual from the spatial discretizations, and $\mathcal{N}(i)$ denotes the set of neighbors to node i. The fluxes $\tilde{\boldsymbol{F}}_{c,ij}$ and $\tilde{\boldsymbol{F}}_{v,ij}$ are the convective and viscous fluxes projected along each edge, and ΔS_{ij} is the face area shared between control volumes.

Convective fluxes are evaluated at the midpoints of edges. For this project, the second-order Jameson-Schmidt-Turkel (JST) scheme [44] was used for convective flux computation, as it offers a good compromise between numerical robustness and accuracy. To prevent spurious oscillations and ensure monotonicity, slope limiting is applied using the Venkatakrishnan limiter [8]. For the scalar transport in the Spalart–Allmaras turbulence model, a first-order upwind scheme is used for stability.

To compute viscous fluxes, both the values and gradients of flow variables are needed at the cell faces. These gradients are calculated at each node using a weighted least-squares method [45] and then interpolated to the faces. Source terms are treated using piecewise constant reconstruction within each control volume.

Time Integration

To evolve the system in time, the residual $R_i(\boldsymbol{U})$ is evaluated at the future time step t^{n+1}, implementing an implicit time-stepping scheme. This leads to a linear system for the update $\Delta\boldsymbol{U}_i^n$, given by:

$$\left(\frac{|\Omega_i|}{\Delta t_i^n}\delta_{ij} + \frac{\partial R_i(\boldsymbol{U}^n)}{\partial \boldsymbol{U}_j}\right) \cdot \Delta\boldsymbol{U}_j^n = -R_i(\boldsymbol{U}^n), \tag{24}$$

where $\Delta \boldsymbol{U}_i^n = \boldsymbol{U}_i^{n+1} - \boldsymbol{U}_i^n$. For each flux $\tilde{\boldsymbol{F}}_{ij}$ that depends on a pair of nodes $\{i, j\}$, the corresponding Jacobian contributions appear at four locations in the sparse matrix structure:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} := \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}} + \begin{bmatrix} \ddots & & & & \\ & \frac{\partial \tilde{\boldsymbol{F}}_{ij}}{\partial \boldsymbol{U}_i} & \cdots & \frac{\partial \tilde{\boldsymbol{F}}_{ij}}{\partial \boldsymbol{U}_j} & \\ & \vdots & \ddots & \vdots & \\ & -\frac{\partial \tilde{\boldsymbol{F}}_{ij}}{\partial \boldsymbol{U}_i} & \cdots & -\frac{\partial \tilde{\boldsymbol{F}}_{ij}}{\partial \boldsymbol{U}_j} & \\ & & & & \ddots \end{bmatrix} \tag{25}$$

To solve this linear system, the Generalized Minimal Residual (GMRES) method [39] is employed. GMRES finds an approximate solution in a Krylov subspace by minimising the residual, and it uses the Arnoldi iteration process to construct the orthonormal basis needed for the minimisation.

3.4.4 Convergence Criteria

Since this project was only concerned with steady-state flows, the convergence field was set to drag, meaning convergence was assessed based on the stability of the drag coefficient over iterations. SU2's Cauchy convergence criterion was applied here to monitor convergence and act as the stopping criteria for simulations. The check would only initiate after the first 100 iterations to avoid transient startup fluctuations, and then SU2 tracks the drag coefficient over the last 50 iterations and determines whether the change fell below the specified tolerance of 1E-5. If the variation remained within this threshold, the solution was considered converged.

3.5 Aerodynamic shape optimisation with SU2

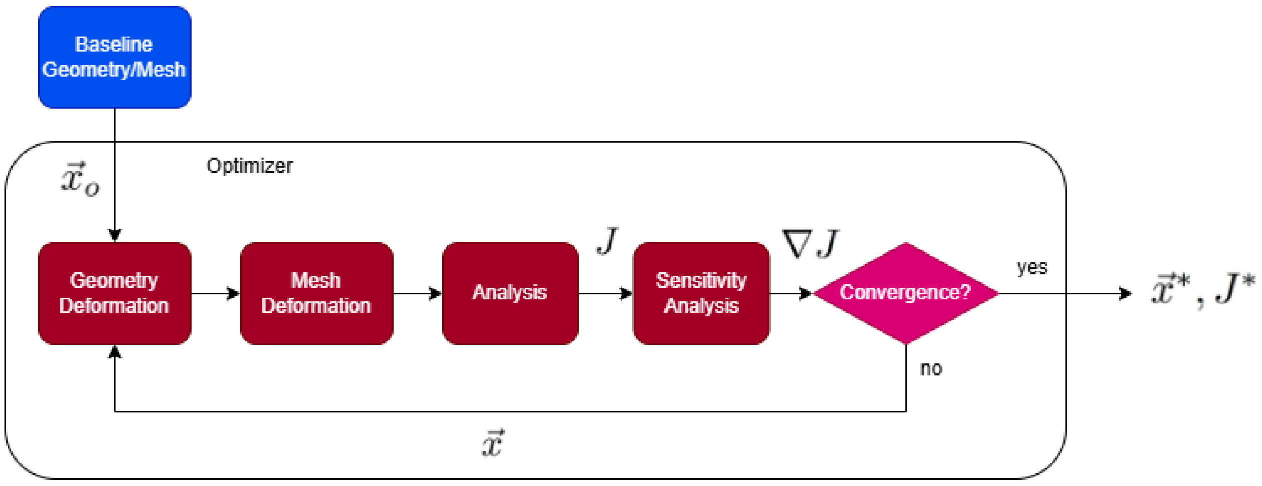


Figure 10. SU2 optimizer workflow [38]

ASO in SU2 was carried out using the `shape_optimization.py` script, which manages the full gradient-based optimization cycle and is illustrated in Figure 10. This iterative process includes the execution of the flow solver, adjoint solver, geometry evaluation, and mesh deformation tools available as described in section 3.4.1. The optimization loop continues until the objective function reaches a minimum or a maximum number of iterations is met. This allows for the computation of sensitivities for a wide range of objective functions, such as drag minimisation, lift maximisation, noise reduction, etc [8]. The framework also supports multiple constraints, including fixed aerodynamic performance metrics (e.g., minimum lift or maximum drag) and geometric limits (e.g., thickness, volume, or curvature bounds). For this project, a gradient-based optimization strategy was employed using the Sequential Least Squares Programming (SLSQP) algorithm provided by the SciPy library [46].

3.5.1 Discrete Adjoint Methodology

The discrete adjoint method is used to compute the gradient of an objective function with respect to design variables in optimization problems governed by PDEs [8]. The constrained optimization problem can be expressed as:

$$\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}, \mathbf{x}) \quad \text{subject to} \quad \mathcal{R}(\boldsymbol{\alpha}, \mathbf{x}) = 0, \tag{26}$$

where J is the objective function, $\boldsymbol{\alpha}$ denotes the design variables, $\mathbf{x}$ is the state vector, and $\mathcal{R}$ represents the residuals of the discretized governing equations. The constraint $\mathcal{R} = 0$ ensures that the primal solution satisfies the underlying flow physics. To avoid the computational cost of directly differentiating the state vector with respect to each design variable, SU2 instead solves the discrete adjoint equation:

$$\left(\frac{\partial \mathcal{R}}{\partial \mathbf{U}}\right)^T \boldsymbol{\Psi} = -\left(\frac{\partial J}{\partial \mathbf{U}}\right)^T, \tag{27}$$

where $\boldsymbol{\Psi}$ is the adjoint variable vector, $\mathbf{U}$ is the flow solution vector, and the Jacobians represent sensitivities of the residual and objective function with respect to the state variables. Once the adjoint variables are determined, the total derivative of the objective with respect to the design variables is calculated as:

$$\frac{dJ}{d\boldsymbol{\alpha}} = \boldsymbol{\Psi}^T \frac{\partial \mathcal{R}}{\partial \boldsymbol{\alpha}} + \frac{\partial J}{\partial \boldsymbol{\alpha}}. \tag{28}$$

In terms of implementation, SU2 constructs the required Jacobian matrix by mimicking the procedure used in the direct flow solver [8]. The residuals are computed by iterating over mesh edges to accumulate flux contributions using an upwind scheme, and then adding appropriate boundary condition terms. For the adjoint problem, this process is repeated in reverse: flux derivatives across internal edges are computed first, followed by contributions from the boundaries. The necessary derivatives of fluxes in both cases are obtained through automatic differentiation.

3.5.2 Design Variable Definitions

In this project, the airfoil geometry was modified using Hicks–Henne bump functions [47]. These functions are commonly used in aerodynamic shape optimization due to their smooth, localised control over surface deformation. Each bump function introduces a smooth perturbation to the airfoil surface, centered around a specific location x_n along the chord. The mathematical form of a single Hicks–Henne bump function is given by:

$$f_n(x) = \sin^3(\pi x^{e_n}), \quad \text{where} \quad e_n = \frac{\log(0.5)}{\log(x_n)}, \quad x \in [0, 1], \tag{29}$$

where x_n determines the peak location of the bump on the normalised chord. To compute the total shape change, multiple bump functions are combined:

$$\Delta y = \sum_{n=1}^{N} \delta_n f_n(x), \tag{30}$$

with N being the number of bumps applied and δ_n representing the design variable (i.e., the amplitude) for each bump. These functions are applied independently to the upper and lower surfaces of the airfoil, allowing asymmetric shape modifications [8]. An example of this bump-based surface parameterisation is illustrated in Figure 11. Once the surface geometry is updated using these functions during each design iteration, the surrounding volume mesh is deformed accordingly using a spring analogy method to maintain mesh quality around the airfoil.

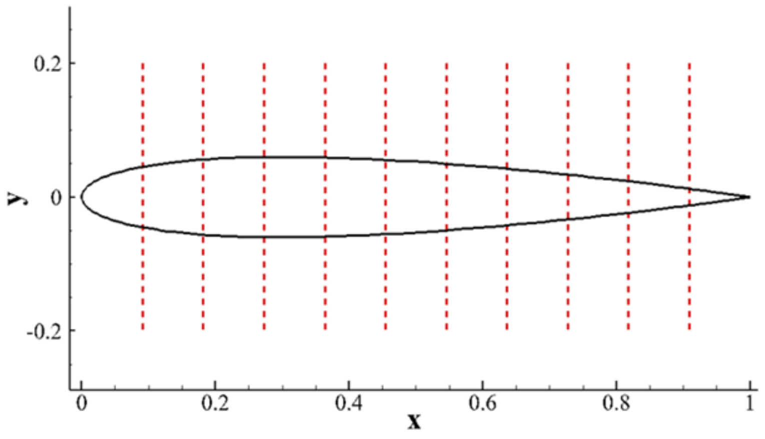


Figure 11. Hicks-Henne discretizations

3.6 Methods for Reducing Simulation Time

Reducing simulation time is critical for enabling efficient design iterations, especially when dealing with high-fidelity CFD simulations involving complex geometries for a high number of cases to study. Long runtimes can limit the scope and practicality of these studies and may hinder the application of open-source tools in time-sensitive environments. A number of the methods evaluated to tackle this problem are outlined below.

3.6.1 MPI

Both Gmsh and SU2 use MPI (Message Passing Interface) which is a C++ library to support distributed-memory parallelism through domain decomposition. This involves splitting the computational domain into subdomains that are processed simultaneously across multiple nodes. In Gmsh, MPI primarily aids in mesh partitioning and parallel mesh export for large 3D problems. In SU2, each process handles a portion of the mesh, and inter-process communication at shared boundaries is managed using non-blocking routines like `MPI_Isend` and `MPI_Irecv`, with synchronisation via `MPI_Waitall`. This parallel structure enables faster computations for larger simulations [48].

3.6.2 CFL Adapt

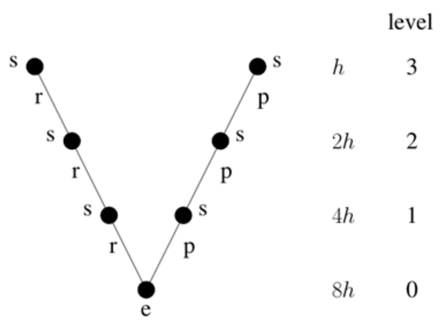
In SU2, the Courant–Friedrichs–Lewy (CFL) number is used to control the time step size relative to the local flow speed and mesh spacing. It is defined as:

$$\mathrm{CFL} = \frac{u\Delta t}{\Delta x}, \tag{31}$$

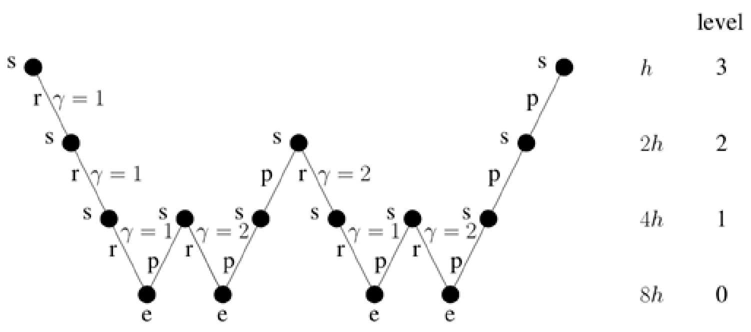
where u is the local flow velocity, Δt is the time step, and Δx is the local cell size. Although implicit schemes are theoretically unconditionally stable any time step, using an inappropriate CFL number can still negatively impact simulation accuracy and convergence. SU2 can employ a CFL adaptation strategy to increase or decrease the CFL number over the course of a simulation [49]. This helps to stabilise convergence when the initial solution is far from steady, or accelerate convergence if the CFL number is unnecessarily low. The `CFL_ADAPT_PARAM` controls how the CFL number is adjusted during the simulation [38]: the first two values set the factors by which the CFL is decreased or increased based on convergence behavior, while the last two define the minimum and maximum allowable CFL values.

3.6.3 Multigrid Method

SU2 incorporates multigrid methods to accelerate the convergence of both primal and adjoint solvers by efficiently resolving errors across multiple spatial scales [48]. It does this by recursively applying correction schemes to increasingly coarser grids. The general approach follows a structured multigrid cycle [50], where smoothing operations are performed on the fine grid to reduce high-frequency errors, and residuals are restricted to coarser grids to target low-frequency error components. These corrections are then interpolated back to the fine grid. SU2 uses this approach within a recursive framework, solving the coarsest grid problem either directly or with a few iterations of an iterative method. The two different cycle configurations were tested, and these determine how many times the process traverses down and up the grid hierarchy; this is visually illustrated in Figure 12. `MG_PRE_SMOOTH` and `MG_POST_SMOOTH` specify the number of pre- and post-smoothing iterations at each level, improving convergence by reducing high-frequency errors. `MG_CORRECTION_SMOOTH` applies optional smoothing to the correction after prolongation. The damping factors `MG_DAMP_RESTRICTION` and `MG_DAMP_PROLONGATION` control how strongly the residuals and corrections are transferred between levels, helping to stabilise and balance the multigrid process [38].



(a) V Cycle



(b) W Cycle

Figure 12. Multigrid methods, , s – smoothing, r – restriction, p – prolongated, e – exact solver [50]

4 Results and Discussion

This section presents and analyses the results obtained from the aerodynamic simulations and optimization cases following the methodology and aims. Key trends, solver behavior, and limitations are discussed, alongside visualisations of appropriate variables to support the findings.

4.1 Wing-winglet Direct Flow Analysis

The first part of this project focused on analysing wing–winglet geometries generated in OpenVSP. This served as a verification for Gmsh by allowing comparison of flow results from Gmsh-generated meshes against previously obtained StarCCM+ simulations. Due to time constraints, this study was limited to inviscid analyses, with variations in sweep and cant angles used to assess how effective the simulations were. Tow and twist were kept at 0°.

4.1.1 Initial Setup and Mesh discretizations

The `OpenCASCADE` geometry kernel was used as an alternative to Gmsh's native geometry engine to enable the import of `.step` files. In this case, two geometries were imported: `wing.step`, generated via the `winggen.vscript` in OpenVSP, and `domain.step`, a bullet-shaped fluid domain. Both files were provided by Herrera [19]. A `BooleanDifference` operation was then performed to subtract the wing geometry from the domain, resulting in a clean fluid volume suitable for meshing.

The next stage involved setting up the mesh for this fluid volume, which was done by first applying refinements to the wing surface. The surface IDs corresponding to the wing geometry were first identified using the Gmsh GUI. Each boundary curve defining these surfaces was then assigned an appropriate number of equally spaced mesh points, proportional to the size of the surface it bounded. Then the volume mesh could then be created along with a mesh independence study. This was done by varying the `CharacteristicLengthFactor` parameter in Gmsh, which effectively acts as the global mesh base size for the volume mesh. The surface mesh fineness was also studied here by proportionally increasing the number of mesh points used to discretize the wing . Two separate wing–winglet configurations were tested, one at 0° cant and 0° sweep and one at 80° cant and 20° sweep, to ensure the robustness and generalisability of the results. The flow assumptions for this study are presented in Table 2. The mesh independence study results are presented in Table 3 and Table 4.

h	M_∞	Re	p_∞	T_∞	V_∞	ρ_∞	α
10 000 m	0.6	2.5×10^6	26 500 Pa	223 K	$230\,\mathrm{m\,s^{-1}}$	$0.41\,\mathrm{kg\,m^{-3}}$	3°

Table 2. Wing flow conditions.

Wing c0s0	**CL**	**Cd**	**ΔCL**	**ΔCd**
1	0.3534	0.01105		
0.8	0.2834	-0.01021	19.81	192.4
0.6	0.4393	-0.00977	-55.01	4.31
0.4	0.4876	-0.00940	-10.99	3.79
0.2	0.5011	-0.00925	-2.77	1.60
0.1	0.4937	-0.00923	1.48	0.22

Table 3. Wing mesh convergence case 1

Wing c80s20	**CL**	**Cd**	**ΔCL**	**ΔCd**
1	0.4379	-0.0254		
0.8	0.5113	-0.0232	-16.76	8.66
0.6	0.6358	-0.01461	-24.35	37.03
0.4	0.6892	-0.01198	-8.40	18.00
0.2	0.6689	-0.01141	2.95	4.76
0.1	0.6605	-0.01121	1.26	1.75

Table 4. Wing mesh convergence case 2

The drag coefficient (C_D) varies by less than 5% when the base size is set to 0.2, indicating that mesh convergence is achieved at this resolution. Therefore, the base size of 0.2 was selected for all subsequent simulations. The exact number of mesh points used for each wing component is listed in Table 5, and the resulting surface mesh can be seen in Figure 13. The average total number of triangular elements used for the wing surface was 59612, and the Frontal-Delaunay algorithm was employed. The 3D volume mesh was generated using Gmsh's Delaunay tetrahedralisation algorithm and contained 920,480 cells and 1,234,567 nodes on average.

Surface	**Number of Mesh Points**
Main Wing	160
Blended Section	30
Winglet	40
Wing End	20
All Trailing Edges	20

Table 5. Number of mesh points used on different curves defining the different surfaces of the wing geometry.

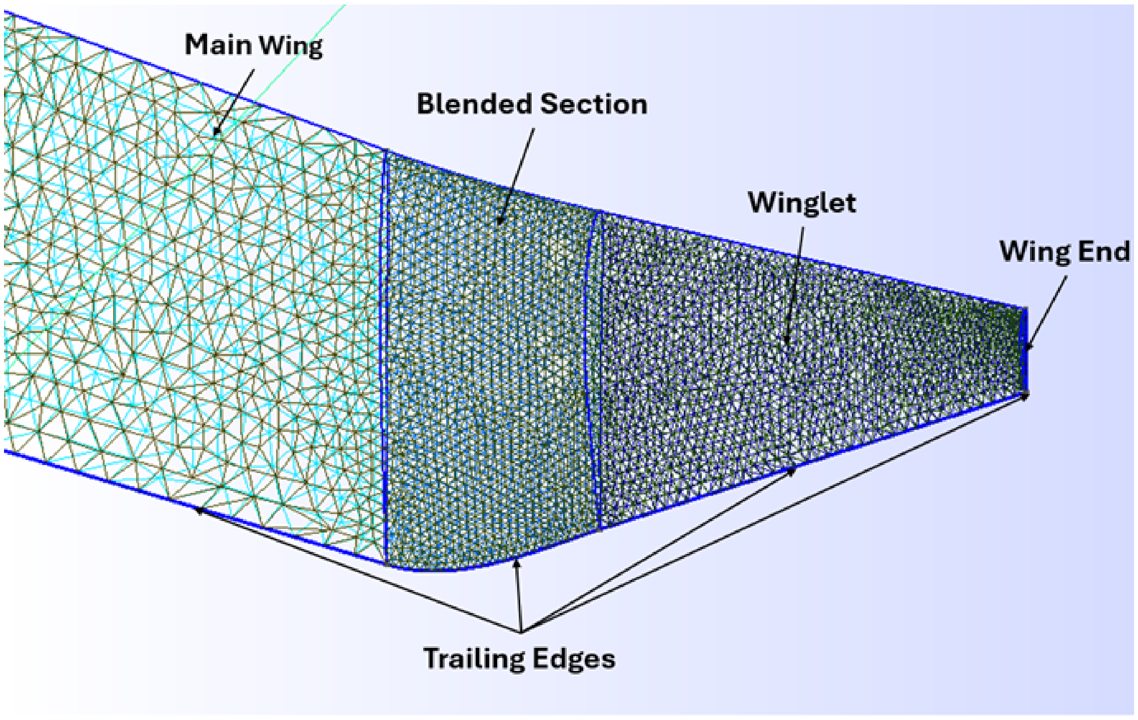


Figure 13. Wing surface mesh

4.1.2 Direct Flow Simulation Anlysis

A study investigating the effects of cant angle and sweep angle on lift and drag coefficients was conducted. The cant angle was varied from -120° to 120° in 4° increments, and the sweep angle from -20° to 20° in 2° increments, resulting in a total of 1121 unique configurations. The corresponding results are presented in Figure 14 and an example of the velocity field around the wing is shown in Figure 15.

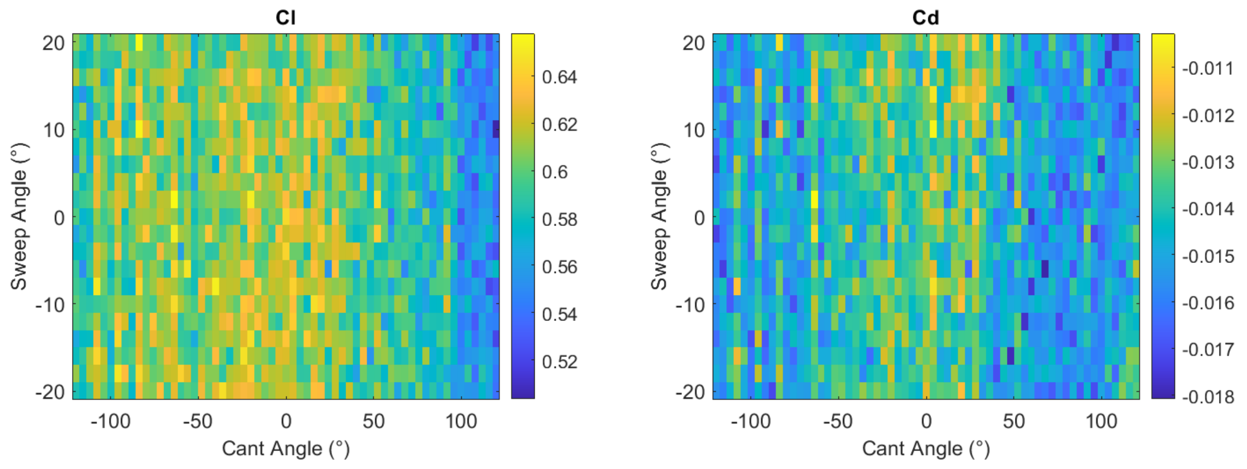
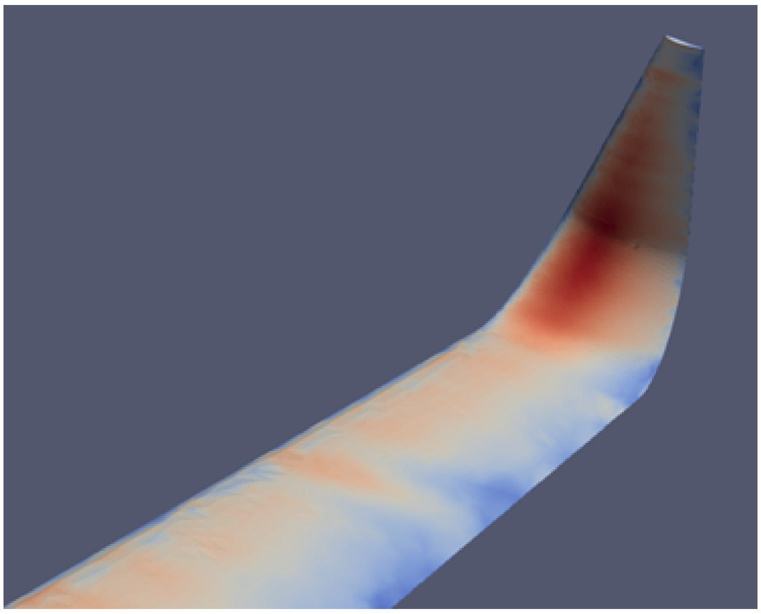
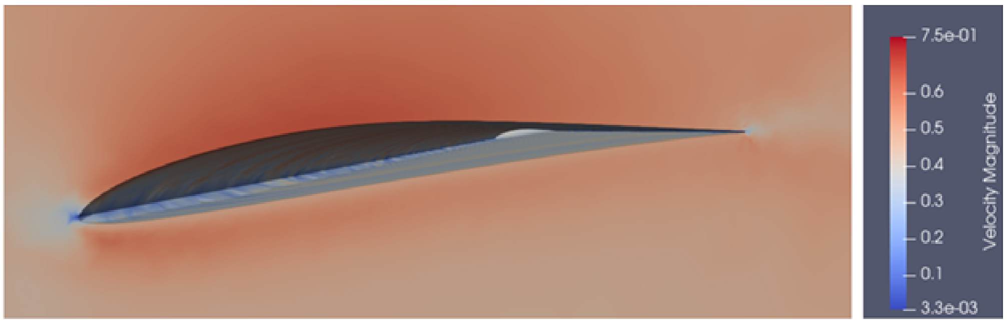


Figure 14. C_L and C_d of varying wing-winglet configurations

The results show that C_L is maximized in a central region where the cant angle ranges approximately from −40° to +40°, while variations in sweep angle appeared to have a relatively minor effect on lift. Beyond this range, lift performance declines, particularly as the cant angle approaches +120°. Interestingly, the drop in C_L is less pronounced at −120°, indicating that positive angles of attack yielded better performance when paired with negative cant angles. For C_d, all the results indicated that there was negative drag. Physically, drag should always be non-negative, so this indicates an issue with the mesh quality. Several factors likely contributed to this anomaly. First, messages such as "nodes could not be inserted" and "ill-shaped tets are still in the mesh" from the Gmsh meshing stage suggest that the computational mesh contains poorly defined, highly skewed, or missing elements. These can introduce significant errors into the pressure distribution and surface integration, particularly at the trailing edge, leading to unreliable force calculations. Additionally, since the Euler equations neglect viscous effects, errors introduced by a poor-quality mesh can be further amplified, leading to unreliable aerodynamic predictions. These issues highlight the sensitivity of Euler simulations to mesh quality. This suggests that further research should focus on exploring alternative mesh settings or improved surface meshing strategies to enhance simulation reliability and accuracy.



(a) Wing surface



(b) Root airfoil

Figure 15. Euler simulation velocity contours

4.2 Aerodynamic Shape Optimization

The next part of this project focused on investigating the capabilities of the SU2 shape optimization framework for viscous cases using the RANS solver. The study began with a series of 2D airfoil cases in transonic flow, analysed using the SU2 optimizer, followed by a 3D wing case in subsonic conditions. For the airfoil problem, we want to minimize the drag by changing the surface profile shape. To do so, we define a set of 38 equally spaced Hicks-Henne bump functions which serve as shape modification parameters across the airfoil surface.

4.2.1 Transonic Airfoil Flow Conditions and Mesh Validation

The base airfoil chosen for this study was the NACA 0015 which was deemed as an appropriate and simple enough geometry for simulation and optimization benchmarks. Transonic flow was chosen for the airfoil optimization cases due to its inherently non-linear and challenging nature, which serves as a rigorous test for the capabilities of the solver. Designing effective airfoils in this regime requires careful control of pressure distributions to minimize wave drag and avoid flow separation. According to transonic airfoil theory [51], a desirable pressure profile includes a well-defined, smooth supersonic plateau on the upper surface, terminated by a weak, well-positioned shock. The aft pressure recovery should exhibit a gentle gradient to prevent boundary layer separation, which would otherwise compromise lift and increase drag. This typically leads to airfoil shapes with large leading-edge radii and low upper surface curvature to help delay the shock and smooth out the pressure recovery.

The flow parameters used and mesh set up are presented in Table 6. The airfoil chord length was set to 1m, and the domain was defined as a 2D bullet-shaped region. The top and bottom boundaries were positioned 10m away from the airfoil, the inlet boundary was placed 10m upstream, and the outlet boundary extended 20m downstream. Both sharp and blunt trailing edge configurations were tested to assess how trailing edge geometry influences optimization behavior. The mesh was verified through a mesh independence study by varying the global base size, with the results shown in Figure 16. The convergence criterion was chosen such that the C_d should vary by less than 1%. This condition was satisfied at a base size of 0.25, which was therefore chosen as the design point for all airfoil optimization simulations. The resulting meshes contained an average of 68,848 elements and 37,407 nodes.

h	M_∞	Re	C_f	δ_{99}	y_1	N_{BL}
10 000 m	0.8	6.77×10^6	0.002 73	0.0159 m	3.99×10^{-6} m	37

Table 6. Transonic airfoil flow conditions.

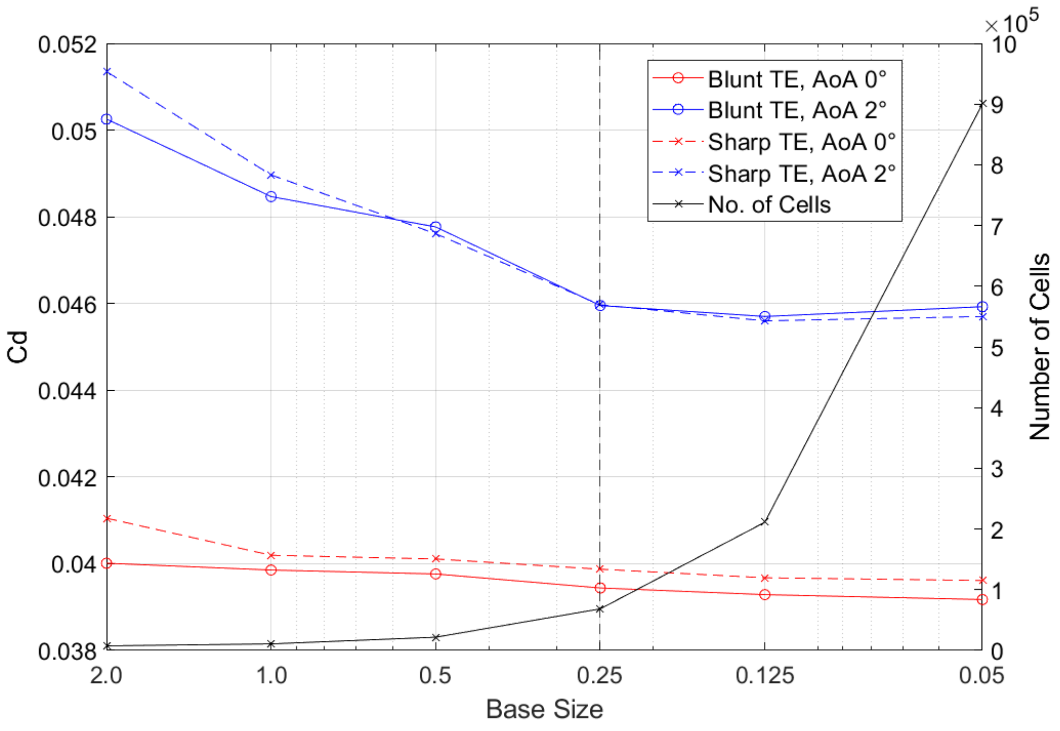
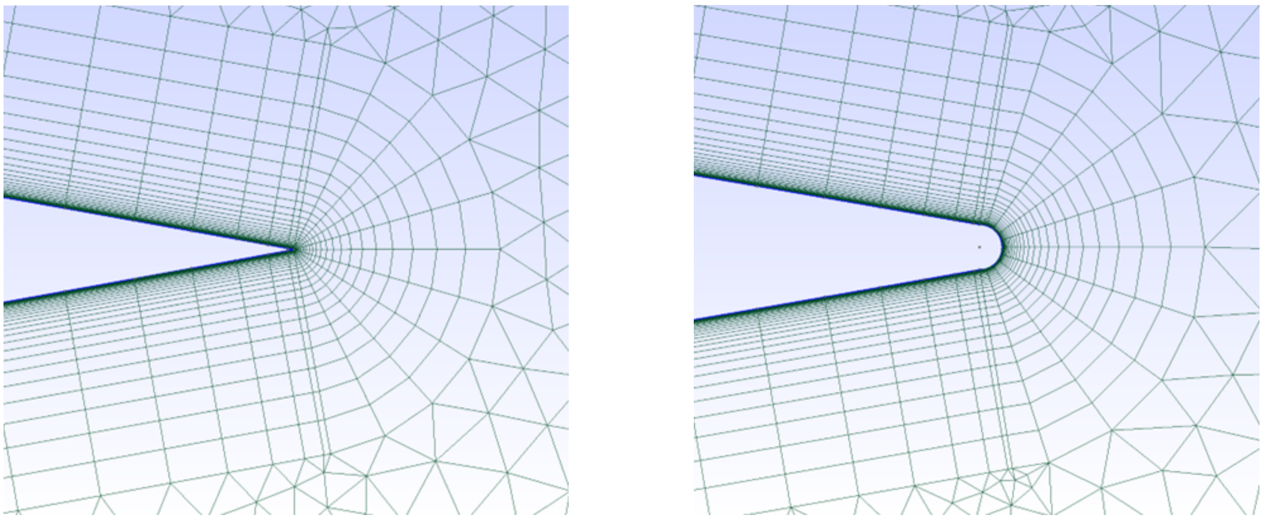


Figure 16. Airfoil mesh independence

4.2.2 Sharp Trailing Edge Issues

An analysis of using both a sharp and blunt Trailing Edge (TE) for the airfoils was done. For the sharp TE, the `BoundaryLayerFanPoints` setting was used on the point at the TE, and the number of fan elements was set to 10 and can be seen on Figure 17a. For the blunt TE configuration, seen on Figure 17b,a `Transfinite Circle Curve` was used with 10 linear panels to capture this blunt end.



(a) Sharp TE **(b)** Blunt TE

Figure 17. Sharp and blunt trailing edge

While the blunt TE configuration produced promising results, the sharp TE configuration encountered several issues when used with the SU2 optimizer. In many cases, the optimization produced unrealistic or highly distorted geometries, leading to simulation failures. These failures occurred either due to numerical divergence or the inability to converge to a steady-state solution. Examples of the shapes from resulting optimizations are shown in Figure 18. Unconstrained optimization was performed under both subsonic (Figure 18c) and supersonic (Figure 18b) conditions. The lack of geometric constraints near the trailing edge likely allowed the optimizer to explore infeasible regions of the design space, leading to highly distorted and non-physical airfoil shapes. Minimum thickness constraints were also tested with the optimizer (Figure 18a and Figure 18d) but unrealistic geometries were still produced. This behavior could be attributed to the sharp TE's increased sensitivity to small geometric perturbations, which can introduce instabilities in the flow solver. This means that the adjoint sensitivity field near a sharp TE can contain large gradients which can lead to overly aggressive or unstable shape updates.

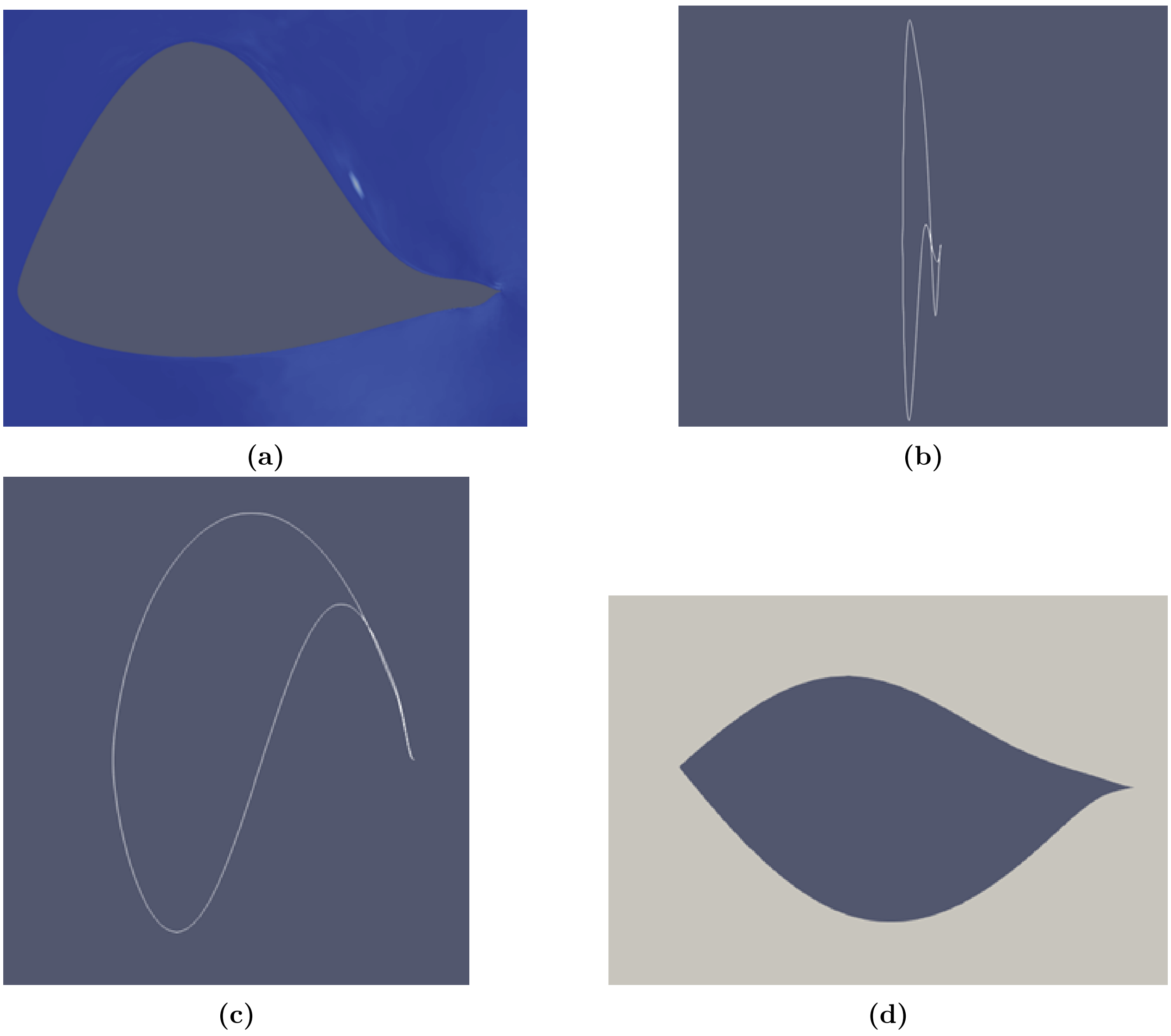


Figure 18. Resulting shapes from sharp TE

4.2.3 Blunt Trailing Edge Transonic Airfoil Analysis

NACA0015 Starting at 0° Angle of Attack

Two optimization cases were explored for the NACA0015 starting at 0° Angle of Attack (AoA). These can be written as:

$$\begin{aligned} \text{Minimise:} &\quad C_d \\ \text{Subject to:} &\quad C_l = 0.0 \\ &\quad -0.093 \leq C_m \leq 0.093 \end{aligned} \qquad\qquad \begin{aligned} \text{Minimise:} &\quad C_d \\ \text{Subject to:} &\quad C_l = 0.1 \\ &\quad -0.093 \leq C_m \leq 0.093 \end{aligned}$$

Case 1 focused on drag minimisation by allowing the optimizer to reduce the drag by reshaping the airfoil while maintaining a target C_L of 0.0. Case 2 aimed to evaluate whether the optimizer could effectively increase the lift coefficient from its initial value, either by modifying the airfoil shape or by adjusting the angle of attack. In fixed-C_L mode, SU2 uses a proportional control strategy based on the parameter `DCL_DALPHA`, which governs how the angle of attack is updated according to the difference between the current and target C_L values. Specifically, the angle of attack is incremented by $\delta\alpha = (C_{l,target} - C_{l,current})/(\frac{\partial C_l}{\partial \alpha})$. The constraint on the pitching moment coefficient is used to ensure the airfoils have acceptable longitudinal static stability characteristics. Both cases started with a C_d of 0.03929.

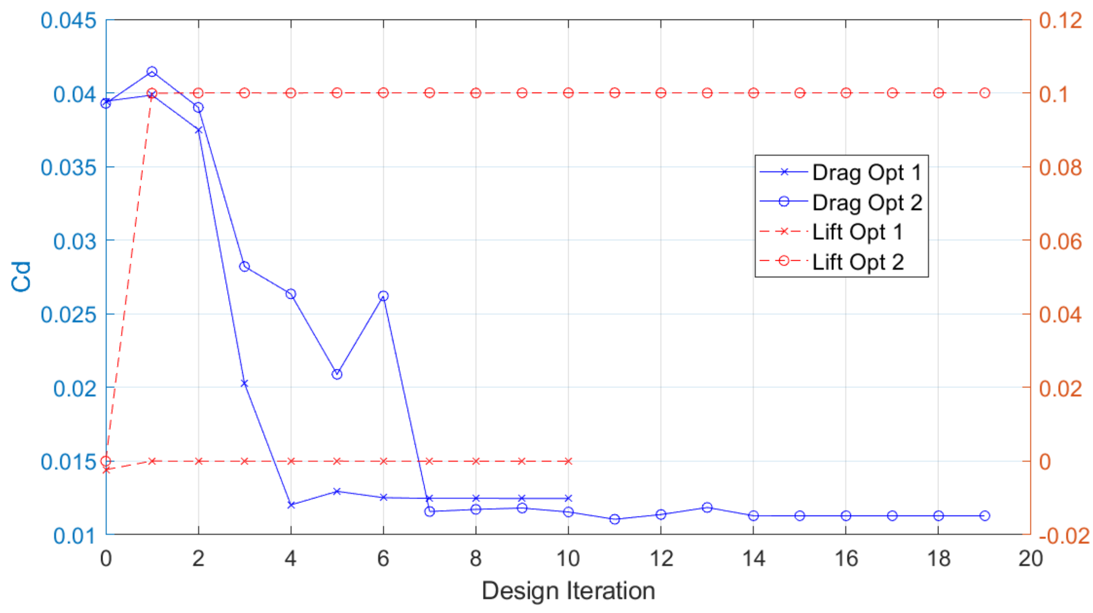
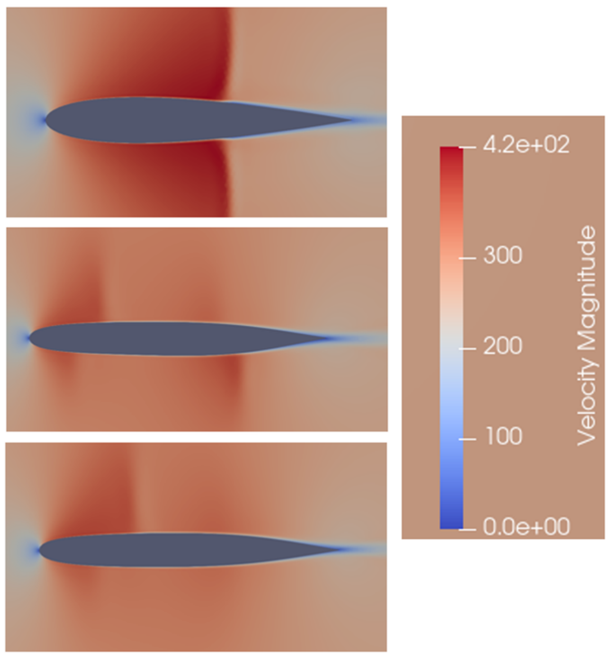


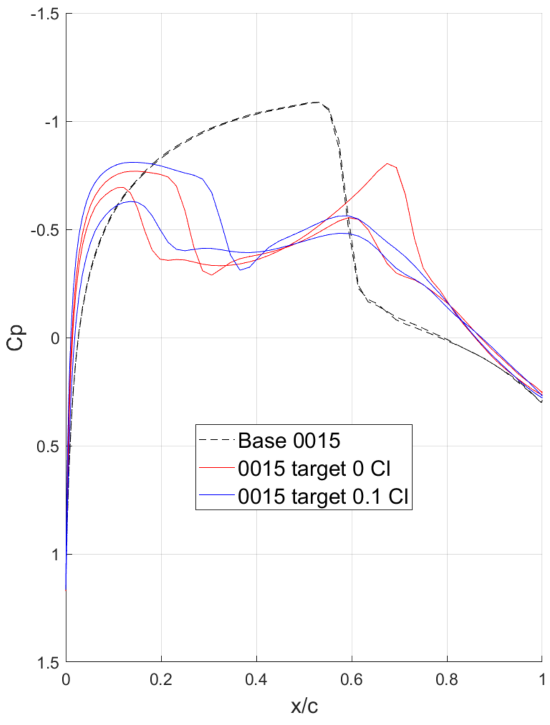
Figure 19. NACA0015 starting at 0° AoA optimization history

The resulting design history is shown in Figure 19. This illustrates the progression of both lift and drag over the course of the optimization for the two separate cases. The optimizer was able to significantly reduce the drag coefficient for both cases, reaching a C_d of 0.01246 for case 1 (68% improvement) and a C_d of 0.01129 for case 2 (71% improvement). The lift maintained a steady value throughout the optimization process, demonstrating that the fixed-C_l constraint was successfully enforced. For case 2, the C_l was set to the target of 0.1 within the first iteration and stayed there, and from the velocity profile and surface pressure plots shown in Figure 20, it can be observed that a slight increase in angle of attack was used to achieve the target lift coefficient, thereby demonstrating that the `DCL_DALPHA` control strategy was functioning as intended.

Figure 20 shows that the optimizer effectively modified the airfoil shape to produce a steeper rise in the negative pressure coefficient near the leading edge, followed by a more gradual pressure recovery leading into a much weaker normal shock compared to the baseline airfoil. Notably, the optimized shape exhibits two weaker shocks at around $x/c \approx 0.3$ and another near $x/c \approx 0.65$, this is indicated by both the surface pressure distribution and velocity profiles. This shock-splitting strategy appears to be effective, as it contributes to the substantial reduction in drag. Additionally, the optimized airfoils exhibit reduced overall thickness as seen from the velocity profile, which leads to a smaller wake and further contributes to the decrease in drag.



(a) Velocity profile of base shape (top), target C_l 0.0 (middle), and target C_l 0.1 (bottom)

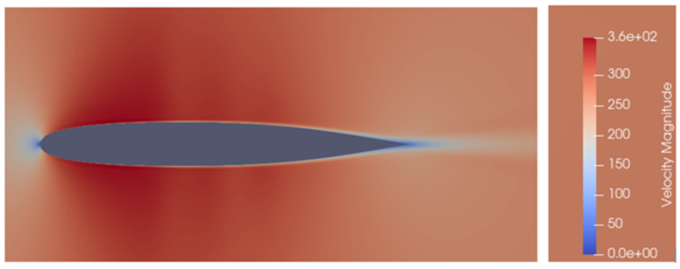


(b) Surface pressure plots

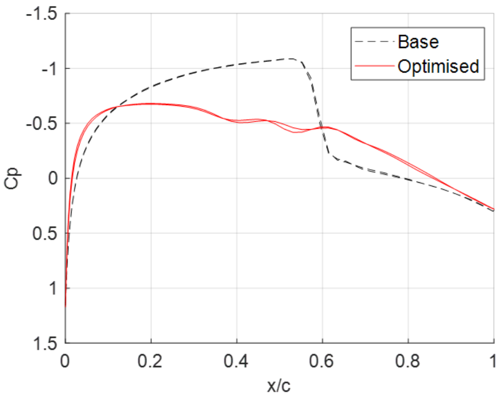
Figure 20. NACA0015 starting at 0° AoA results

Interestingly, some asymmetry appears to have been introduced for case 1 in the airfoil shape and/or angle of attack during optimization. This is unexpected, as a symmetric airfoil targeting zero angle of attack should maintain its symmetry since, in theory, the aerodynamic gradients on the upper and lower surfaces should be identical. This observation suggests some initial asymmetry existed in the optimization setup. One possible explanation is that the starting mesh was not perfectly symmetric, perhaps due to uneven spacing of mesh points along the airfoil surface. However, a more likely cause is that the C_L at 0° angle of attack was not exactly zero, prompting the optimizer to adjust the angle of attack slightly and change the shape asymmetrically to achieve exact zero lift. This small adjustment would naturally result in an asymmetric flow field. Given that the surface mesh points were symmetrically distributed, the latter explanation is more probable.

Therefore, the target C_L was readjusted to be -0.0023 (this value was recovered from the mesh independence study) and produced the results in Figure 21. This shows more expected behavior, as the optimized airfoil remained symmetric. This optimizer is once again able to increase the rate of negative C_P increase, and then gradually recover it along the chord. Curiously, the optimized shape appears to exhibit three very weak shocks on both the upper and lower surfaces. The airfoil's overall thickness was also noticeably reduced, leading to a smaller wake and further drag reduction. The final C_D value achieved was 0.01068 which is an improvement of 73%. This analysis highlights the sensitivity of the optimizer, as even a small change in the equality constraint (less than 0.01 C_L) significantly influenced the final results.



(a) Velocity profile



(b) Surface pressure plots

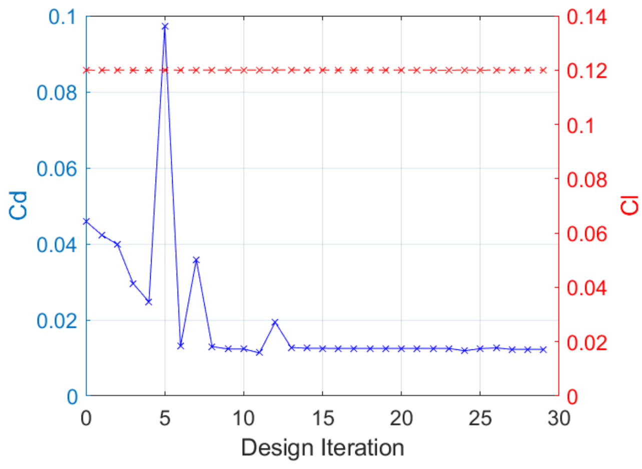
Figure 21. Readjusted target C_L results.

NACA0015 Starting at 2° Angle of Attack

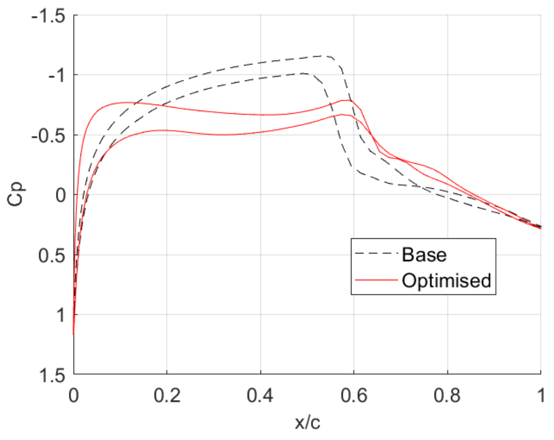
The next case investigated was using the NACA0015 airfoil at an initial AoA of 2°. This setup introduces a non-zero lift condition from the start, which is more representative of typical operating scenarios for airfoils. The exact optimization performed is written as:

$$\begin{aligned} \text{Minimise:} \quad & C_d \\ \text{Subject to:} \quad & C_l = 0.12 \\ & -0.093 \leq C_m \leq 0.093 \end{aligned}$$

As shown in Figure 22a, the optimizer successfully reduced the drag from an initial value of 0.0459 to 0.0122 (74% reduction) within the first 10 design iterations. Meanwhile, the lift coefficient stayed at a constant value of 0.12 throughout the process, verifying that the lift constraint was effectively enforced.



(a) Design iteration history



(b) Surface pressure plots

Figure 22. NACA0015 starting at 2° AoA results

The pressure coefficient distribution in Figure 22b further illustrates the aerodynamic changes. The baseline airfoil exhibits a strong normal shock on the upper and lower surfaces, indicated by a sharp negative pressure drop just after the mid-chord. The optimizer is able to create a shape that demonstrated a well-defined, smooth supersonic plateau on both the upper and lower surface, terminated by a much weaker, slightly more aft positioned shock. These results align with what is expected from transonic airfoil theory.

The velocity magnitude plots in Figure 23 support the findings from the surface pressure plots. In the baseline case, a distinct shock wave is visible on the airfoil. In contrast, the optimized airfoil shows a significantly weakened shock, with a smoother velocity gradient and a more uniform flow field around the airfoil. The reduction in velocity discontinuity indicates less intense shock formation, which directly contributes to the observed drag reduction. A slight flow separation can be seen just behind the shock on the upper surface of the initial airfoil; this is gone with the optimized airfoil, with the wake being narrower and more symmetric. The overall thickness is also reduced.

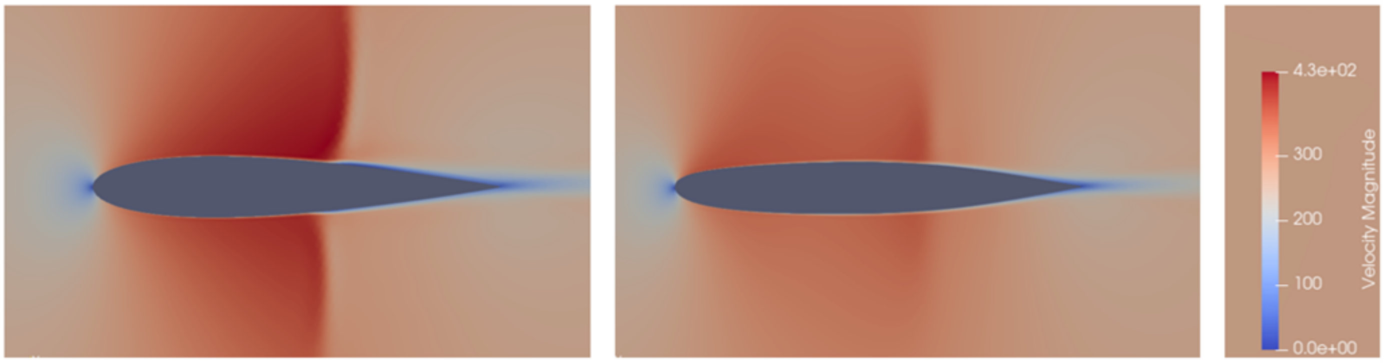


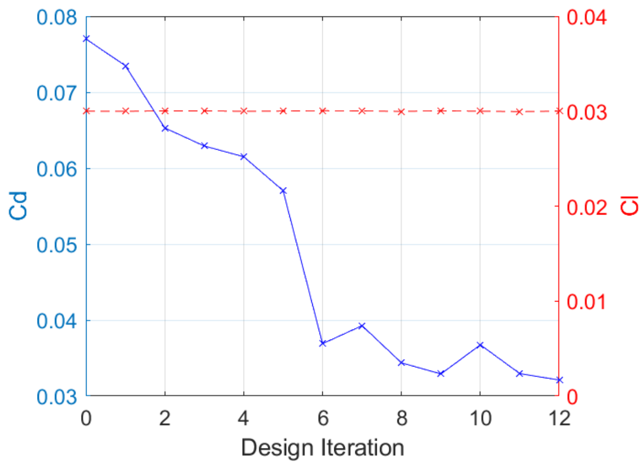
Figure 23. Velocity field of NACA0015 starting at 2° AoA results, initial shape (left) and optimized shape (right).

NACA4415 Starting at 0° AoA

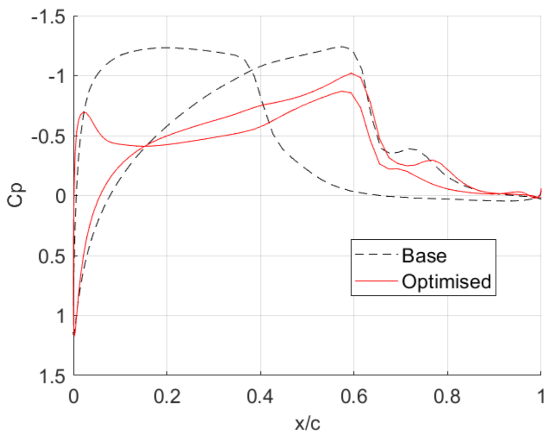
To further assess the capabilities of the SU2 optimizer under transonic flow conditions, an additional test case was conducted using the NACA4415 airfoil at 0° angle of attack. This setup aimed to test how the optimizer modifies a less familiar baseline aerodynamic geometry to manage shock-induced drag without sacrificing the original lift performance. The optimization problem can be written as follows:

$$\begin{aligned} \text{Minimise:}\quad & C_d \\ \text{Subject to:}\quad & C_l = 0.03 \\ & -0.093 \leq C_m \leq 0.093 \end{aligned}$$

Figure 24 shows the evolution of drag and lift over the course of the optimization. The drag is reduced from 0.07701 to 0.0321 (58% reduction) over 12 design iterations while the lift constraint was effectively maintained. The pressure coefficient comparison in Figure 24b shows that the baseline airfoil exhibits strong shock waves on both the upper and lower surfaces located at around $x/c \approx 0.6$ and $x/c \approx 0.4$ respectively. After optimization, both shocks are significantly weakened and align more closely around $x/c \approx 0.6$, resulting in reduced shock strength and a more balanced pressure distribution and better pressure recovery.



(a) Design Iteration History



(b) Surface pressure plots

Figure 24. NACA4415 starting at 0° AoA results

The corresponding velocity magnitude plots in Figure 25 further illustrate the aerodynamic improvements achieved during optimization. Notably, the shock on the lower surface shifts rearward, aligning more closely with the upper surface shock just after the mid-chord region. Furthermore, the visibly reduced shock strength is evident from the less sharp velocity transition upstream of the shocks. Notably, the wake region behind the airfoil remains relatively large in both the baseline and optimized airfoils, especially when compared to the NACA0015 results, indicating the inherent aerodynamic complexity and greater flow disturbance associated with the cambered NACA4415 profile in transonic conditions.

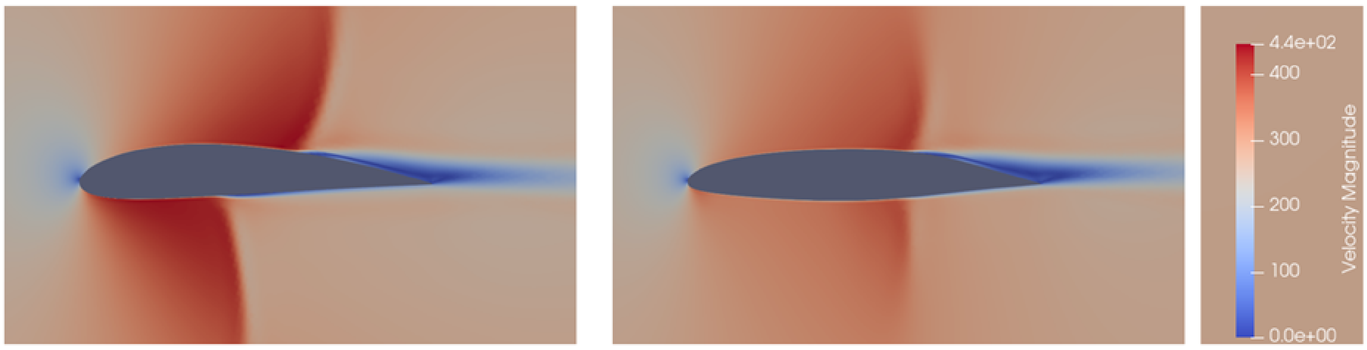


Figure 25. Velocity field of NACA4415 starting at 0° AoA results, initial shape (left) and optimized shape (right).

However, despite these promising intermediate results, the optimization did not fully converge. Shortly after the iterations shown, the simulation diverged and produced an unrealistic and highly distorted airfoil geometry, ultimately causing the solver to crash. This behavior highlights a known limitation of gradient-based shape optimization which is that if the initial setup lacks sufficient geometric stability mechanisms, the optimizer may explore infeasible regions of the design space. In this case, the combination of transonic sensitivity and a cambered profile likely amplified small shape perturbations, leading to numerical instability. This outcome suggests that more careful control of the design space, such as applying geometric bounds, is essential when dealing with less familiar baseline shapes like the NACA4415 in transonic flow.

4.2.4 3D Wing Optimization Progress

The next goal of this project was to explore the feasibility of extending shape optimization into three dimensions using a wing geometry generated with Gmsh. However, despite several efforts, this part of the study proved unsuccessful, largely due to difficulties associated with mesh generation and viscous flow simulation setup. The main challenge stemmed from creating a suitable 3D mesh that could support viscous simulations in SU2. While Gmsh offers powerful meshing capabilities, it lacks built-in support for automated 3D boundary layer generation through its native `.geo` scripting language. This significantly complicates mesh preparation for RANS simulations.

Gmsh does provide support for 3D boundary layer meshing via its Python API, but these scripts encountered instability when executed on the HPC cluster and would encounter segmentation faults when attempting to run. Therefore `.geo` scripts had to be used. Initial tests were performed using simplified geometries, including a box wing and a very coarse straight wing mesh. However, either the SU2 solver failed to converge for direct flow simulations, or the meshes themselves were not successfully generated or took impractically long to compute. These setbacks highlight a key limitation when using open-source tools like Gmsh for advanced 3D optimization workflows. In contrast, commercial software such as Star-CCM+ handles 3D meshing and boundary layer generation automatically and with greater robustness—features that are still maturing in open-source alternatives.

Nonetheless, some progress was made in Gmsh through manual extrusion of surface layers, where boundary layer thickness, number of layers, and growth rate had to be explicitly defined for this extrusion. The current state of the Gmsh scripting approach for 3D boundary layer extrusion is available in the project's GitHub repository for future development.

4.3 Simulation Runtime Reductions

Two test cases were investigated to evaluate runtime acceleration strategies. The first involved the inviscid wing configuration described in Section 4.1, used to assess MPI-based speedup for both Gmsh and the SU2. In this case, 3D mesh generation in Gmsh and flow simulation with the `SU2_CFD` executable were tested using varying MPI settings. A total of nine consecutive simulations were run, each with a cant angle fixed at 20°, and sweep angle varying from −20° to +20° in increments of 4°. The second test case focused on a single design iteration of shape optimization for a NACA 0015 airfoil at 0° angle of attack with a target C_L of -0.0023. This case was used to evaluate the runtime performance of the full SU2 toolchain described in Section 3.4.1, including all relevant executables. MPI values ranging from 2 to 48 were tested on the Imperial HPC, and the results are presented in Figure 29.

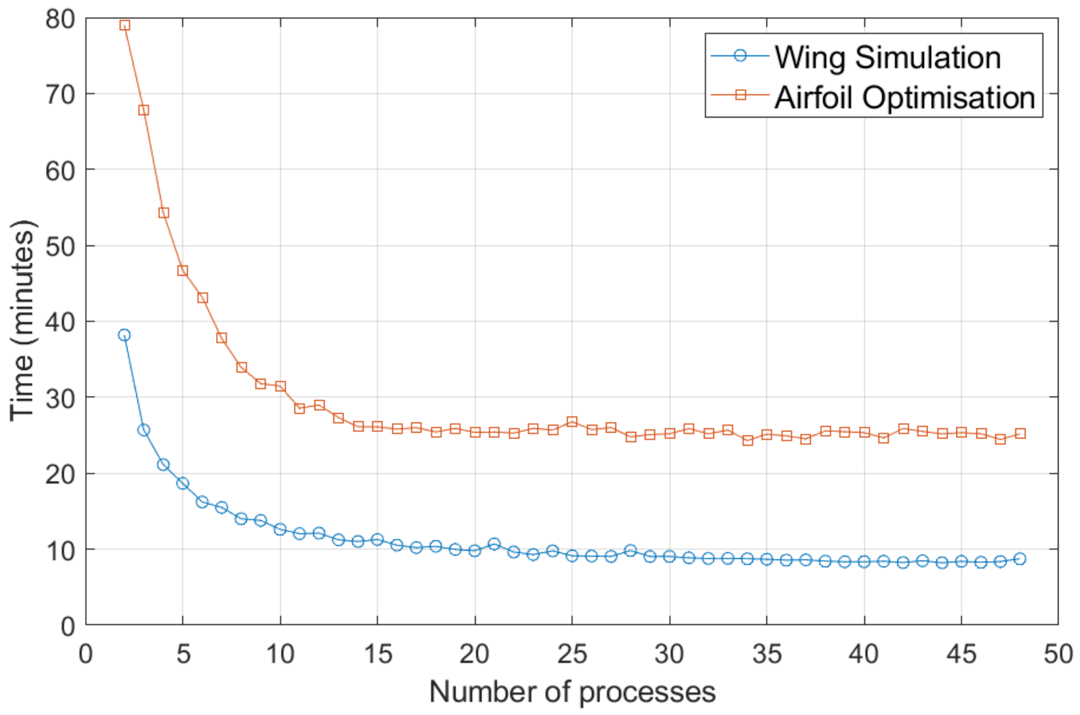


Figure 26. MPI speedup

The results for both cases show the typical behavior of parallel speedup with increasing MPI processes, as simulation time decreases rapidly (over 3x) up to around 14 processes, and then performance gains start to plateau due to communication overheads and process management inefficiencies. Another issue encountered was that requesting a higher number of processes on the HPC increased queue wait times, as more nodes were required to become available. Therefore, 16 processes was determined to be a good compromise between computational speed and job scheduling efficiency.

Multigrid acceleration and adaptive CFL techniques were then tested using 16 MPI processes for both the same cases. Both V-cycle and W-cycle multigrid schemes were evaluated using three levels, with the following settings: `MG_PRE_SMOOTH=( 1, 1, 1, 1)`, `MG_POST_SMOOTH=( 1, 1, 1, 1)`, `MG_CORRECTION_SMOOTH=( 0, 0, 0, 0)`, `MG_DAMP_RESTRICTION =0.7`, and `MG_DAMP_PROLONGATION=0.7`. The parameter used for CFL adaptation was `CFL_ADAPT_PARAM = ( 0.5, 1.5, 1.0, 10000.0 )`. These settings were previously explained in Section 3.6 and the following results are presented in Figure 27, with the error in C_L and C_d from the base case recorded in table 7.

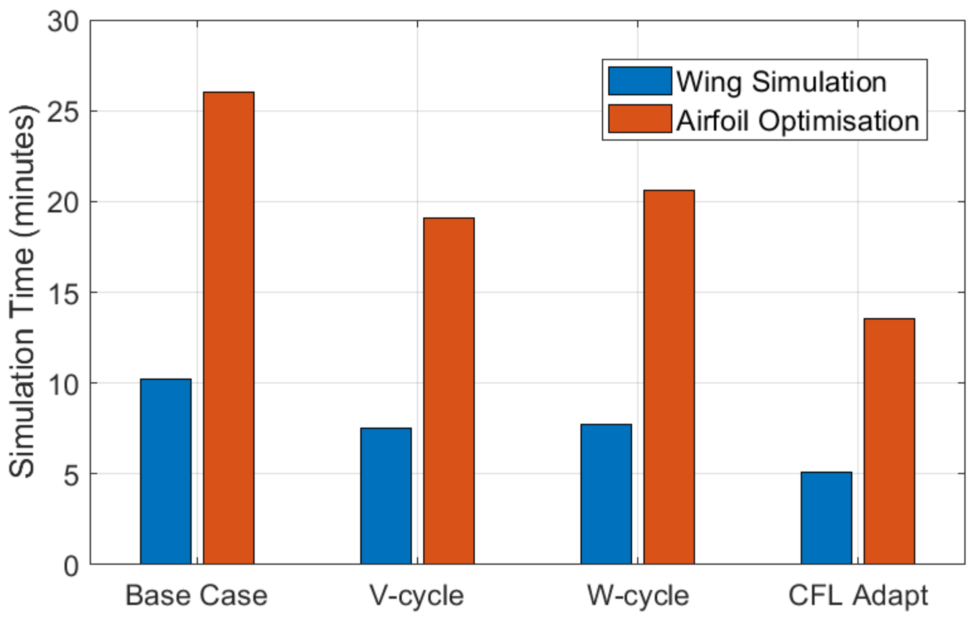


Figure 27. Effect of Multigrid and CFL Adaptation on Simulation Time

	V-cycle (%)		**W-cycle (%)**		**CFL adapt (%)**	
	Case 1	**Case 2**	**Case 1**	**Case 2**	**Case 1**	**Case 2**
ε_{C_L}	2.06	1.56	0.88	0.53	3.13	2.17
ε_{C_D}	3.54	1.97	0.84	0.61	4.26	2.44

Table 7. Errors of multigrid and CFL adaptation.

The results show that using CFL adaptation yields the greatest reduction in simulation time, with an average speedup of 46% across the two test cases. However, it introduced the largest error in both C_L and C_d. Multigrid V-cycle had average runtime improvements of 36%, and the multigrid W-cycle provided the most accurate results relative to the baseline, while still offering an average runtime improvement of 29%. This makes sense because W-cycles perform more aggressive inter-level corrections and involve more frequent traversal between coarse and fine grids.

5 Conclusion and Future Work

5.1 Conclusion

This project was successfully able to explore the feasibility and effectiveness of conducting aerodynamic shape optimization using entirely open-source software. A foundational workflow for simulating complicated wing–winglet geometries using OpenVSP-generated geometries and Gmsh-based meshing was implemented and a large parametric study was carried out using inviscid Euler simulations on SU2 to assess the influence of cant and sweep angles on lift and drag. While general trends in lift were consistent with aerodynamic expectations, the appearance of negative drag values in all cases highlights critical limitations in mesh quality. This highlights the need to adopt viscous solvers for more reliable aerodynamic analysis, and also underscores Gmsh's limitations with complex 3D meshes compared to commercial meshers.

ASO was explored on 2D airfoils that were both generated and meshed using Gmsh. The SU2 optimizer was employed and achieved a reduction in C_D of up to 74% in transonic cases, demonstrating its ability to improve the airfoil shape for a specified flow regime. This workflow successfully showed that, with appropriate setup and parameter tuning, open-source tools can yield meaningful aerodynamic insights and shape improvements in 2D transonic regimes that are comparable to those obtained using commercial software.

Efforts to extend the workflow to 3D wing optimization encountered practical barriers. Gmsh lacks built-in support for automated 3D boundary layer mesh generation, and while Python scripting offers a workaround, it was unstable on the HPC environment used. These limitations point to the current challenges of scaling open-source workflows to full 3D viscous simulations without more advanced meshing tools or significant manual setup.

Simulation runtime reduction testing showed over a $3\times$ speedup through effective use of MPI and up to a 36% reduction in runtime using multigrid methods, while also maintaining low errors in the measured variables.

Despite these challenges, the work accomplished in this project demonstrates the potential and current limitations of open-source software for high-fidelity aerodynamic optiimzation and the work so far can be found here (https://github.com/JopaulJ/FYP2025). With further development, the presented workflow can form the foundation for more accessible and scalable aerodynamic design pipelines.

5.2 Future Work

Several logical improvements to the current workflow can be identified. First, revisiting the mesh generation process for the complex wing–winglet configuration to address the occurrence of negative drag coefficients. Investigating alternative meshing strategies within Gmsh or other open-source tools could improve element quality and solver stability. Extending the workflow to support viscous simulations would be a valuable next step, including efforts to get the Gmsh Python API running on the Imperial HPC system. In the 2D optimization cases, introducing additional constraints, such as maximum airfoil area or spacing limits, could help prevent the divergence issues observed in some simulation cases. Exploring 3D shape optimization using SU2's Free Form Deformation (FFD) capabilities could also be a promising direction. This would allow for a more realistic assessment of aerodynamic improvements in full 3D wing configurations, pushing the workflow closer to practical design scenarios. More rigorous testing is recommended for the simulation runtime reduction analysis by exploring additional solver settings, fine-tuning the parameters used in the three evaluated methods, and testing combinations of these approaches.

References

[1] UK Government. Net zero strategy: Build back greener, 2021. URL https://www.gov.uk/government/publications/net-zero-strategy. Accessed: 2025-05-10.

[2] UK Government, Department for Transport. Greenhouse gas emissions from transport in 2022: Transport and environment statistics 2024, 2024. URL https://www.gov.uk/government/statistics/transport-and-environment-statistics-2024/greenhouse-gas-emissions-from-transport-in-2022. Accessed: 2025-05-10.

[3] J. Reuther, A. Jameson, W.J. Batina, and D. Saunders. Concept to reality: Contributions of the langley research center to us civil aircraft of the 1990s. Technical Report NASA-CR-201745, NASA, 1996. URL https://ntrs.nasa.gov/citations/20030059513. Accessed: 2025-05-10.

[4] D. McLean. Wingtip devices: What they do and how they do it. Technical report, Boeing, 2005. URL https://mentourpilot.com/wp-content/uploads/2020/10/Wingtip_Devices-Doug-McLean-Boeing-flight-safety-conference-2005.pdf. Accessed: 2025-05-10.

[5] W. Freitag and E.T. Sculze. Blended einglets improve performance. *Boeing*, 2013.

[6] A.M.St. Laurent. *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing New Software*. O'Reilly Media, Inc, 2004.

[7] J.E. Corbly. The free software alternative: Freeware, open-source software, and libraries. *Information Technology and Libraries*, (Vol. 33 No. 3), 2014.

[8] T.D. Economon, F. Palacios, S.R. Copeland, T.W. Lukaczyk, and J.J. Alonso. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016. doi: 10.2514/1.J053813. URL https://doi.org/10.2514/1.J053813.

[9] J.D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill Education, 2011.

[10] International Civil Aviation Organization. Annex 14 – Aerodromes, Volume I: Aerodrome Design and Operations, 2018. URL https://www.iacm.gov.mz/app/uploads/2018/12/an_14_v1_Aerodromes_8ed._2018_rev.14_01.07.18.pdf.

[11] M.A.C. Queirolo. Impact of morphing winglets on aircraft performance. Master's thesis, Mauricio Andrés Cancino Queirolo, 2018.

[12] R.T. Whitecomb. A design approach and selected wind tunnel results at high subsonic speeds for wing-tip mounted winglets. Technical report, NASA Langley Research Center Hampton, VA, United States, 1976.

[13] Boeing Commercial Airplanes. 737 max at winglet: The most efficient winglet on any airplane, 2024. URL https://www.boeing.com/commercial/737max/737-max-winglets.

[14] I.G. Martínez, F. Afonso, S. Rodrigues, and F. Lau. A sequential approach for aerodynamic shape optimization with topology optimization of airfoils. *Mathematical and Computational Applications 26, no. 2: 34.*, 2021.

[15] J. Martins and A. Lambe. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal 51(9):20492075*, 2013. doi: 10.2514/1.J051895.

[16] T. August, W. Chen, and K. Zhu. Competition among proprietary and open-source software firms:the role of licensing in strategic contribution. *Management Science Vol. 67*, 2020.

[17] G. Pinto, I. Steinmacher, L.F. Dias, and M. Gerosa. On the challenges of open-sourcing proprietary software projects. *Empir Software Eng Vol 23*, 2018.

[18] M. Hoffmann, F. Nagle, and Y. Zhou. The value of open source software. Technical report, Harvard Business School, 2024.

[19] J.G. Herrera. Automated aerodynamic shape optimisation of winglets with su2 on imperial's high-performance computer cluster. Master's thesis, Department of Aeronautics, Imperial College London, 2024.

[20] C. King. Aerodynamic shape optimisation of winglets. Master's thesis, Department of Aeronautics, Imperial College London, 2023.

[21] J.T.W. Loong. Aerodynamic shape optimisation of winglets. Master's thesis, Department of Aeronautics, Imperial College London, 2022.

[22] M. Alder, E. Moerland, B. Nagel, and J. Jepsen. Recent advances in establishing a common language for aircraft design with cpacs. Technical report, Institute of System Architectures in Aeronautics, German Aerospace Center (DLR), 2020.

[23] Siemens Digital Industries Software. Simcenter star-ccm user guide v. 2021.1, 2021.

[24] NASA OpenVSP. Openvsp: Open vehicle sketch pad. https://openvsp.org/, 2025. Accessed: 2025-05-10.

[25] ResearchGate. Flexop demonstrator aircraft image, 2029. URL https://www.researchgate.net/figure/FLEXOP-demonstrator-aircraft_fig1_337487592.

[26] C. Geuzaine and J. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. https://gmsh.info, 2009. Version 4.13.1.

[27] R.L. Fearn. Airfoil aerodynamics using panel methods. *The Mathematica® Journal*, 2008.

[28] M. Drela and H. Youngren. Airfoil tools. https://airfoiltools.com, 2024. Accessed: May 2025.

[29] C. Geuzaine and J. Remacle. Gmsh reference manual. Technical report, 2025. URL https://gmsh.info/dev/doc/texinfo/gmsh.pdf.

[30] B. Delaunay. Sur la sphère vide [on the empty sphere]. Technical report, Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles, 1934. URL https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=im&paperid=4937&option_lang=eng.

[31] G.W. Lucas. An introduction to the delaunay triangulation, tinfour project. URL https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html.

[32] R.A. Dwyer. A simple divide-and-conquer algorithm for computing delaunay triangulations in o(n log log n) expected time. Technical report, Yorktown Heights, 1986. URL https://dl.acm.org/doi/pdf/10.1145/10515.10545.

[33] N.P. Weatherill. The integrity of geometrical boundaries in the two-dimensional delaunay triangulation. Technical report, Institute f o r Numerical Methods in Engineering, University College of Swansea, 1990. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1630060206.

[34] S. Rebay. Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm. *Journal of Computational Physics*, 1993. URL https://www.academia.edu/71741105/Efficient_Unstructured_Mesh_Generation_by_Means_of_Delaunay_Triangulation_and_Bowyer_Watson_Algorithm.

[35] K. Gersten H. Schlichting. *Boundary-Layer Theory*. Springer Berlin, Heidelberg, 2017.

[36] S.B. Pope. *Turbluent Flows*. Cambridge University Press, 2000.

[37] A. Wimshurst. *Computational Fluid Dynamics Fundamentals Course*. 2025. URL https://www.fluidmechanics101.com/pages/shop.html.

[38] SU2 Foundation. Su2 tutorials, 2024. URL https://su2code.github.io/tutorials/home/. Accessed: 2025-05-16.

[39] F. Palacios, T.D. Economon, A. Aranake, S.R. Copeland, A.K. Lonkar, T.W. Lukaczyk, D.E. Manosalvas, K.R. Naik, S. Padron, B. Tracey, A. Variyar, and J.J. Alonso. *Stanford University Unstructured (SU2): Analysis and Design Technology for Turbulent Flows*. doi: 10.2514/6.2014-0243. URL https://arc.aiaa.org/doi/abs/10.2514/6.2014-0243.

[40] D.C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, 1998.

[41] F. White. *Viscous Fluid Flow*. McGraw Hill Inc., New York, 1974.

[42] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA*, 439, 1992. doi: 10.2514/6.1992-439.

[43] T.D. Economon, F. Palacios, J.J. Alonso, G. Bansal, D. Mudigere, A. Deshpande, A. Heinecke, and M. Smelyanskiy. *Towards High-Performance Optimizations of the Unstructured Open-Source SU2 Suite*. American Institute of Aeronautics and Astronautics, 2015. doi: 10.2514/6.2015-1949. URL https://arc.aiaa.org/doi/abs/10.2514/6.2015-1949.

[44] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the euler equations by finite volume methods using runge-kutta time stepping schemes. In *AIAA Paper*, number 1981-1259, July 1981. doi: 10.2514/6.1981-1259. URL http://aero-comlab.stanford.edu/Papers/jameson.aiaa.1981-1259.pdf.

[45] A. Syrakos, Y. Dimakopoulos, A. Goulas, and J. Tsamopoulos. A critical analysis of some popular methods for the calculation of the gradient in finite volume methods, with suggestions for improvements. *CoRR*, abs/1606.05556, 2016. URL http://arxiv.org/abs/1606.05556.

[46] D. Kraft. A software package for sequential quadratic programming. Technical report, DLR German Aerospace Center — Institute for Flight Mechanics, Koln, Germany, 1988.

[47] R.M. Hicks and P.A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–412, 1978. doi: 10.2514/3.58379. URL https://doi.org/10.2514/3.58379.

[48] T. Economon, D. Mudigere, G. Bansal, A. Heinecke, F. Palacios, J. Park, M. Smelyanskiy, J. Alonso, and P. Dubey. Performance optimizations for scalable implicit rans calculations with su2. *Computers Fluids*, 129, 2016. doi: 10.1016/j.compfluid.2016.02.003.

[49] P. Gomes, T.D. Economon, and R. Palacios. *Sustainable High-Performance Optimizations in SU2*. AIAA Scitech 2021 Forum, 2021. doi: 10.2514/6.2021-0855. URL https://arc.aiaa.org/doi/abs/10.2514/6.2021-0855.

[50] V. John. Multigrid methods. Technical report, 2014. URL https://www.wias-berlin.de/people/john/LEHRE/MULTIGRID/multigrid.pdf.

[51] W.H. Mason. Transonic aerodynamics of airfoils and wings. Technical report, Virginia Tech, 2006.

Acknowledgment is given to ChatGPT for grammar and readability improvements for this report.

Appendix

A Domain of Wing and Airfoil

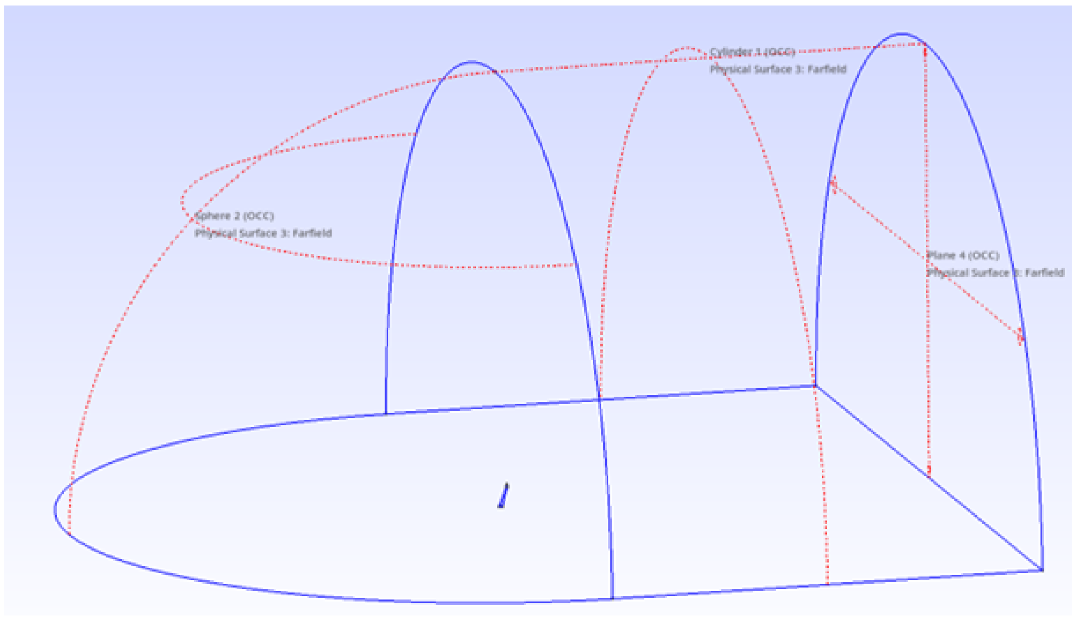

Cylinder 1 (OCC)
Physical Surface 3: Farfield
Sphere 2 (OCC)
Physical Surface 3: Farfield
Plane 4 (OCC)
Physical Surface 3: Farfield

Figure 28. Wing domain

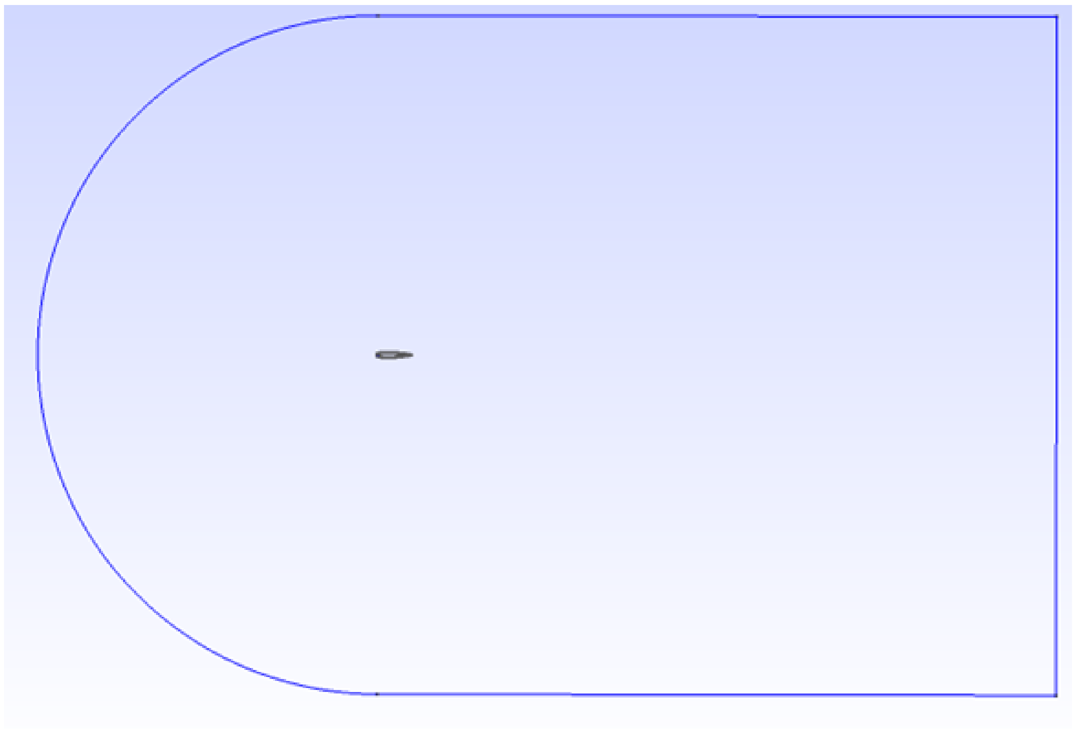


Figure 29. Airfoil domain