THE UNIVERSITY OF QUEENSLAND

Bachelor of Engineering Thesis

A Parametric Grid Generation Tool for the Aerodynamic Evaluation of Hypersonic Vehicles

Student Name: Alexander WARD

Course Code: MECH4500

Supervisor: Dr. Ingo Jahn

Submission date: 28th October 2016

A thesis submitted in partial fulfilment of the requirements of the Bachelor of Engineering degree in Mechanical and Aerospace Engineering

UQ Engineering

Faculty of Engineering, Architecture and Information Technology

Acknowledgements

During this thesis there have been numerous people that have put in time and effort or otherwise contributed to the present work. My thanks are directed towards:

First and foremost, my supervisor Dr Ingo Jahn, for giving me the opportunity to work on such an interesting project outside my skillset. Your help and suggestions played an important role I should have taken further advantage of.

Dr. Rowan Gollan, for your expertise and enthusiastic willingness to help me with running the Tinaroo simulations. Without your help, the full vehicle simulations would not have been possible.

Mr Sholto Forbes-Spyratos, although not your area of expertise, those few hours of your time provided motivation and generated interest in me for days. Good luck!

Thanks should also extend to helpful EAIT IT staff who offered guidance on using the computer clusters.

Finally, to my family and friends, thanks for your subconscious (and deliberate) support, be it pretending to listen to what I'm saying or distracting me.

Abstract

A large proportion of modern satellites are smaller, cheaper and short-lived relative to their historical counterparts. Dedicated launch of these is associated with prohibitive financial burdens and with current rocket technology operating close to theoretical limits, further improvement is limited. In light of this market potential, a reusable, staged launch vehicle employing air-breathing scramjet propulsion has been proposed. A configuration of this type is expected to reduce launch costs and improve the responsiveness of access to space.

Accurate prediction of aerodynamic performance and subsequent geometric optimisation are critical to the success of such a vehicle. Furthermore, modelling of the trajectory requires a preferably comprehensive database of aerodynamic coefficients. Current analysis methods are limited in their ability to predict certain behaviour and model low speed flow.

This paper details the development of a parametric grid generation tool for the analysis of hypersonic winged cone vehicles. The vehicle geometry is based on a concept developed for the National Aerospace Plane and is made up of a spherically blunted, conical forebody, cylindrical fuselage and truncated conical boat tail. The wings are swept delta wings.

The structured grid is made up of 265 blocks and is generated using e3prep, the pre-processor and grid generation tool for The University of Queensland's compressible CFD code, Eilmer. Significant modification of vehicle geometry is possible while retaining acceptable grid quality. However, poor quality issues present themselves at swept geometry.

In addition to a demonstration of the parametric nature of the tool, results of a proof of concept simulation of the complete vehicle are given. A short forebody study detailing the analysis of a widened, cambered forebody with a flattened windward side was completed as further demonstration. The modified geometry was found to improve the inlet onset flow and increase precompression. These results are presented without formal verification and validation and hence considered provisional.

Recommendations are made on the continued development of the tool. These primarily address the quality issues preventing full functionality of the tool and include utilising sections of unstructured cells to form a patched grid and combining the current work with GridPro for the generation of a highly smoothed grid. The use of OpenFOAM for subsonic simulation is also discussed along with increased parallelisation of the simulations to decrease runtime.

Table of Contents

List of Figures

List of Tables

Nomenclature

Abbreviations and Terms

2D	Two dimensional
3D	Three dimensional
AMARA	pArametric Mesher for AcceleratoR Analysis
AR	Aspect Ratio
Block	A 3D (or 2D) section of the domain which is subdivided into cells
Cell	A discrete volume within which flow properties are constant, collections of which make up the fluid domain.
CFD	Computational Fluid Dynamics
LEO	Low Earth Orbit
NACA	National Advisory Committee for Aeronautics (now NASA)
NASA	National Aeronautics and Space Administration
Parameter	An independent variable which may fall within a defined range or in the case of a parameterisation function, zero to one.
Parameterisation	The description of a geometric object in terms of its parameters
RLV	Reusable Launch Vehicle
SPARTAN	Scramjet Powered Accelerator for Reusable Technology AdvaNcement
UQ	The University of Queensland

1 Introduction

A reusable, staged launch vehicle employing air-breathing scramjet propulsion is a promising candidate to reduce significantly launch costs of small satellites – the market for which is expected to increase. However, the development of such a vehicle is not without significant difficulty – success depends heavily on the accurate prediction of its aerodynamic performance and subsequent optimisation. The current works develops a block-structured, parametric CFD grid for a generic winged cone hypersonic vehicle.

1.1 Research Context

Owing to the development of small size, low-power electronics, a large portion of modern satellites are smaller, cheaper and offer comparable performance at relatively low altitude. Many satellites, which once weighed of the order of 1000 kg, may now be as little as 100 kg (Baker & Worden, 2008). Almost all small payloads are launched on a 'ridesharing' basis, piggybacking with large satellites, rendering the small satellite operator with no control over orbit altitude, inclination and launch date. This suggests significant market potential for a low cost launch architecture exists which:

- is designed specifically for the launch of small payloads at low cost;
- offers quick turnaround between missions - desired for the relatively short life cycle of small modern satellites, and finally;
- gives flexibility to the customer in terms of orbit altitude and inclination through the dedicated launch of small satellites.

After decades of development, current rocket technology is operated close to theoretical limits. Further cost reduction is therefore expected from novel technologies such as reusability and air-breathing propulsion combined with aircraft-like performance (Bowcutt & Smith, 2012; Preller, Smart & Schutte, 2016).

This is no trivial task however, with the task of designing and testing a functional vehicle of this type described as one of the most significant challenges of current times in the field (Hirschel & Weiland, 2011). One needs only to consider the Space Shuttle Orbiter, arguably the only reusable launch architecture ever operated and its accompanying issues, for confirmation.

With extensive expertise in the ground testing and simulation of hypersonic flows in the context of general bodies and scramjet engines, The University of Queensland has cemented itself as an academic leader in the field. In 2009, Smart and Tetlow investigated a rocket-scramjet-rocket system for the launch of small payloads. In 2010, Jazra proposed a refined and optimised second stage - Scramjet Powered Accelerator for Reusable Technology AdvaNcement or SPARTAN. His multidisciplinary analysis environment included the panel code aerodynamics module, HYPAERO. The code has been subsequently updated to include control surface deflections and extend potential geometry through the efforts of Preller and Smart (2014; 2015). Although an excellent conceptual design and evaluation tool, this code is nonetheless limited especially once design progresses from conceptualisation.

This thesis is concerned with the development of a computational fluid dynamics (CFD) grid generation tool for the evaluation of hypersonic vehicles. In particular, the need to model SPARTAN's flyback from payload delivery is identified as recommended by Preller, Smart and Schutte (2016). The tool should be inherently parametric in nature, allowing the investigation and optimisation of geometry.

1.2 Aims of the Thesis

The main deliverable is the code itself. This task may be further broken down into four main sequential aims.

Develop an appropriate three dimensional blocking strategy

The first step to the tool is the development of a blocking strategy for a generic hypersonic winged cone vehicle geometry.

This should include the features and parameters of the existing SPARTAN geometry at a minimum, allow the entire flow field to be captured and lead to acceptable quality cells.

Construct a parametric grid for the analysis of generic hypersonic vehicles

Using the developed blocking strategy a parametric grid will be constructed. This grid will be capable of forming a range of vehicles. It should be of acceptable quality against the essential quality metrics.

Exercise the framework on SPARTAN

As the tool will be used extensively on the SPARTAN geometry, it is desired to generate a grid which matches the baseline SPARTAN. This grid will also be the subject of a proof-of-concept simulation.

Exercise the framework on a baseline and modified forebody

The forebody is a critical component to a hypersonic vehicle of any configuration. Much research exists in the literature however the SPARTAN forebody currently remains fully conical. A modified geometry is presented and analysed as a demonstration of the tool.

1.3 Thesis Scope

The blocking strategy was developed for the inclusion of the SPARTAN vertical empennage and control surface deflection. The grid however does not include these due to quality issues present in the use of an entirely structured grid. Furthermore, modelling the complex engine inlet geometry was deemed unfeasible and out of scope.

Given the prior investment into winged cone type vehicles and requirement of modelling SPARTAN, the grid geometry was limited to this configuration. Further parameterisation of the geometry was out of scope.

The developed framework will ultimately be applied across a range of flight conditions and geometries. Owing to the runtime required for a full scale vehicle simulation however, only a proof-of-concept at a single flight condition is presented. Although a major computational task, running similar simulations at different conditions does not add to the breadth of the thesis. Validation of the results is also left to future work.

1.4 Structure of Thesis

The thesis will begin with a review of the relevant literature in section 2. Information on launch vehicles and their design, grid generation and solver codes is presented. Following this, a 'big picture' view of the approach is given (section 3) before the geometry, construction and parameterisation of the grid is introduced in section 4. Here each group of blocks, from nose to tail, is described including geometric and grid control parameters, as well as the functions behind the block construction.

Next, a discussion of the quality of the developed grid and geometric limitations; simulations of the full vehicle and a demonstration of its parametric ability as a design tool in the context of the forebody are given in section 5. These simulations are without formal grid independence or validation studies and are therefore considered provisional or proof-of-concept.

Section 6 addresses the identified issues of the developed tool through recommendations on future work. Finally, conclusions on the grid and work completed are given in section 7 before the thesis closes with a summary of recommendations for future improvement and applications of the tool.

2 Literature Review

This section aims to provide background information: 1) on some historical and modern hypersonic vehicles and some specific design issues; 2) on CFD meshing and simulation codes and finally 3) for the justification of decisions made during the thesis. On terminology, in most cases ‘grid’ and ‘mesh’ may be used interchangeably. However, ‘mesh’ is the more general term whereas ‘grid’ implies a structured placement of nodes.

2.1 Launch Vehicle Concepts

A launch vehicle is some sort of system to deliver payloads into space. Historically, this has always been achieved with rockets however growing economic pressure. Single or multi stage configurations employing a range of propulsion systems have been proposed over the past decades. This continues today, with growing interest in the next generation of launchers (Pezzella, 2011). Of primary interest to the present study is a concept from the National Aerospace Plane program – the single stage winged cone vehicle or WCV (Figure 1) (Shaughnessy et al., 1990).

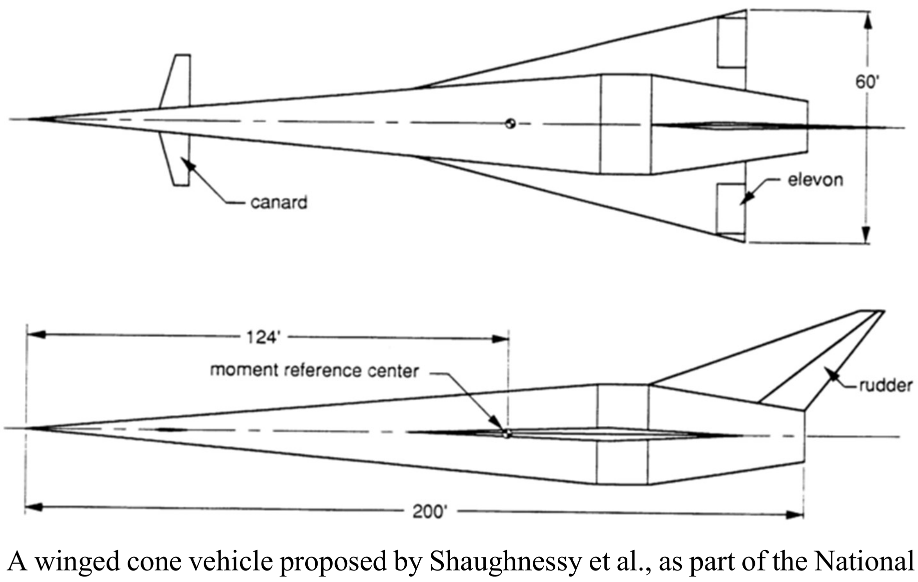


Figure 1. A winged cone vehicle proposed by Shaughnessy et al., as part of the National Aerospace Plane project (Shaughnessy et al., 1990).

2.1.1 Reusable, Air-Breathing Launch Vehicles

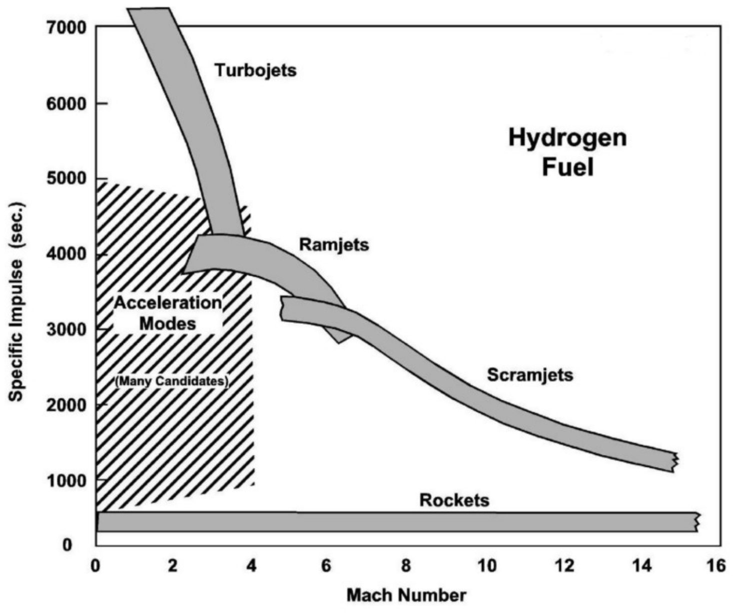


Figure 2. Specific impulse of different propulsion systems operating at different Mach numbers. The width of the bars represents the spread in performance (NASA).

A promising avenue of development for the cost reduction of access to space, is that of reusable launch vehicles (RLVs) (Ahmad et al, 2011). The ability to retain and reuse large portions of a launch vehicle promises to reduce costs. Furthermore, by exploiting air-breathing engines the payload mass fraction may be increased due to not accelerating an oxidiser (Dujjaric et al., 1997). Additionally, the improved specific impulse make them a highly attractive option (Figure 2)

Drawbacks associated with air-breathing propulsion systems should also be noted. These include high aerodynamic heating due to the relatively high required dynamic pressure and added complexity of engine-airframe integration, as raised by Bertin and Cummings (2003). Accurate prediction of the aerothermodynamics of hypersonic vehicles, including aerodynamic coefficients is critical to the design (Gnoffo et al., 1999).

For a reusable vehicle to be economically viable, rapid turnaround is required (illustrated in Figure 10). That is, time between missions should be on the order of weeks or even days.

This requires the vehicle to return to its starting point requiring good aerodynamic performance. This places further importance on the need for extensive aerodynamic evaluation.

2.1.2 Single-stage and Multi-stage to orbit

Single stage to orbit RLVs offer simplicity and convenience in their rapid turnaround and high productivity capability, however there are performance shortcomings. Their justification is generally made on the basis of an "airline-like economic model" – distribute very high vehicle cost over multiple missions (Ahmad et al., 2011). However, single-stage systems are flawed fundamentally when compared to staged systems.

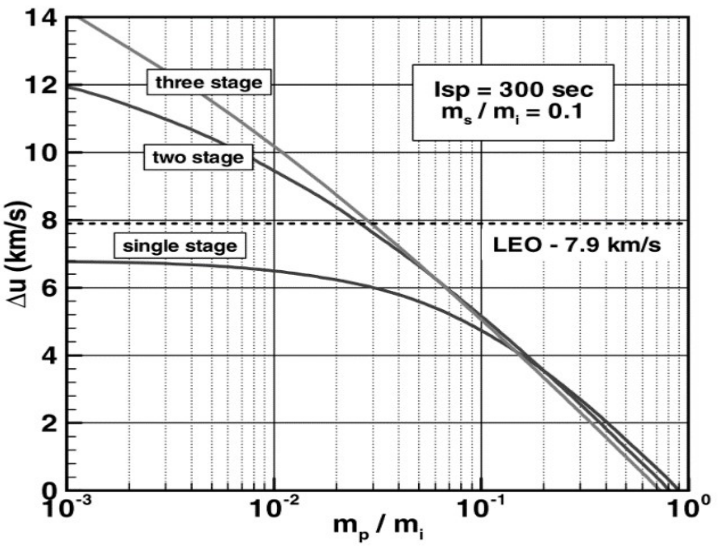


Figure 3. Payload mass fractions for staged system calculated using the rocket equation (Smart & Tetlow, 2009).

Firstly, for a given Δv, (the impulse required to propel a mass to orbit; LEO for example), by staging the vehicle the payload mass fraction may be increased. This may be shown by plotting the ideal Tsiolkovsky rocket equation, neglecting gravity and drag, as shown in Figure 4. This is explained by the fact the vehicle does not need to accelerate as much structural mass for as long (Smart & Tetlow, 2009).

Secondly, multi-stage systems lend themselves well to hybrid propulsion, such as air-breathing - rocket systems. Air-breathing propulsion may only operate over a portion of the trajectory (within the atmosphere) (see section 2.1.1). A staged vehicle allows the separation of the inert air-breathing system that would otherwise be accelerated to orbital speeds (Bertin & Cummings, 2003) Figure 5 illustrates the advantages of hybrid propulsive systems in terms of increased payload mass fraction.

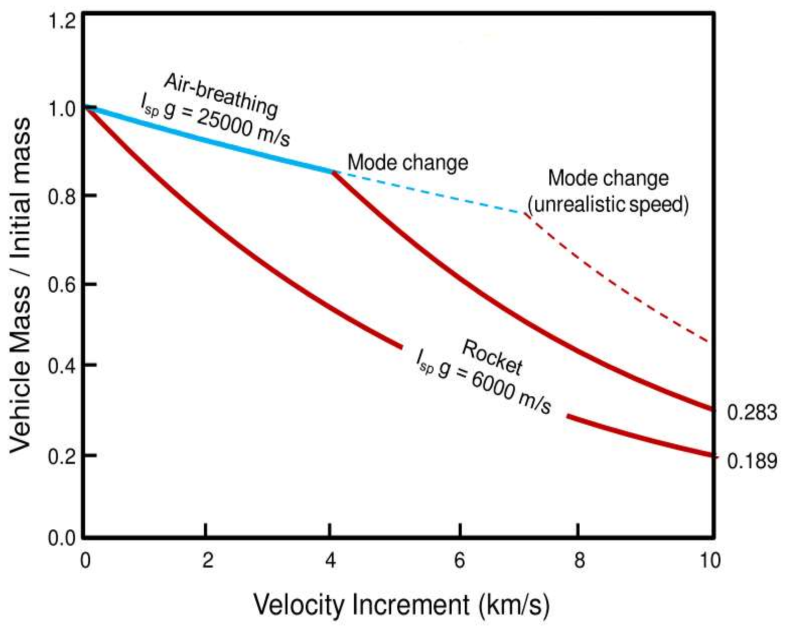


Figure 5. Velocity increment (Δv) as a function of mass fraction (Ahmad et al., 2011).

2.2 Hypersonic Air-breather Design

The advantages of a multi-stage, air-breathing RLV are clear. However, the design of vehicles to operate at such high speeds represents one of the biggest challenges of modern aerospace engineering. Several of the key issues will be presented here for background on the geometry choices of the developed CFD grid. Notable types of vehicles will also be discussed with a final emphasis on SPARTAN, the vehicle under development at The University of Queensland.

2.2.1 Engine-Airframe Integration

In contrast to the vast majority of modern subsonic airlines and small aircraft, super- and hypersonic vehicles tend to have extremely tightly integrated propulsion systems. There are a few notable exceptions such as Reaction Engines' Skylon[1] and the SR-71[2]. The advantages of a propulsion system tucked in to the fuselage of note are: 1) the possible forebody precompression and potential for flow alignment,

[1] http://www.reactionengines.co.uk/space_skylon.html
[2] http://www.lockheedmartin.com.au/us/100years/stories/blackbird.html

and 2) the reduction in drag due to the likely increase in slenderness and decrease in frontal area (Hirschel & Weiland, 2009). This design choice simplifies the aerodynamic modelling of these vehicles as the geometry essentially becomes a wing-body or wing-body-tail model. A schematic of a generic airframe integrated engine and vehicle is shown Figure 6.

.

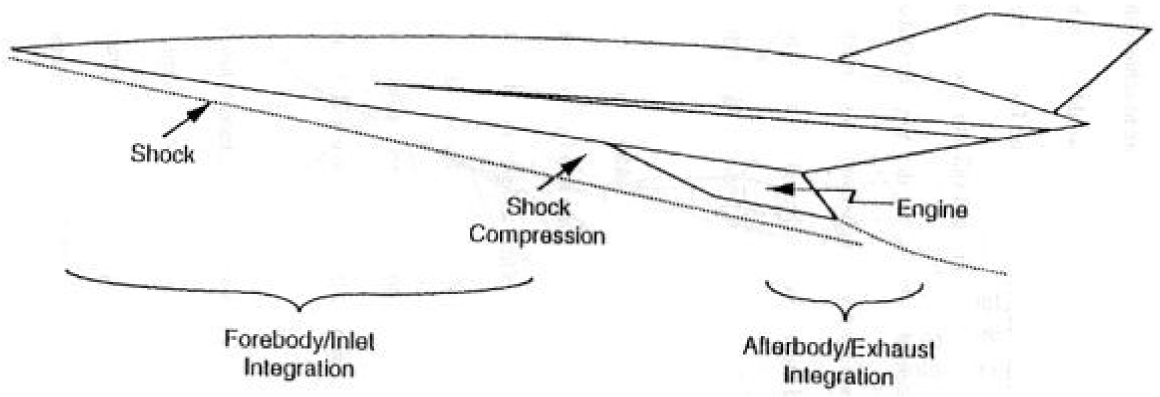


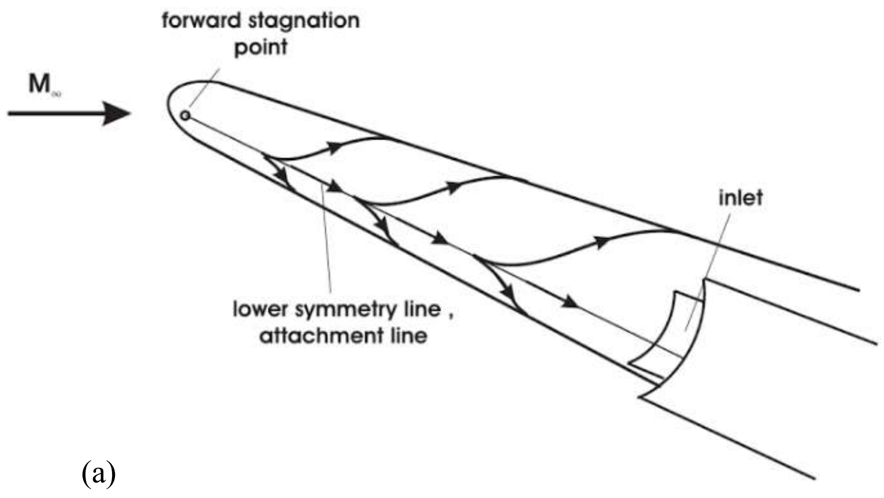
Figure 6. Schematic of a hypersonic vehicle showing the bow shock formed by the nose as well as forebody and aftbody nomenclature (Hallion, 1998).

2.2.2 Forebody Design

The forebody design is a key issue of a vehicle’s design. It is dictated by not only aerodynamic performance requirements but also structural, internal volume and centre of gravity demands (Duveau et al., 1999). The inflow to the engine inlet must be as uniform as possible for optimum engine performance. Fully conical forebodies return a well-defined inlet flow and fixed precompression. However, at an angle of attack, these benefits are lost due to the resultant 3D flow field around the body (Hirschel & Weiland, 2009). This is illustrated in Figure 6.

Owing to the criticality of drag, the nose bluntness is also an important parameter. The resulting bow shock must a) not interact with the engine cowl (shock-on-lip condition) avoiding detrimental effect on the inlet performance and vehicle wave drag and b) at the largest Mach number, just envelop the propulsion system to avoid unnecessary wave drag (Bing, Chun-lin & Liang-xian, 2015). Finally, another important characteristic is the role the under- or windside of the forebody ramp plays in the inlet precompression (Hirschel & Weiland, 2009).

Using the forebody in this manner however is not without disadvantages. Firstly, the forebody boundary layer will be ingested, potentially upsetting the inlet flow stability. Secondly, as the windside becomes a flatter surface, internal packaging becomes less efficient as this geometry is less suited to pressurised tankage. Furthermore, this cross-section has a poorer surface area-to-volume ratio which leads to detrimental effects on drag, structural mass and aeroshell construction. Finally, as significant compression and expansion surfaces, the forebody and boat tail contribute to the lift and pitching moment with subsequent strong effects on vehicle longitudinal trim (Varvill & Bond, 2003).



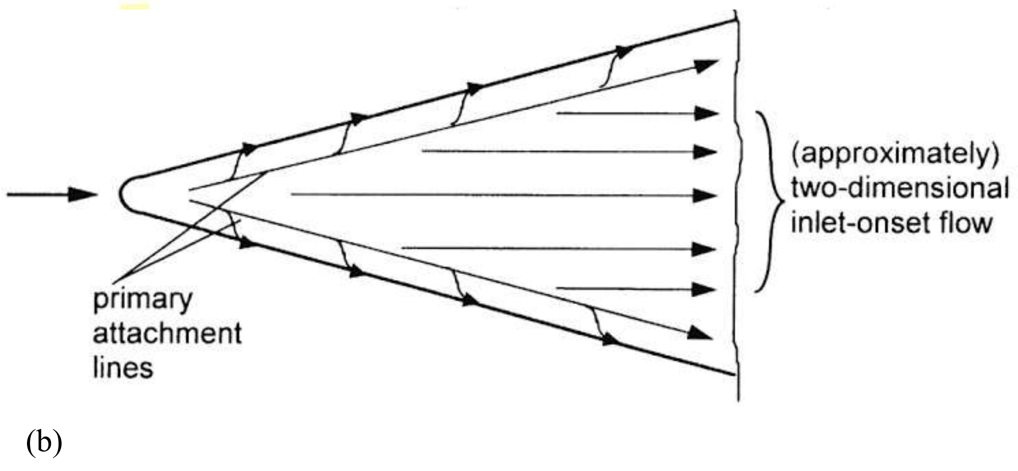


Figure 6. Examples of conical and flat bottomed, wide forebodies and representative streamlines. (a) The conical forebody flying at an angle of attack results in a divergent streamline pattern thus reducing the effective inlet capture area. (b) A flat bottomed forebody returns a more favourable inlet flow. This is indicative of a good design. Adapted from Hirschel and Weiland (2009).

The ability to introduce bluntness, aspect ratio, both longitudinal and transverse camber and combinations of the above to a standard conical forebody would allow the vast majority of forebody geometries to be formed. Leading on from this, a wide conical forebody with a flattened bottom and blunt nose will result in a desirable flow topology at both a large Mach number and angle of attack range (Hirschel & Weiland, 2009). It follows that the ability to model this sort of geometry is a key goal of the thesis.

2.2.3 Waverider vehicles

A wide, flat-bottomed (or even transversely concave) forebody geometry begins to be reminiscent of a waverider type geometry. These are a special sort of vehicle, designed to 'ride' the leading edge shock wave which differ substantially from those seen previously Figure 1. The excellent lift-to-drag ratio means the pure waverider geometry is one of the most promising aerodynamic configurations (Huang et al., 2009). The lower side of a waverider is determined from the stream surface attached to the prescribed leading edge. As the pressure downstream of a shock is increased, this surface generates lift and accompanying wave drag. The upper surface tends to be a free-stream surface. Coupled with a sharp leading edge and no deflection angle, this surface has therefore no contribution to the aerodynamic coefficients. Ideal L/D ratios are expected to be around $7 - 8$, however, the inclusion of viscous effects and blunt edges reduce these by between 40 and 20% (Hirschel & Weiland, 2009).

The modifications required for a practical shape are not without detrimental performance effects, namely in terms of L/D (Ferguson, Dasque & Dhanasar, 2015). In particular, the difficulty of incorporating significant internal volume, means the winged cone type of hypersonic airbreathing launch vehicles is considered to be the more practical option (Hirschel & Weiland 2009). Nonetheless, their theoretical high aerodynamic performance continues to foster modern development (Ferguson, Dasque and Dhanasar, 2015).

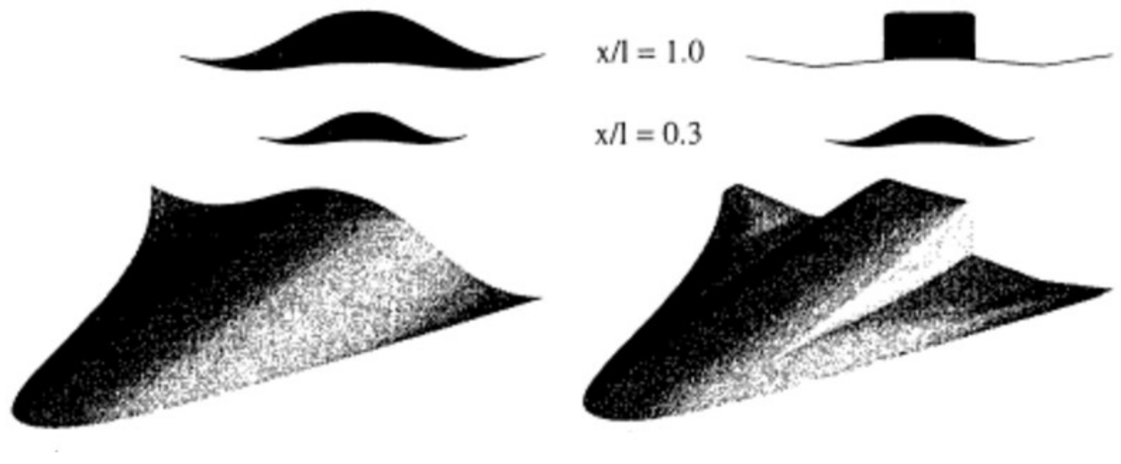


Figure 7. Comparison of an analytically designed and practical waverider shape. (a) shows a generic viscous optimised waverider designed for Mach 12, (b) is a modified version of the shape to include a sharp trailing edge (reduce base drag), some internal volume (propulsion and payload) and two plane surfaces at the rear of the vehicle for the addition of control surfaces (straight hinge lines) (Eggers et al., 1995).

2.2.4 SPARTAN

Closer to home, the use of a staged, rocket-scramjet-rocket system was proposed and examined by Smart and Tetlow in 2009. The second stage, known as SPARTAN (Figure 8) is a scramjet powered, reusable, winged cone vehicle based on the design by Shaughnessy et al. (Figure 1). Owing to internal volume concerns, this architecture over a waverider type vehicle was chosen (Smart & Tetlow, 2009). The most recent work by Preller, Smart and Schutte (2016) show promising results with the three stage system boosting a payload of 257.4 kg payload into low earth orbit at a favourable payload mass fraction of 1.26%.

The vehicle has a conical fuselage with four tightly integrated scramjet engines. The swept delta wings have a diamond cross section for lift generation (Preller, Smart & Schutte, 2016). Side and top views are given in Figure 9. An illustrative schematic of a SPARTAN mission and trajectory is shown in Figure 10. In particular, the return to base or flyback section of the mission is key to the system's practicality and requires good aerodynamic performance.

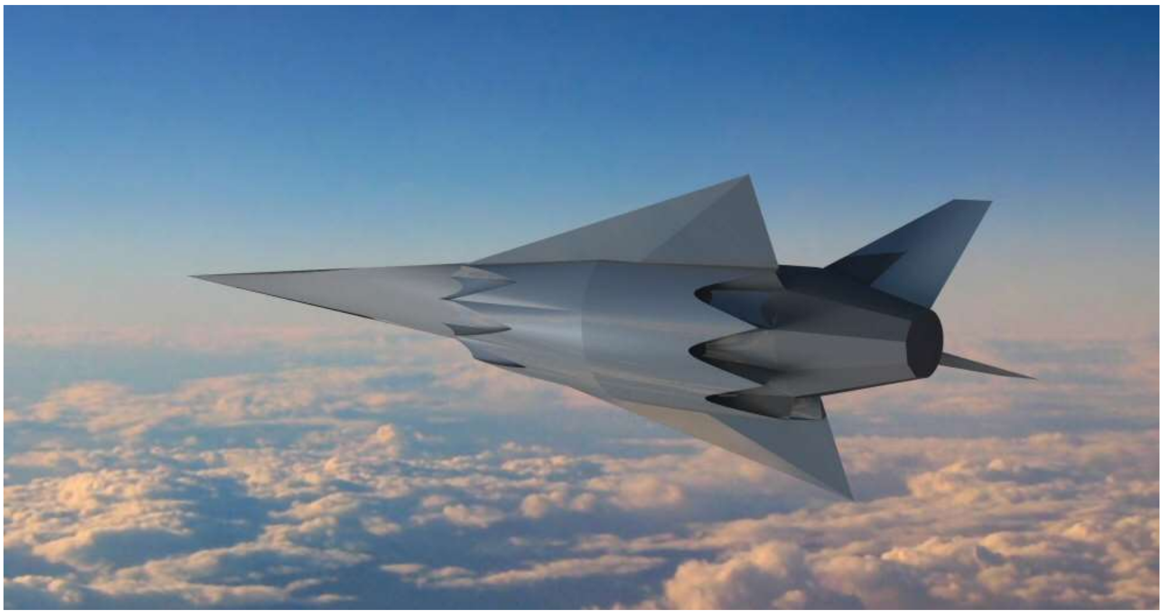
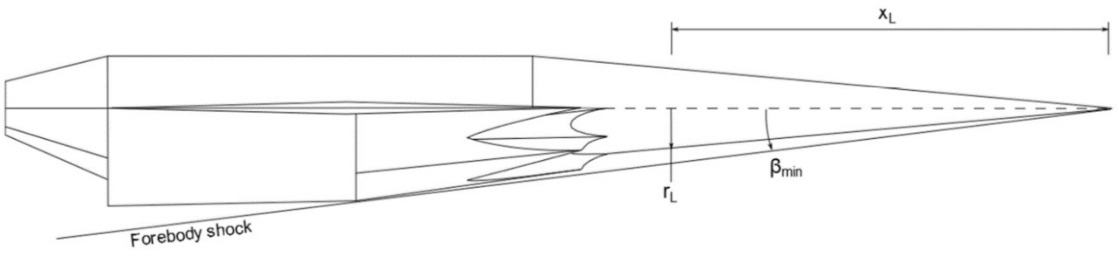
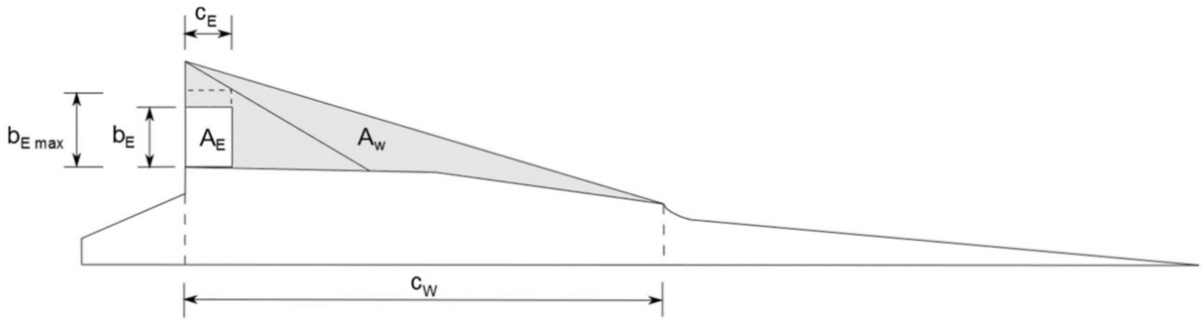


Figure 8. An artist's impression of SPARTAN.



(a) Side view



(e) Plan view

Figure 9. Plan and side views of SPARTAN. The side view illustrates the forebody shock and nomenclature of the local radius at position . The plan view shows elevon positioning and sizing (Preller, Smart & Schutte, 2016).

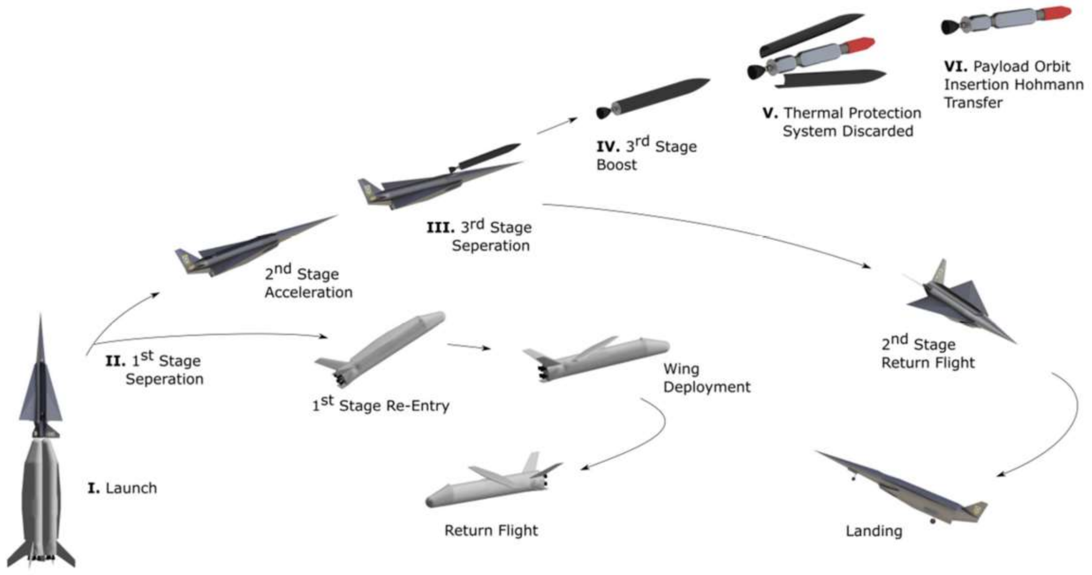


Figure 10. Three stage rocket-scramjet-rocket mission employing SPARTAN as the second stage showing the required flyback of the vehicle.

2.3 Grid Generation

The construction of a grid around even a simple wing-body configuration is described as no trivial matter (Eriksson, 1982). Furthermore, the numerical solution to the governing equations presents a formidable task which in turn places requirements on the mesh quality (Eriksson, 1982). Although the techniques and tools available for grid generation have improved markedly over the past decades, the geometry definition and grid generation stages in a conceptual CFD study remain the biggest bottleneck in the process (Bertin & Cummings, 2006). The review presented here generally refers to 3D situations however may be roughly extended to 2D.

2.3.1 Meshless Methods

Meshless methods are a wide class of techniques which use either surface panels or clouds of points over which governing equations may be discretised. Meshless methods are commonly used when the 3D geometry is very complex as the discretisation is generally simpler and easier (Katz, 2009).

One such panel method of evaluating the aerodynamic performance of hypersonic configurations is based on strip theory. This method, has been applied to super- and hypersonic flows extensively, divides a 3D surface into 2D longitudinal panels which from strips marking the flow-path over a vehicle's surface. The surface pressure and Mach number distribution may be calculated using inclination methods from the idea that the properties on a panel depend only on the panels before it. An example of a surface panel grid is shown in Figure 11. This will be discussed further when HYPAERO, UQ's panel code is introduced (Jazra, Smart, 2009).

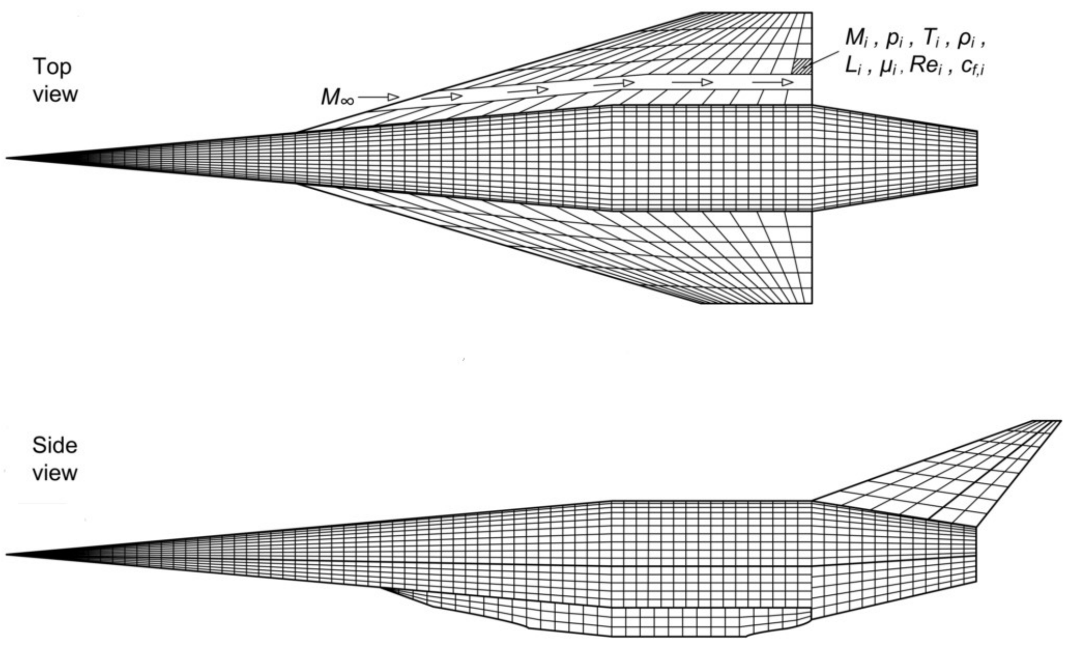

Top view
M_∞
M_i, p_i, T_i, ρ_i,
L_i, μ_i, Re_i, $c_{f,i}$
Side view

Figure 11. Top and side views of a HYPAERO surface grid. The strips are defined in a stream-wise direction from the nose of the vehicle. Properties shown in the top view are constant in each panel. Note these grids are shown assembled – each component is analysed separately before the aerodynamic coefficients are resolved to a common fixed coordinate system (Jazra, 2010).

The most prevalent, modern meshless codes applied to hypersonic vehicles are the Aerodynamic Preliminary Analysis System, APAS (Anderson, 2006), and more recently NASA's CBAERO (Kinney, 2004). CBAERO demonstrates good agreement for lift, drag, and surface heat transfer with wind tunnel data on full-vehicle test cases of the X-33, lifting body.

Time for computation for the X-33 on a single CPU range from a few seconds to around 160 seconds (Kinney, 2004). APAS has been used for evaluations of many hypersonic launch and re-entry designs (Cruz & Wilhite, 1998)

These meshless methods are well suited to the preliminary design phase, where a large number of results are desired at low computational expense with acceptable error. For higher fidelity analyses however, 3D Navier-Stokes CFD simulations are required (Robinson & Martin, 2008). Furthermore, with modern increases in computer technology however, Navier-Stokes numerical techniques are becoming increasingly common (Hirschel, 2005).

2.3.2 Structured and Unstructured Grids

A structured grid is fundamentally defined as one in which node connectivities have a fixed pattern and is generally made up of hexahedral (quadrilateral in 2D) cells (Mavriplis, 2008). Therefore, each node has six associated edge connectivities. The cell data are stored in a computationally efficient 3D $(i \times j \times k)$ array, where the indices correspond to a cell's spacial position meaning neighbouring cells are known implicitly $(i_{+1} \times j_{+1} \times k_{+1})$. This format requires six-sided cell domains, or blocks, be defined, with matching cell counts on opposite sides (Chawner, 2013). Conversely, an unstructured mesh is without these constraints. Varying cell numbers and cell types (triangular, pentagonal for example) may be employed.

With regards to comparison, there are advantages and disadvantages to both. Unstructured meshes have a clear ability to conform to complex geometry and allow cell counts to vary between opposite block boundaries. Furthermore, the construction process is generally highly automated. However, this relinquishes some control over cell size, distribution and orientation (Mavriplis, 2008).

A structured grid will, in general, offer increased control over the cells. From the boundary definitions and discretisation, the exact grid specified by the user is returned. Coupled with clustering, a highly controlled grid may be formed. This control extends to the orientation of cells to the predominant flow direction and resolution of areas of strong gradients.

CFD codes typically converge more reliably and achieve more accurate results on aligned grids (Tannehill, Anderson & Pletcher, 1997).

Improved grid quality is also frequently achieved for simple geometry (Chawner, 2013). The domain may be made more manageable by dividing it into six-sided blocks, as demonstrated by the example multi-block structure of Figure 12. This has the secondary effect of allowing the computational load to be spread easily between cores. Finally, due to their efficient computational structure, a structured code will in general run faster (Hirschel, 2005).

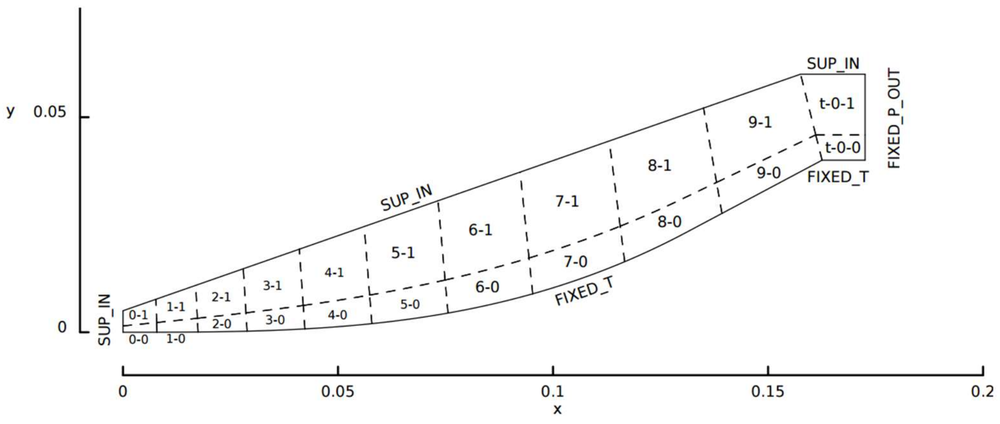


Figure 12. An example of a 2D multi-block structure taken from the Eilmer3 userguide. Although in this example the domain could have been defined with only two blocks, 22 have been used. This allows the computational load to be easily shared across more CPU cores (Jacobs et al., 2015).

2.3.3 Grid Quality

It is well known know the quality, resolution and orientation of the grid at a particular location can have large effects on the flow predictions (Bertin & Cummings, 2006). High-quality grids are needed to capture adequately discontinuities and separation (Pirzadeh, 2010). The following refers to structured grids, although to an extent most ideas may be extended to unstructured. Hexahedral cell quality is frequently measured through geometric criterion such as skewness, smoothness, orthogonality and aspect ratio (Knupp, 2008). Evaluating these measures individually is not sufficient to characterise the overall grid quality, rather they should be considered simultaneously (Alter, 2004).

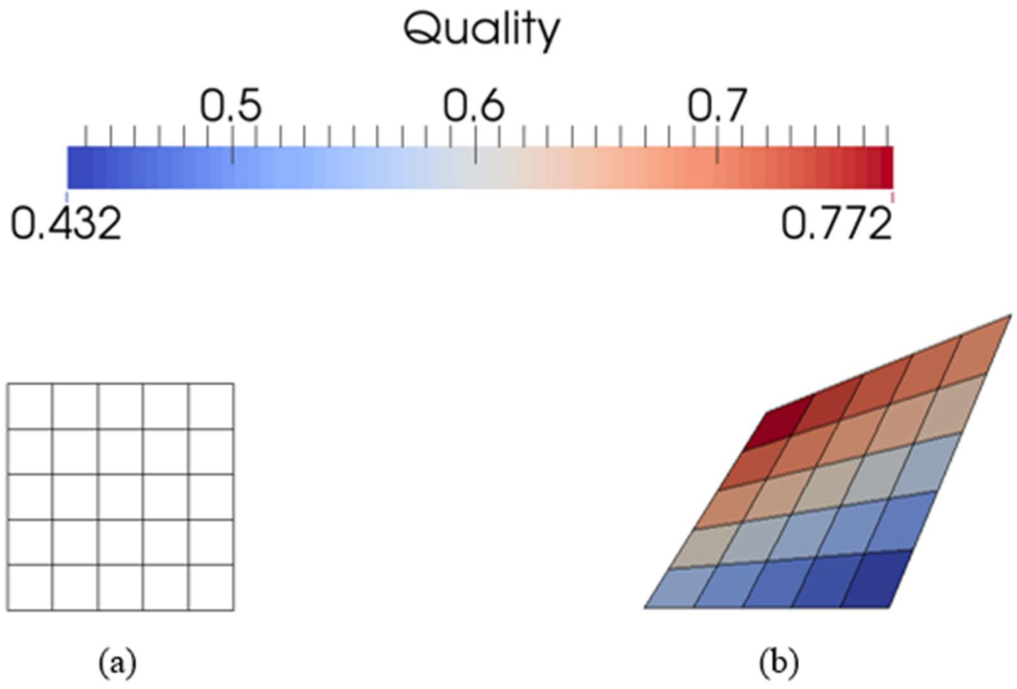
Skewness is a measure of the squareness of an individual cell and is defined below. Depending on the specific problem, a maximum of 0.85 is generally deemed acceptable (Knupp, 2008).

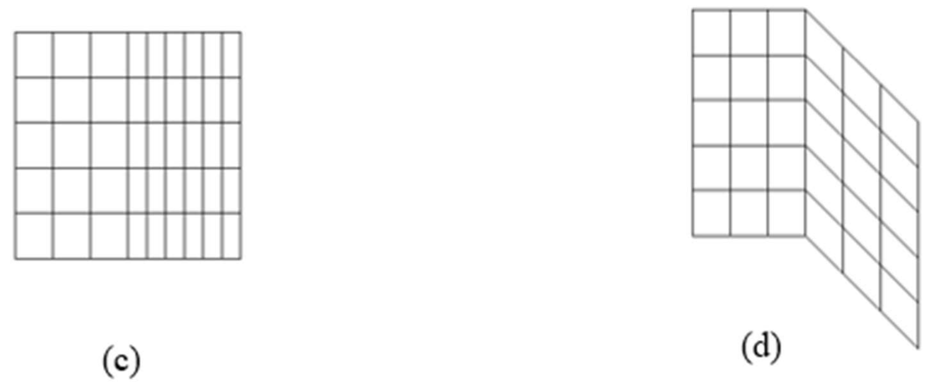
$$\text{Skewness} = \max\left[\frac{\theta_{max} - 90}{90}, \frac{\theta_{max} - 90}{90}\right]$$

Smoothness, or stretching, refers to the size a cell relative to its neighbours where poor smoothness is characterised by abrupt changes. The orthogonality of adjacent cells becomes important due to the error associated with the reconstruction of property fluxes when linearly interpolating between cell centres rather than correctly resolving to the face centre.

Finally, aspect ratio refers to the ratio of the shortest to longest cell side length. Typically cells should be as close to uniform as possible ($AR < 1.2$) with the exception of regions of anisotropic flow, such as within a boundary layer where gradients close to the wall are small (Alter, 2004).

As a further explanation, examples of all these quality metrics are presented Figure 13. Although these are good indicators, the true measure of grid quality is in the validation of the solution and depends on the specific physical problem (Papadopoulos et al., 1999). Figure 13(f) is the same boundaries as (b) however it demonstrates the ability of an area-orthogonal generation method. The method tries to keep cells orthogonal to edges and keep cell areas constant across the surface. It does this by interpolating over a defined background mesh, in this case $8 \times\ 8$ (versus actual cell counts of 5×5).





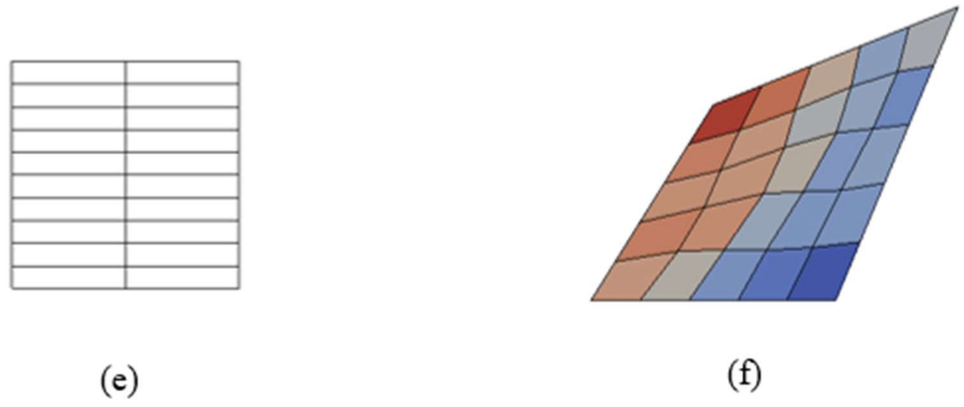


Figure 13. Examples showing (a) a uniform, orthogonal grid; (b) a skewed grid coloured by quality; (c) poor smoothness; (d) poor orthogonality; (e) high aspect ratio; and (f) the same skewed grid of (b) however this time using `AOPatch`. The maximum skewness of 0.772 in (b) was reduced to 0.740 in (f). These images were generated using e3prep and visualised with ParaView.

2.3.4 Tools

GridPro[3]

GridPro is a commercial multi-block structured grid generation package which provides a high level of automation. The attractiveness of GridPro lies in its advanced smoothing scheme which returns very high grid quality. The generation methodology is as follows; firstly, surfaces which define the geometry and computational are imported or generated. The general topology of the domain is then constructed and written to file in GridPro's Topology Input language, TIL. The TIL file is a logical record specifying the desired geometry. It is from this file that GridPro then generates its smoothed volume grids. An example of a complex grid is given in Figure 14.

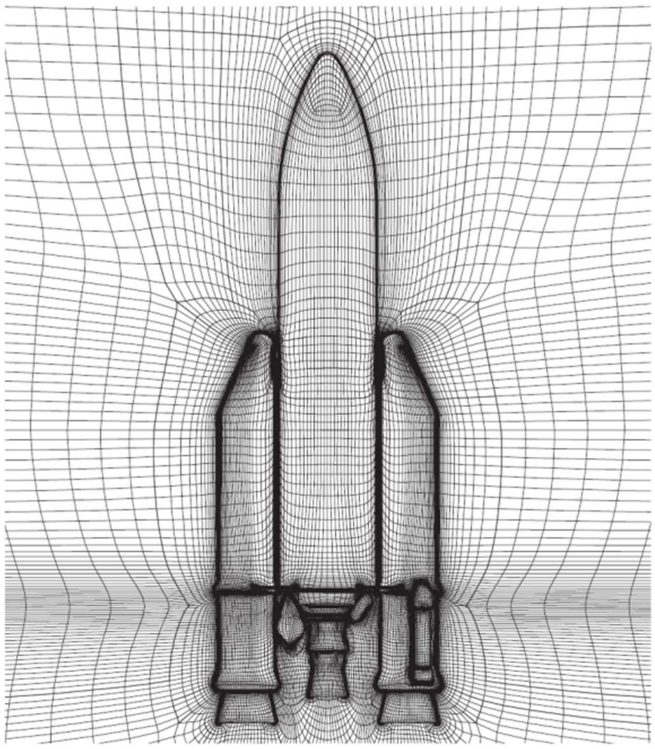


Figure 14. An example of a smoothed grid generated using GridPro of an Ariane rocket and accompanying boosters (GridPro).

[3] http://www.gridpro.com/

e3prep.py[4]

e3prep.py is the preparatory Python program associated with Eilmer (introduced in section 2.4.3). It sets up the simulation parameters and generates a block-structured grid with accompanying initial, boundary, and inflow conditions. It is a rudimentary but powerful program due in part to the functionality added with python scripting. The grid is generated with fundamental geometric objects such as vectors (nodes), paths (lines, arcs, splines) and surfaces (planes, bounded surfaces). Volumes are then generally formed through sets of either six parametric surfaces or 12 paths (Jacobs et al., 2015).

Unlike GridPro, e3prep offers significant parametric ability, primarily through the python functionality. This advantage however is a trade-off with GridPro's superior cell quality and automation. An example of an e3prep grid and Eilmer simulation is shown in Figure 15. This spherically blunted cone is constructed by revolving an arc blended into a straight line about the *x* axis.

e3post is Eilmer's rudimentary but versatile post processing program. It now contains a multitude of options including extracting specific data such as pitot pressure or Mach number from the solution to plot over specific surfaces, lines or points. e3post will write solution files to the VTK file format for viewing and any further processing in Paraview[5].

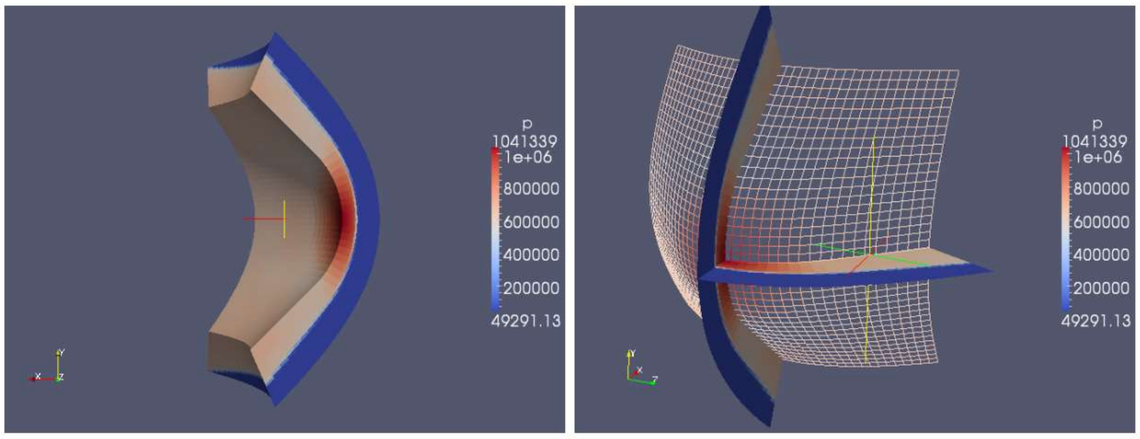


Figure 15: An example of a 3D e3prep generated grid - a spherically blunted cone. The left image shows contours of pressure on the surface of the domain. The right shows a surface grid and two slices through the domain. Post processing was done with e3post.py and visulised in Paraview. (Jacobs et al., 15).

[4] http://cfcfd.mechmining.uq.edu.au/eilmer3/e3prep.html
[5] http://www.paraview.org/

2.4 Aerodynamic Codes

A review of relevant aerodynamic CFD codes is given below. Given the short timeframe, the author's familiarity with a program was a critical factor in favour of the final program selected, Eilmer3.

2.4.1 HYPAERO

As introduced in section 2.3.1, HYPAERO is a surface panel code operating on the principles of strip theory. Although a comprehensive design tool, it has several shortcomings. These include an inability to model separation and thus high angles (8°) of attack, interaction effects between bodies or low speed flows (Jazra & Smart, 2009). This means a full aerodynamic database of the SPARTAN flyback (Figure 10) is not possible.

Furthermore, large errors are observed when compared with CFD methods. For example, in the case of an ogive-cylinder combination, errors of approximately 2% for horizontal pressure forces, 10 – 30% for vertical pressure forces and less than 10% for viscous forces are apparent (Jazra, 2010).

2.4.2 VULCAN

VULCAN[6] is a structured, full and parabolised Navier-Stokes code which will solve turbulent, non-equilibrium and chemically reacting flows. It has been used to compute accurate solutions for the full X-43 vehicle (Figure 16) as well as scramjet and other geometry shock tunnel tests (Robinson & Martin, 2008). There is some recent experience at The University of Queensland for the simulation of hypersonic flows (Jazra, 2010; Doherty, Smart & Mee, 2015). Although an attractive option which is appropriate for the task, the author's lack of familiarity and short thesis duration reduced appeal.

[6] https://vulcan-cfd.larc.nasa.gov/

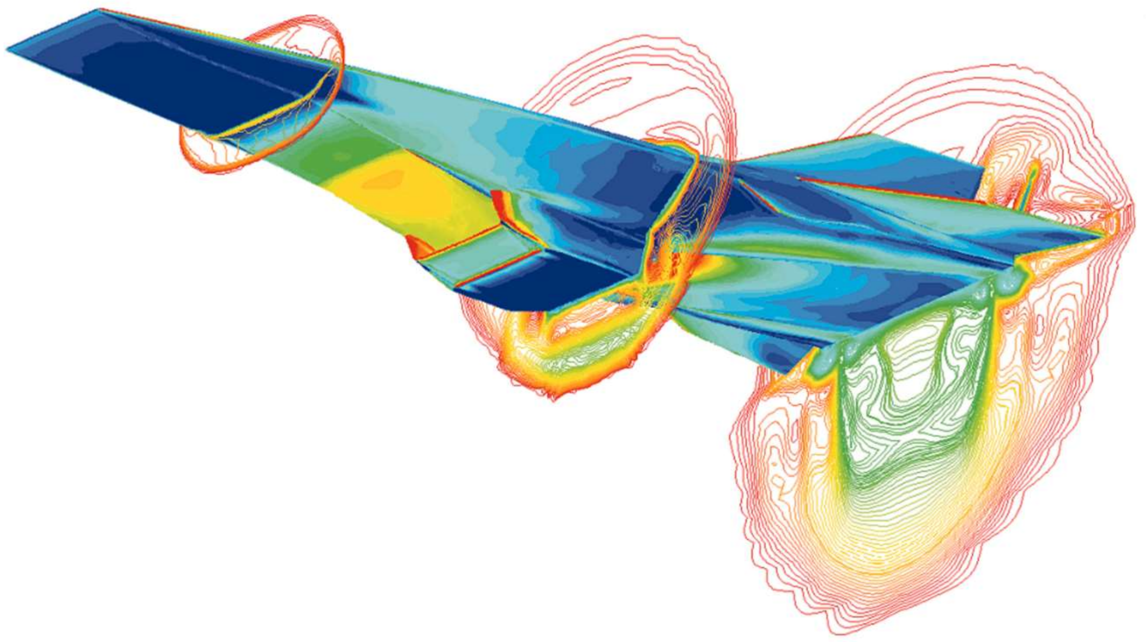


Figure 16. Pressure contours on the surface and cross sectional planes around NASA's X-43 scramjet vehicle calculated using VULCAN. This vehicle exhibits flat, 2D forebody and boat tail or aft body geometry (NASA).

2.4.3 Eilmer3

The Eilmer codes are a series of C++ and python programs developed by the Compressible Flow CFD Group[7] at The University of Queensland for over 25 years (Jacobs et al., 2012). The formulation and implementation is extremely complex, requiring a deep understanding of fluid mechanics and computational techniques. In brief, the code solves the integral form of the compressible Navier-Stokes equations, integrating through time to give a transient solution. It includes general thermochemistry and high temperature effects such as chemically reacting air (Jacobs et al., 2015). The general Navier-Stokes conservation equation is:

$$\frac{\partial}{\partial t}\int_V U dv = -\oint_S (\overline{F_i} - \overline{F_v}) \cdot \hat{n} dA + \int_V Q dV$$

[7] See http://cfcfd.mechmining.uq.edu.au/

Where the terms are the change in time, flux and source terms from left to right. The code is specifically designed with hypersonic flows in mind and has been thoroughly verified and validated against a multitude of cases (Gollan & Jacobs, 2013). As a time-marching, Navier-Stokes code, it is one of the most commonly used type in hypersonic vehicle simulation (Hirschel, 2005). Furthermore, with the author's existing experience and relatively short time period available, Eilmer is an excellent CFD code for the project. Comprehensive information on Eilmer may be found in the user guide by Jacobs et al., (2015) or the source code.

Courant-Friedrichs-Lewy criterion

The CFL criterion allows an appropriate time-step to be selected which limits propagation of information to a distance of one cell width or less. The stability limit of the time-step is a function of the flow field. In the case of poor grid quality, the CFL number should be reduced. Depending on the specific flow-field and location, this generally means skewness of 0.85 or more. It is difficult to comment generally, but an initial CFL value of 0.5 is a reasonable starting point to ensure time accuracy (Gollan & Jacobs, 2013). Eilmer can also be set to check the CFL condition more regularly however this check is expensive (Jacobs et al., 2015).

2.5 Further Reading

The reader is referred to the following; an explanation of hypersonic flow in Anderson (2006) and Anderson (2003); comprehensive details on SPARTAN are given in Thomas Jazra's (2010) and Dawid Preller's (2016) PhD theses respectively; information on parametric forebody design and optimisation may be found in Berens & Bissinger (1996) and Berens & Bissinger (1998) and finally Duveau et al. (1999).

For details on analyses of similar launch vehicles the reader is referred to the papers by Mehta et al. (2015); Eggers and Dittrich (2011) and Wuilbercq et al. (2014) for information on SKYLON. Additionally, Savino et al., present findings on the performance of a suborbital hypersonic vehicle, HyPlane (2015). Details on these studies are not discussed here owing to differences in methodology.

3 Approach

The big picture of the grid generation tool developed is now briefly discussed. The tool is coined AMARA, pArametric Meshing for AcceleratoR Analysis, (a catchy acronym although not the most accurate of adjectives).

Three files AMARAgeom.py, AMARAblock.py and AMARAlibrary.py have been developed (Figure 17). AMARAgeom.py is the master file. Here the geometry, initial and inflow conditions are defined and cell numbers and simulation parameters may be tuned. AMARAlibrary.py contains all the function definitions in a streamwise order. The functions define the blocks in one of three ways:

1. Construct a parametric volume using the parameters r, s, t to return Cartesian coordinates x, y, z.
2. Build the 12 paths which define a hexahedral volume using the e3prep function `WireFrameVolume()`.
3. Linearly interpolate between surfaces to create a volume.

These methods will be explained further as the blocks are introduced. With the exception of the forebody blocks, each group defined in Figure 17, are defined programmatically in a python class. The subsequent functions define the subtly different blocks above or below the wing as well as functions to rotate the blocks around the fuselage.

Finally AMARAblock.py contains all the block and boundary condition definitions. In both AMARAlibrary and AMARAblock, there are a small number of variables defined. In extreme cases it may be advantageous to change these otherwise all that need be changed lies in AMARAgeom.

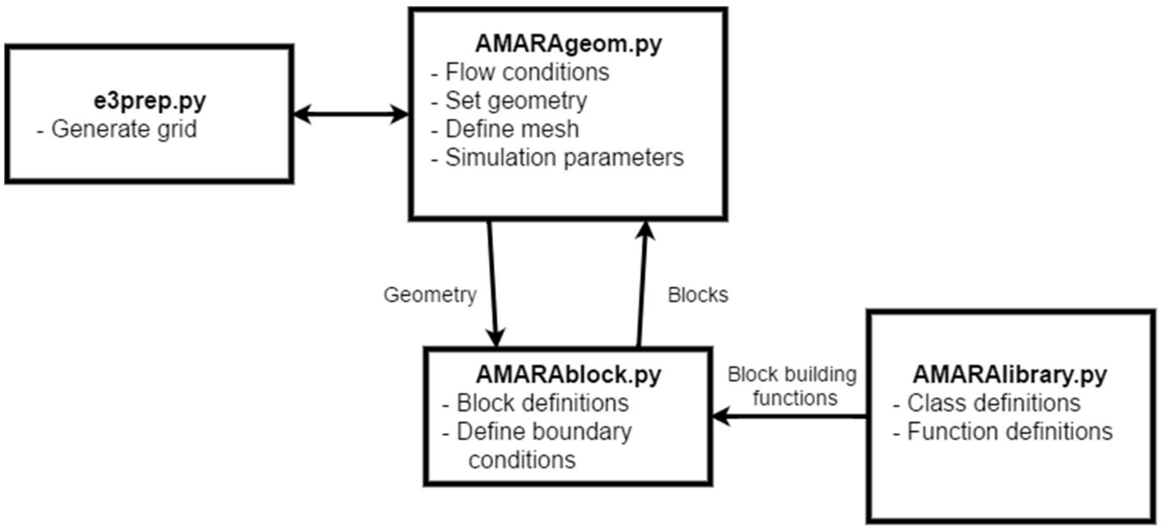


Figure 17: A big picture description of AMARA.

4 Geometry and Grid

This section will give a further in depth discussion of the AMARA geometry with reference to SPARTAN. Each section of the grid will be discussed with an explanation of the parametric variables of note and how the blocks are constructed. Where appropriate snippets of code will also be given. Anyone who has read lengthy portions of code will appreciate the difficulty in understanding other's work. This will be presented in a stream-wise order, from nose to boat tail. Finally, examples of the grids generated, grid quality, limitations and verification of geometry will be given.

4.1 Forebody

The forebody is constructed as a spherically blunted cone and is shown in Figure 18. The spherical nose is truncated such that it leaves a smooth transition into the ramp at the forebody half angle φ_{fb}. The length of the forebody is a dependent variable and given by

$$L_{fb} = \frac{(r_{fb} - r_j)}{\tan(\varphi_{fb})}$$

Where r_j refers to the radius at the junction between nose and ramp as shown in Figure 18.

$$r_j = r_{nose}\cos(\varphi_{fb})$$

Aspect ratio in the forebody is included as shown in Figure 19. This is simply a redefinition of the x coordinate, stretching the fuselage to an ellipse.

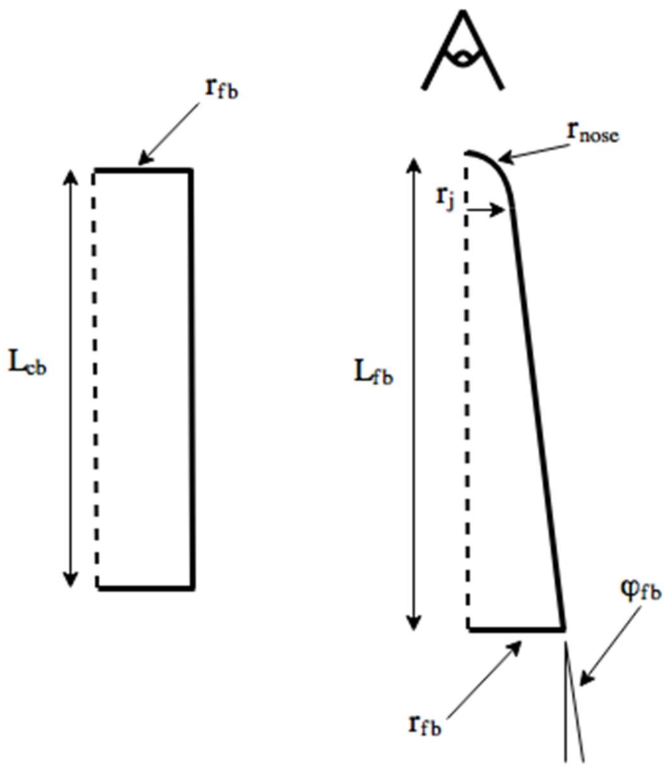


Figure 18. Forebody geometry. Note the nose radius shown here is greatly exaggerated.

Finally, camber in the forebody is also included in two directions. Longitudinal camber has been included to study the effect on the vehicles longitudinal trim. The blocks are first defined forming a straight cone and then manipulated using the NACA 4-digit cambered airfoil equation.

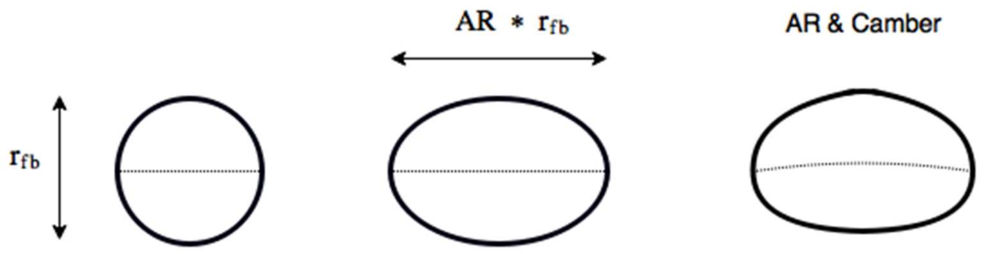


Figure 19. Definition of aspect ratio and transverse camber in the forebody.

$$y_c = \begin{cases} C_{a,l}\, {}^{x}\!/_{C_{p,l}{}^{2}} \left(2C_{p,l} - {}^{x}\!/_{L_{fb}}\right) & 0 \le x \le C_{p,l}L_{fb} \\ C_{a,l}\, {}^{(L_{fb} - x)}\!/_{(1 - C_{p,l}{}^{2})} \left(1 + {}^{x}\!/_{L_{fb}} - 2C_{p,l}\right) & C_{p,l}L_{fb} \le x \le L_{fb} \end{cases}$$

As an extension on previous work, transverse camber has been added to the forebody. This is implemented as a simple correction as a fraction of the forebody base radius.

$$y_c = C_a\, {}^{x}\!/_{r_{fb}} \qquad 0 \le x \le r_{fb}$$

The forebody blocks are formed out of three sets of blocks, nose tip, nose join and ramp. A two dimensional projection of the tip and join blocks is shown below in Figure 21. following the perspective shown in Figure 18. The first 12 blocks, on the nose tip, are necessary to avoid the grid singularity present at the tip of the conical forebody. These are numbered 7 - 12 in Figure 21. The remainder of the nose is formed by the ‘Join’ blocks, numbered 0 – 6.

These blocks are constructed in a different manner to the remainder of the grid. As the actual construction functions are quite long, comprehensive pseudo code for the nose tip is given below. The process is illustrated in Figure 20.

```
def nose(r, s, t, rInner, dr, phis, AR, thetas, face='TOP',
            block=''):
    “”” Construct a parametric volume starting at rInner of
        thickness dr. phis and thetas are lists of 4 angles
        defining the polar and azimuthal angles of the corners
        nodes respectively in clockwise order [NE, SE, SW, NW]. ”””

    # Vary radius at which mesh is generated using t.
    R = rInner + t*dr

    # Find 2D radii for the four corners
```

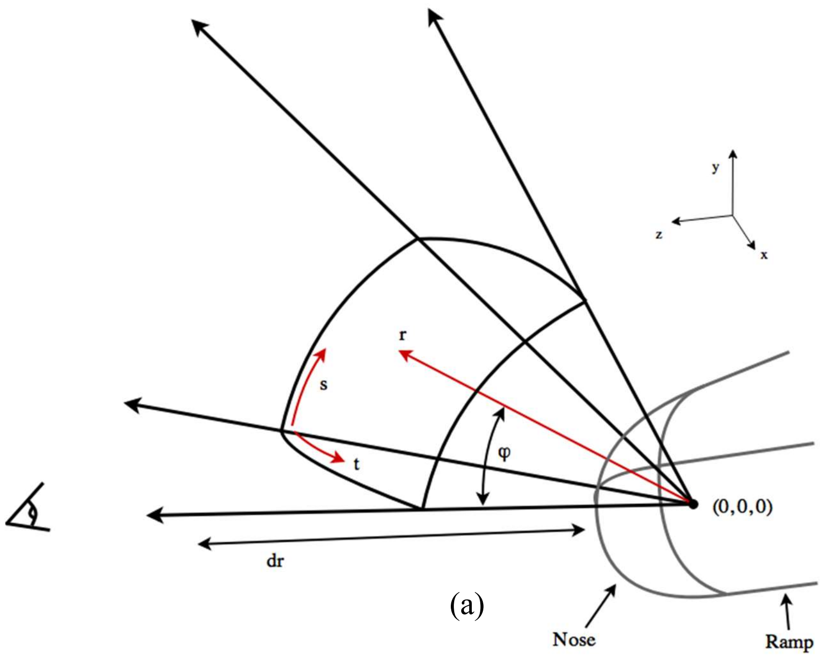
```
r_NE = R * np.sin(phis[0]); r_SE = R * np.sin(phis[1]);

# Define azimuthal angle of each node on 2D plane.
angle_NE = thetas[0]; angle_SE = thetas[1]

# Define nodes for each corner, face is a Boolean value, -1
      # for ‘BOTTOM’, 1 for ‘TOP’
NE = node(r_NE * np.cos(angle_NE),
                face*r_NE * np.sin(angle_NE)
                0.,
                label=’NE’)

# Create paths
SWSE = Line(NW, NE) # N; SENE = Line(NE, SE) # E

# Create 2D patch
Face = make_patch(NWNE, SENE, SWSE, SWNW)

# Extract polar coordinates defining 2D patch
r_2D, theta_2D = cart_to_2D_polar(Face.eval(r, s).x,
                                            Face.eval(r, s).y)
# Calculate polar angles
phi = np.arcsin(r_2D/R)

# Return Cartesian coordinates to be modified with AR and
camber
x,y,z = sphericalpolar2car(R, theta_2D, phi)
```

The ramp blocks form the remainder of the forebody. It is along this length that the boundaries are adjusted from the equiangular distribution of Figure 21, to that shown in Figure 22. Construction of the ramp blocks introduces the second of the three block construction methods; interpolation between surfaces.

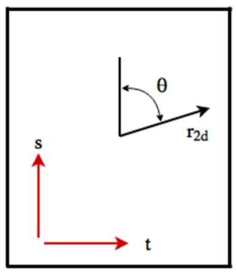
The ramp function takes the corresponding mating face from the join block (bottom face) and then creates the top face following the same method of defining corner nodes from their angle and radius as in the code above. The volume is created by interpolating between the two surfaces using the following function.

```
def surf2surf(r, s, t, surfTop, surfBottom):
    # Use r and s to evaluated the coordinates of the two
    surfaces.
    topCoords = surfTop.eval(r, s)
    bottomCoords = surfBottom.eval(r, s)

    # Interpolate between surfaces using t to get x, y, z
    x = (1 - t)*topCoords.x + t*bottomCoords.x
    y = (1 - t)*topCoords.y + t*bottomCoords.y
    z = (1 - t)*topCoords.z + t*bottomCoords.z

    return x, y, z
```

As these blocks grow along the forebody, smoothness issues present themselves when compared to the neighbouring narrow wing top and tail blocks. This is highlighted in Figure 24. The blocks leading into above the wing (wing top), the wing itself (wing) and beneath (wing bottom) also have their far edge defined as a straight path. The requirement for this is discussed in section 4.3.



(a)



(b)

Figure 20. The construction of the nose blocks. (a) shows a constant radius slice through the mesh at r. This 3D surface is then collapsed to a 2D plane in the x-y plane. (b) the grid is then generated and then transformed back to 3D. Note the perspective for (b) is that shown in (a).

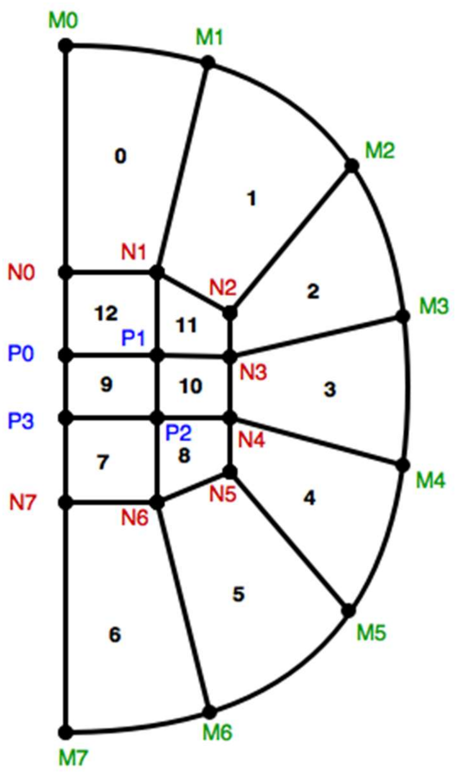


Figure 21. A 2D projection of the surface nodes defining the tip (7-12) and join (0-6) forebody blocks. The nodes are numbered and block numbers are central in each block in black.

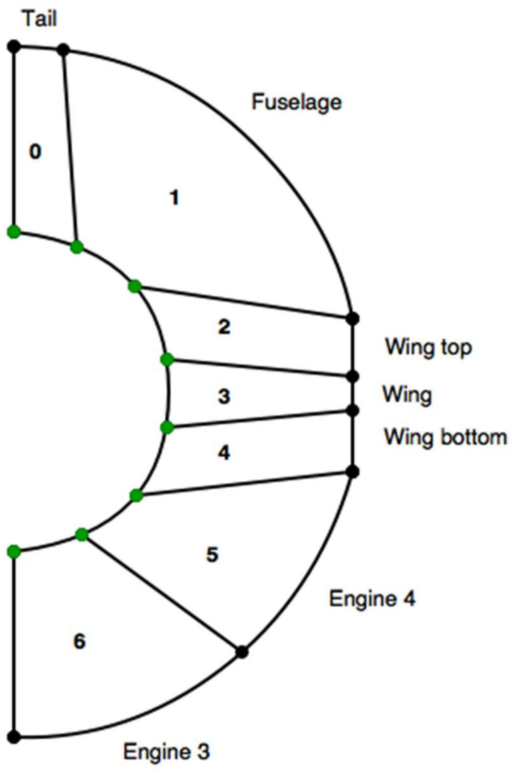


Figure 22. A 2D projection of the bottom face nodes defining the ramp blocks (0-6). The M nodes of the join blocks are shown here in green as in Figure 21. Note the straight paths defining the wing blocks.

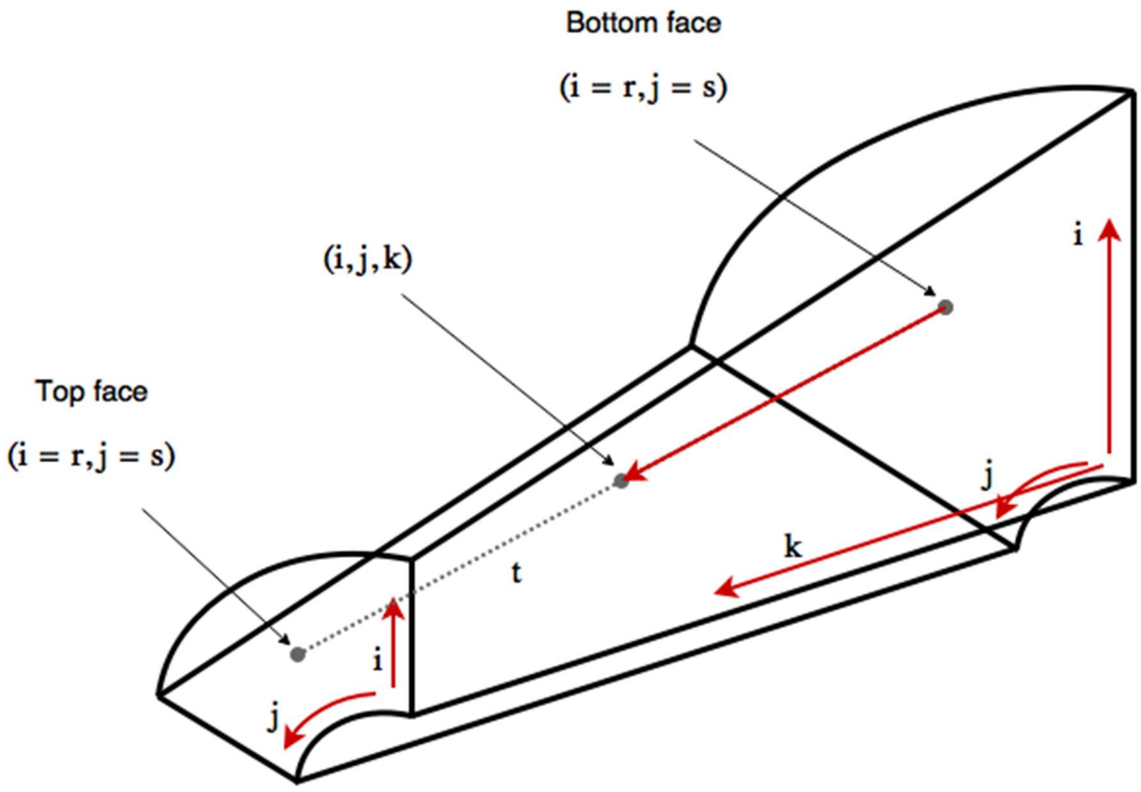


Figure 23. Ramp construction utilising the interpolation method. Here the top and bottom face coordinates are evaluated by r, s. The third dimension is realised by parametrising k with t..

Summary of Parameters

- The geometry is controlled through the nose radius r_{nose}, the forebody halfangle φ_{fb} and the aspect ratio AR. The fuselage diameter r_{fb} may be changed but is left constant.
- Forebody camber may be controlled through the longitudinal and transverse functions. The NACA equation is implemented for each taking the maximum camber amount C_a and location of maximum camber C_p specified as fraction of the chord length (forebody length and local z-x plane radius respectively).
- The mesh thickness is controlled through the radial thickness at the nose t_{nose} and the mesh half angle φ_{mesh} (measured from the x-z plane). For fine tuning at different nose radii, the azimuthal angle of the M nodes defining the tip blocks is φ_m may also be useful.
- For vastly different geometries, the polar angles of the M and/or N nodes may also be changed to encourage grid quality.

Table 1. Summary of parameters describing the forebody and their respective range if applicable.

Geometry		Mesh	
Parameter	Range	Parameter	Range
r_{nose}	~	t_{nose}	~
φ_{fb}	$(1°,\ 20°)$	φ_{mesh}	$\sim 7 \times \varphi_{fb}$
AR	$(1.0,\ 2.3)$	$\varphi_{tip}{}^{*}$	$\sim 0.5 \times \varphi_{join}$
r_{fb}	~	$C_{p,l}$	$(0,\ 0.5)$
$C_{a,l}$	$(0,\ 0.3)$		

* Setting this as a function of φ_{join} gives reasonable results. This should be a target parameter should mesh quality in the nose blocks become an issue at a given geometry.

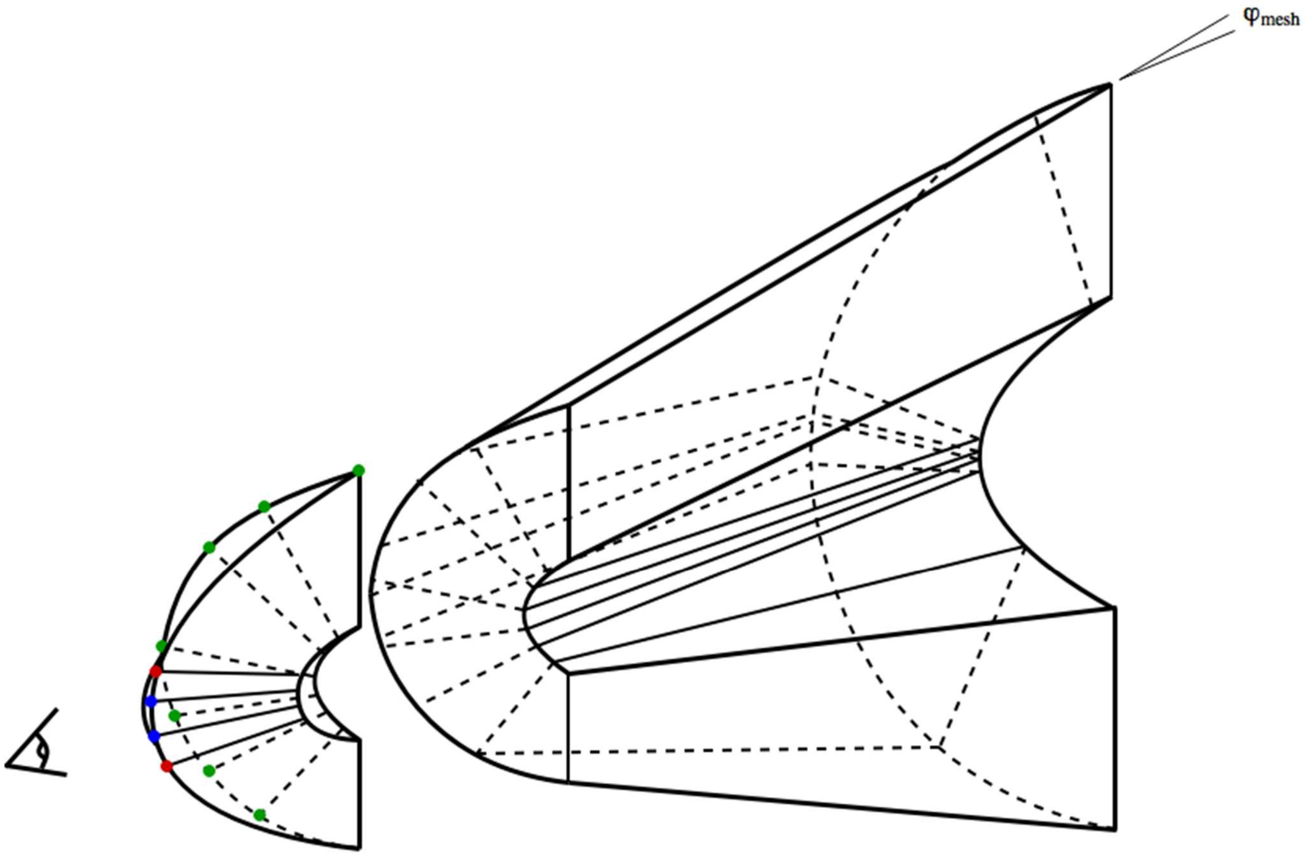


Figure 24. A 3D representation of the forebody blocks. This figure shows the change in size of the ramp blocks as they progress along the forebody ramp to their final angles. Note that only the surface boundaries are shown for clarity. The M, N and P nodes are shown in green, red and blue respectively as in Figure 21.

4.2 Engine

The original engines employ a shape-transitioning inlet configuration, called C-REST, to allow the smooth capture of air without spillage between engines (Preller, Smart and Schutte, 2016). For the purposes of the current work however, the geometry was simplified to a series of axisymmetric sections defined by the length fractions (of the entire engine) and wall angles with respect to horizontal. These approximated the engine modelled the inlet, inlet ramp, combustor and outlet and are illustrated in Figure 25. The engines are installed on the vehicle such that the inherent kink in the flowpath coincided with the forebody-centrebody junction. The length of the centre body L_{cb}, is therefore defined by the length of the combustor $L_{combustor}$ and outlet L_{outlet}, as the outlet is constrained to be coincident with the boat tail. The wall length of the outlet was a dependent parameter:

$$\varphi_{outlet} = \tan^{-1} {r_{outlet} - r_{fb}}/{L_{outlet}}$$

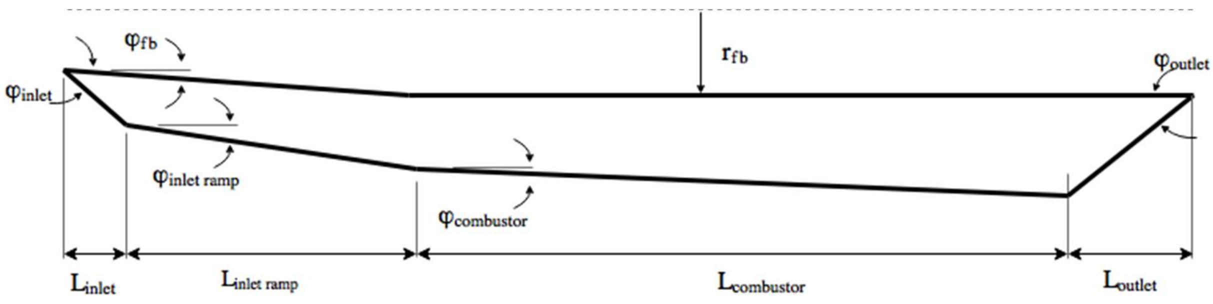


Figure 25. A 2D cross-section through the centre symmetry plane of the simplified engine geometry

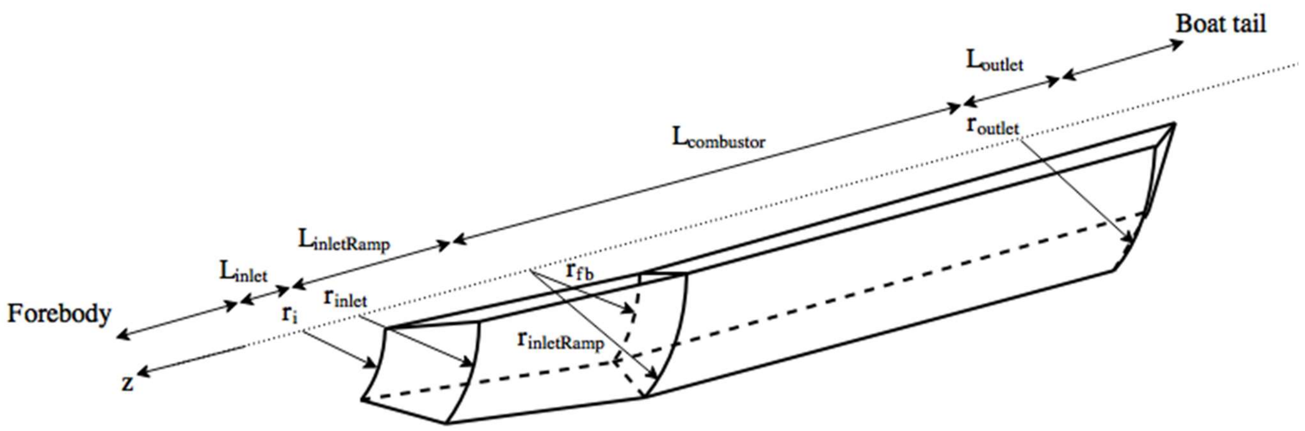


Figure 26. A 3D representation of the simplified engine module showing the radii and lengths for each section.

Summary of Parameters

- The engine was split into four sections, the lengths of which are specified as length fraction of the total engine length, L_{engine}.
- These parameters define the inlet, inlet ramp, combustor and outlet.
- The wall angles for each from horizontal are also specified. The outlet wall angle is dependent on the length specified such that it meets the boat tail.

Table 2. Summary of parameters describing the engine and their respective range if applicable.

Parameter	Range	Parameter	Range
L_{inlet}	~	L_{engine}	> 0 s.t. $\varphi_{sweep} < 85$
$L_{inletRamp}$	$! > \pm\sim 0.5L_{inlet}$	φ_{inlet}	$(0°,\ 50°)$
$L_{combustor}$	~	$\varphi_{inletRamp}$	$(0°,\ 5°)$
L_{outlet}	~	$\varphi_{combustor}$	$(0°,\ 5°)$
$L_{engine} = \sum L_{inlet} + L_{inletRamp} + L_{combustor} + L_{outlet} = 1^{*}$			

* Note this refers to the length fractions.

4.3 Wing

The wings are described by two independent parameters, the halfspan s, and thickness, t_w. The leading edge is constrained to the inlet of the engine and the trailing edge is coincident with the boat tail. Thus, like the centrebody, the chord length is defined by the engine length. Owing to time constraints, the wings are currently modelled as blunt, constant thickness plates rather than the desired diamond profile. However, this dimension is defined by a separate function facilitating the future inclusion of a wing profile.

The halfspan, dictating wing sweep φ_w, is an important parameter with strong favourable effects on the vehicles lift-to-drag ratio L/D as well as trim upon the addition of control surfaces. The mesh may be controlled over the wings farfield halfangle to capture behaviour caused by the wings.

Figure 28 shows the blocking structure over the wings. This is the primary structure for the entire grid has it is required to revolve around the entirety of the vehicle.

The block numbers and naming here are also referred to throughout the thesis. These blocks also introduce the final block construction method – WireFrameVolume. In this method, the 12 paths defining a hexahedral volume are defined. These paths are then passed to the e3prep function WireFrameVolume. This results in a highly efficient method of generating a block for a number of reasons. Firstly, the generation itself is computationally efficient when compared to the interpolation method. This is because it avoids having to evaluate cells on the two surfaces. Secondly, it lends itself well to setting up the blocks as Python classes. Each block class may have path attributes allowing one to extract paths or groups of paths to construct the neighbouring block. This is the primary method used throughout.

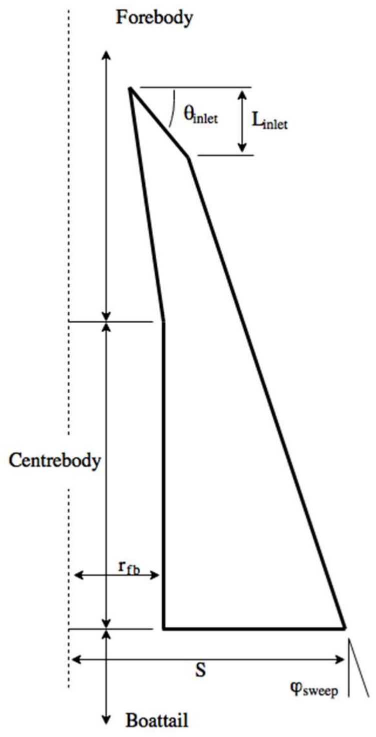


Figure 27. Planview of the wing geometry. Note that all the angles are exagerated.

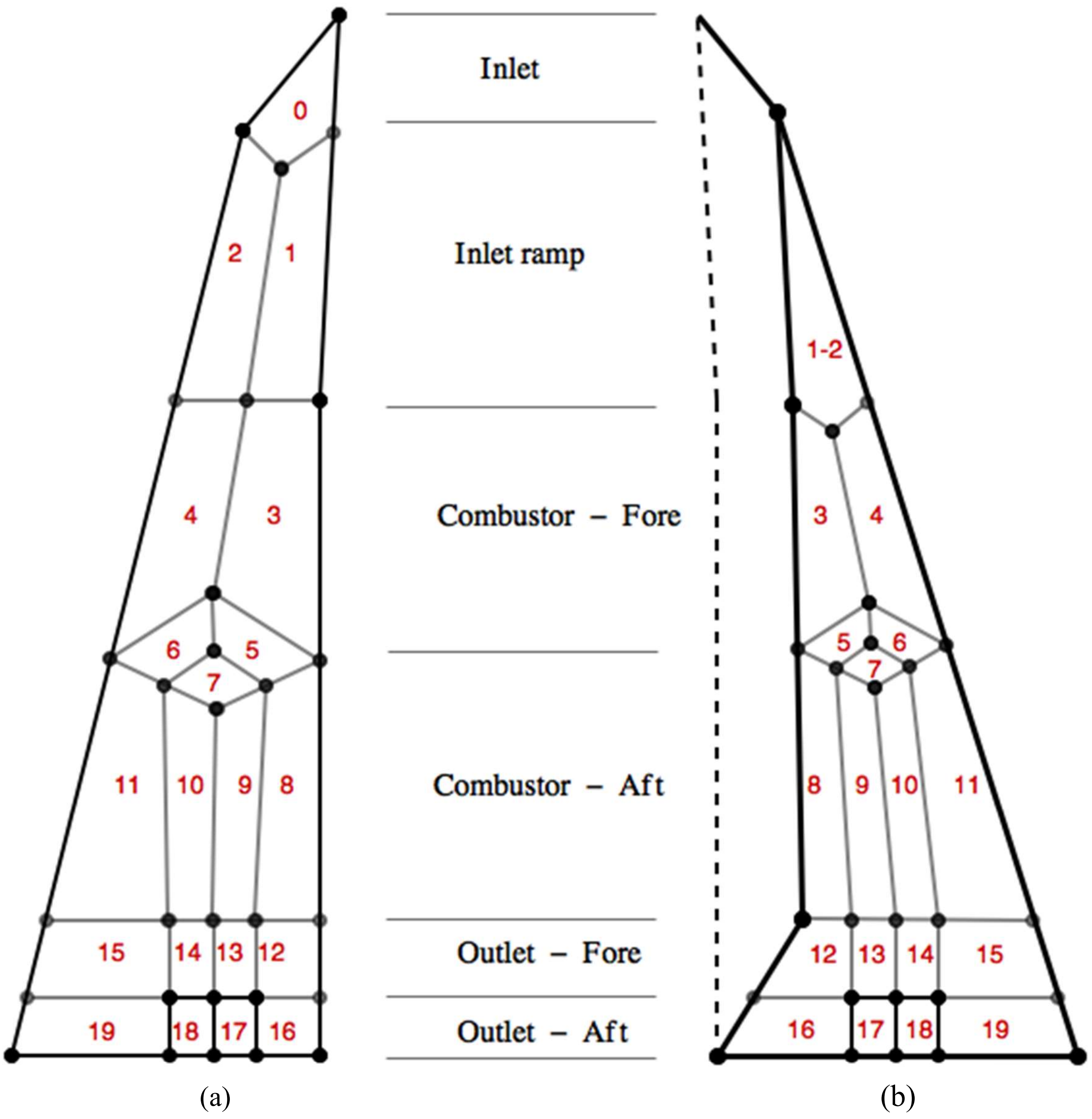


Figure 28. Plan view of the blocking structure for the top (a) and bottom surface (b) of the wing. Note block numbers and the names of the block groups.

A brief discussion of each block will now be given in a streamwise fashion. Firstly, the inlet or root fairing block. This is constrained between the inlet and fuselage and hence has the potential for highly skewed cells. It is also the first point where two downstream blocks collapse. In order to preserve quality at this function of three blocks, the interior angle was forced to 120°.

On the bottom surface, the inlet ramp block (1-2) is constructed in the same way – the aft interior angle is forced to 120°. This block runs for the length of the inlet ramp and allows its variation in the wall angle.

The fore combustor blocks (3,4) are simply constructed by interpolating between the diamonds and inlet ramp. The diamonds (5, 6, 7) were devised to collapse the number of blocks. A detailed image of them showing construction is given in Figure 29. This shows the rear boundaries at split at the midpoint and the central vertex is central to the whole diamond. Again, at the junctions of 7-9-10, the interior angles were forced to 120°. In favour of symmetry, this was continued to the junction of 3-4-5-6, accepting the reduction in quality. The diamond blocks are arbitrarily constrained to be halfway along the combustor length. This was selected without investigation to allow the fore and aft blocks (3,4 and 8-10) as much length as possible to transition.

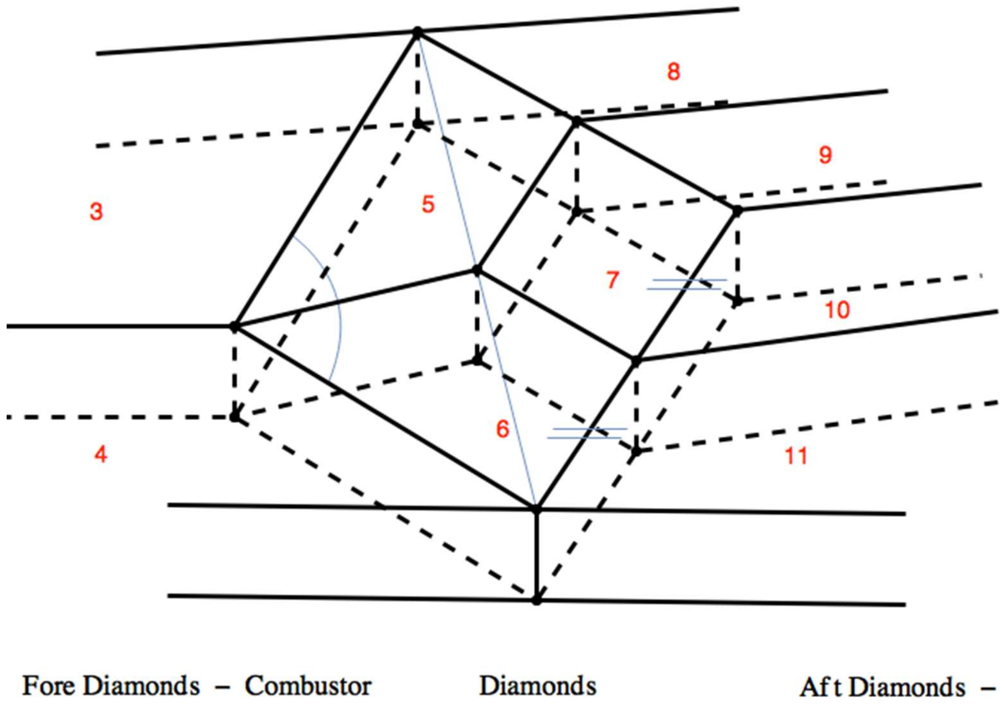


Figure 29. Detail of the diamonds blocks. Note the fine blue lines refer to the 120 angle at vertices of three blocks, the construction line defining the centre node (5-6-7 vertex) and equal subdivision of the aft boundaries.

The aft diamonds blocks are interpolated in the same manner as the fore diamonds. This was down because of the simplicity. It had the added effect of allowing the diamonds to be rotated horizontally. This is discussed in section 6 as a method used to improve the grid quality.

Finally, there are two rows of blocks inline with the outlet. This was split to allow block boundaries at the elevens. Control surface deflection is required as it is a critical parameter allowing the trajectory analysis to include longitudinal trim. However, this was not investigated until the rest of the grid was constructed. Unfortunately, owing to quality issues, modelling the deflection of the elevon surfaces was found to be unrealistic with a structured grid. Nonetheless. A recommended blocking structure is given here. A method to grid this is discussed later.

The strategy suggested is shown in Figure 30. In order for this to operate, the grid must have effectively two modes. One where the deflection blocks exist and another where it reverts to the base strategy. This strategy will result in some fairly poor cells because a) the very sharp angle at the root of the elevon and b) the collinear block boundaries. However, it is difficult geometry and this strategy requires only one change to the remainder of the strategy. The wake blocks must be split behind the wing trailing edge as shown.

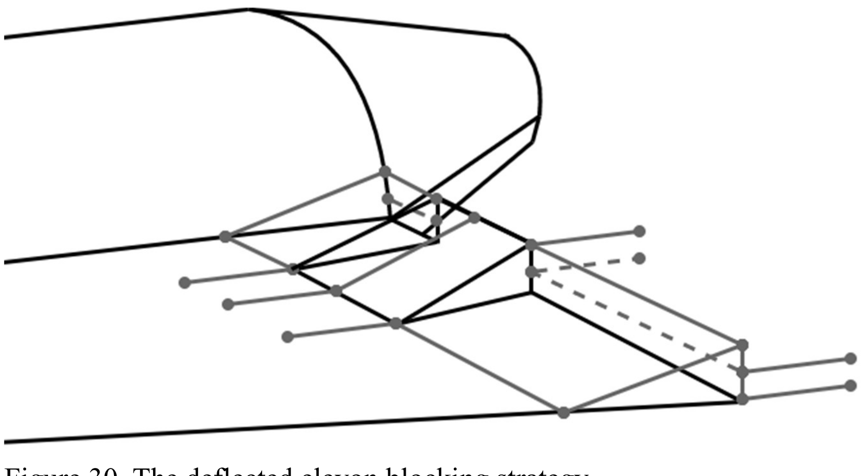


Figure 30. The deflected elevon blocking strategy..

Summary of Parameters

- The wing geometry is controlled through two parameters; the halfspan s and thickness, t_w. The wing length is defined by the engine length.
- The wing sweep φ_{sweep} is thus dependent on the halfspan and length.
- The mesh is primarily controlled through the far field mesh a number of parameters.
- The diamond blocks may be rotated. This is introduced in section 5.
- The angle to which the blocks immediately above and below the wing are rotated may also be changed.

Table 3. Summary of parameters describing the wings and their respective range if applicable.

Geometry		Mesh	
Parameter	Range	Parameter	Range
s	◊	φ_{ff}	$\sim 1.05 * \varphi_{fb}$
t_w	$0.04 * L_{engine}$	$\theta_{diamond}$	(0°, ~10°)
		$\theta_{rotation}$	10°

◊ Note this range is governed by the engine length and vice versa. The critical parameter is wing sweep and discussed in section 5.

4.4 Boat Tail

The boat tail (Figure 31) is modelled as a semi cylindrical top half and flat panels behind the four engines on the underside. It is then truncated at length L_{bt}. The boat tail half angle φ_{bt} dictates the boat tail end radius r_{bt}.

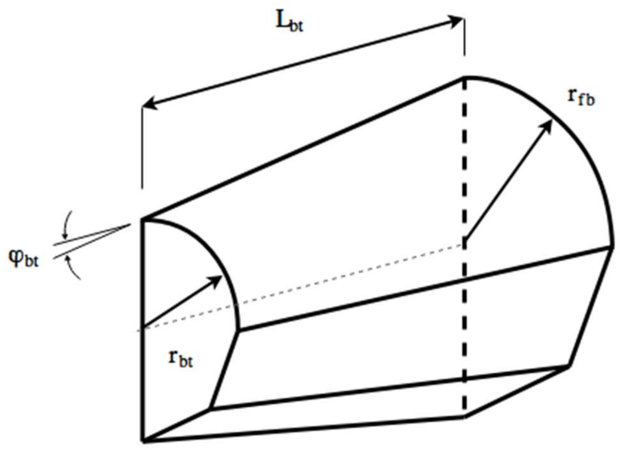


Figure 31. Schematic showing the boat tail construction and three controlling parameters.

The boat tail end radius is given by:

$$r_{bt} = r_{fb} - L_{bt}\tan(\varphi_{bt})$$

The boat tail blocks are simply a continuation of the wing blocks. That is four from the wings and one outer block defining the far field.

Summary of Parameters

- The boat tail half angle φ_{bt} and length L_{bt} control the geometry.
- The aft radius of the boat tail is dependent on the length and half angle. This is used in the construction of the central wake blocks.

Table 4. Summary of parameters describing the boat tail and their respective range if applicable.

Parameter	Range	Parameter	Range
φ_{bt}	(10°, 45°)	$L_{boatTail}$	$(r_{fb}\ ,\ 3r_{fb})$

4.5 Wake and Farfield

The central blocks forming the wake immediately behind the boat tail are formed following a similar structure to that found on the nose the forebody. These are formed by defining the nodes and then extruding them the distance to the outlet boundary L_{ff}. This structure is illustrated in Figure 32. The remainder of the wake blocks are formed simply as a continuation of the boat tail blocks and thus not discussed.

The grid is controlled through two parameters. One, the half angle of the farfield boundary φ_{ff} downstream of the forebody as introduced previously and two, the distance to the outlet boundary.

A schematic of the complete blocking structure of the vehicle is given in Figure 33. This shows the block boundaries on two planes. Firstly the wing plane and secondly the vertical symmetry plane. The boundaries showing the rotation of the blocks around the fuselage of the vehicle are also shown.

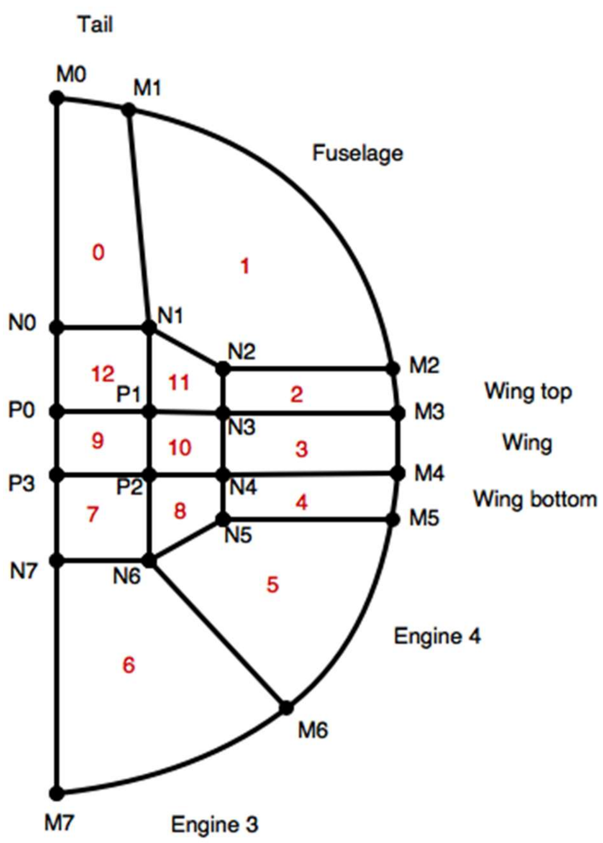


Figure 32. The wake blocks behind the boat tail.

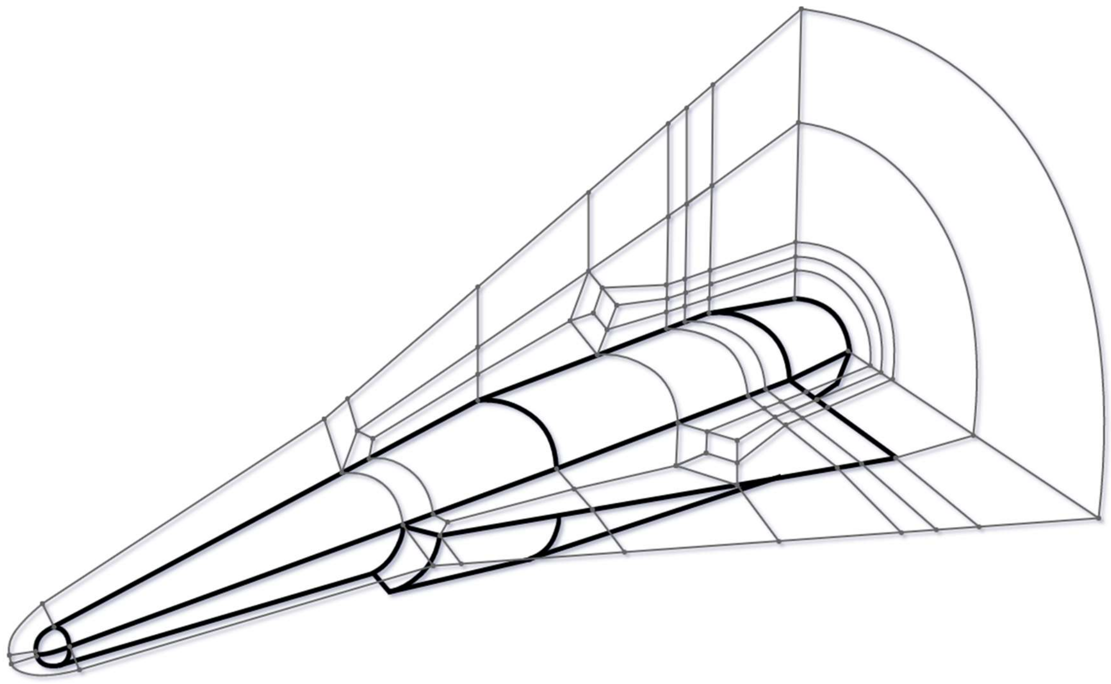


Figure 33. A representation of the three dimensional blocking strategy. Block boundaries (grey) are shown in three planes - top, coincident with the wings, the symmetry plane and front coincident with the rear of the vehicle. Note that block boundaries which are defined by the surface of the vehicle (black) are not shown.

5 Results and Discussion

In this section, the complete grid and some examples of its parametric ability will be given. The baseline SPARTAN geometry will be demonstrated before quality issues and geometric limitations are discussed. A simulation of the full vehicle as a proof of concept will then be presented before the results of a short study on the forebody.

The AMARA collection of codes and scripts may be cloned from the GitHub repository:

```
$ git clone https://github.com/AlexWard6/AMARA
```

5.1 The Grid

The final developed grid is shown below in Figure 34. This shows the surface grid of the SPARTAN geometry as defined by Preller, Smart and Schutte (2016). This grid is 383,300 cells in total and takes approximately 4 minutes and 30 seconds to generate on the Goliath computer cluster (Appendix A) or slightly longer on an average home machine (4 cores, 8 GB RAM).

Four boundary conditions were utilised in the grid for the outflow, inflow, symmetry and vehicle surfaces. The inflow condition, SupInBC, simply copies the supplied conditions into the ghost cells on the outside of the blocks at each time-step. ExtrapolateOutBC linearly extrapolates the flow data to the ghost cells each time step giving a general supersonic outflow condition. The vehicle surface is modelled as a fixed temperature wall at 300K with the temperature arbitrarily chosen. This is without consequence as viscous effects were not included in the simulations. Finally, the symmetry plane was modelled as an inviscid wall. Between blocks, Eilmer's `identify_block_connections()` function automatically pairs neighbouring blocks with the AdjacentBC boundary condition.

The task of setting the correct location of the boundary conditions presented one of the greatest challenges in finalising the grid. To help with this, a rough script, AMARAsurf.py, (see Appendix B) was written. This script extracts the block number, boundary condition and face with which the condition is associated. This output was then used with e3post to extract and display the surfaces of the specified boundary condition. The results of this are shown in Figure 35.

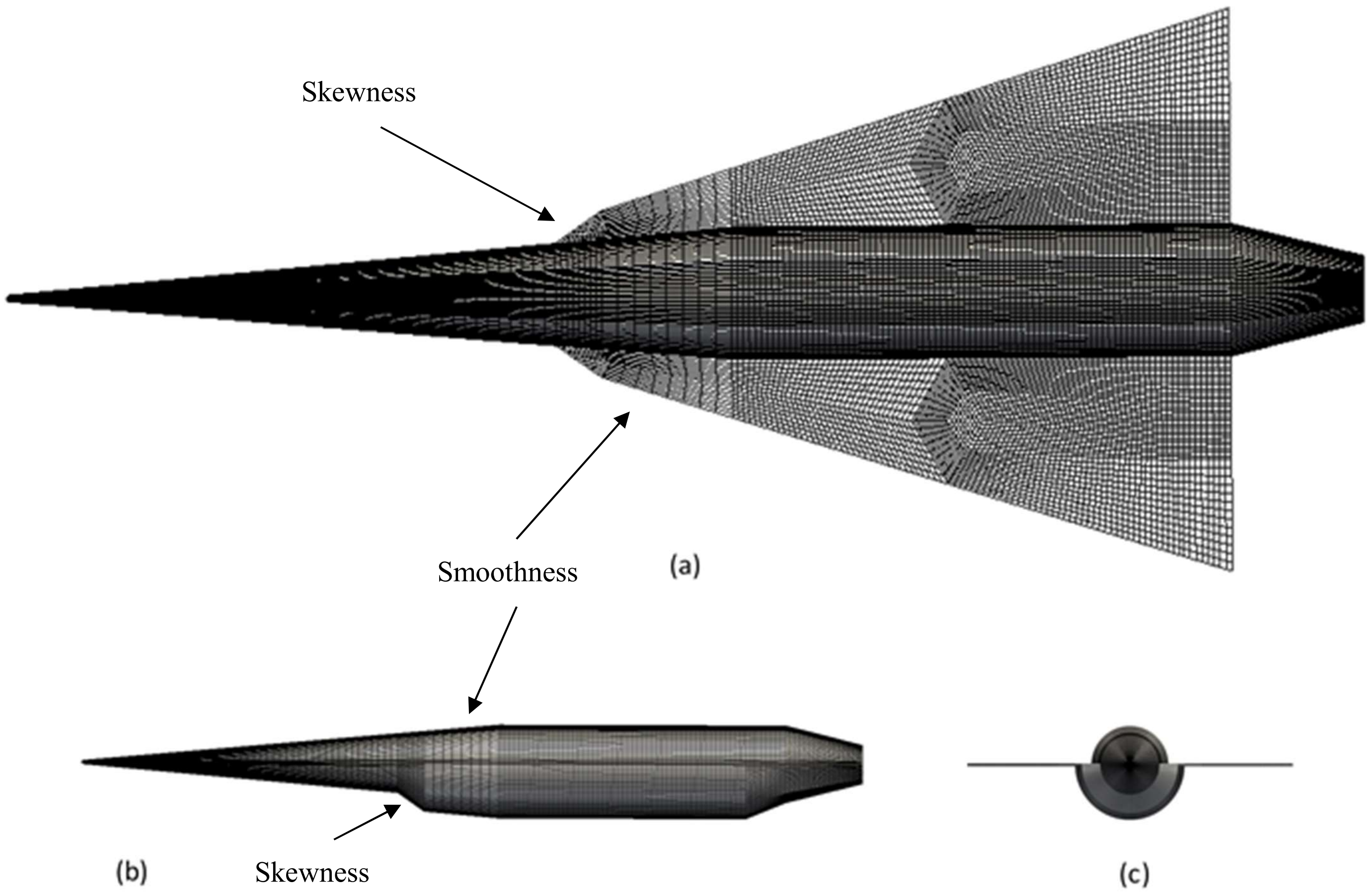


Figure 34. An example of an AMARA grid modelling the baseline SPARTAN of Preller, Smart and Schutte, 2016. This shows top (a), side (b) and front (c) views. The areas of worst skewness and smoothness are also identified.

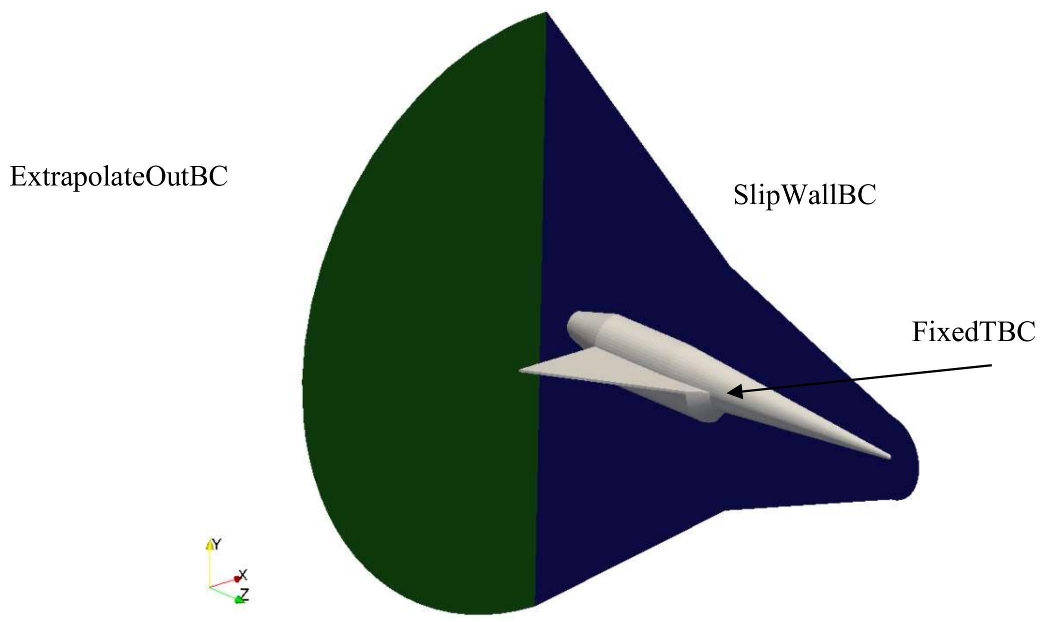


Figure 35. The boundary conditions of the grid – ExtrapolateOutBC, FixedTBC and SlipWall. For visualisations sake, the inflow condition, SupInBC is the last exterior surface and not shown. The global coordinate system is also shown in the bottom left corner with the origin at the centre of the blunted nose.

5.1.1 Geometric Verification

As a rudimentary check of the grid's geometrical accuracy, all parameters were set to model the baseline SPARTAN as defined by Preller, Smart and Schutte (2016). Using Paraview's probe location tool, the key dimensions for each component were checked. Good agreement was found for the majority of dimensions with the known simplifications and changes discussed in section 4. The results of this are shown below in Figure 36. Here a transparent image of the grid is overlaid on the SPARTAN geometry defined by HYPAERO.

One of the images is rotated slightly hence the slight discrepancy noticeable in the forebody and right-hand wing (lower wing in Figure 36). The largest difference is found in the engine geometry. However, this is to be expected. Firstly the engines are simplified substantially and an outlet section added similar to the shape presented by Doherty, Smart and Mee (2014) as discussed in section 5. Secondly, the dimensions of the REST and C-REST engines were unable to be found in the literature. Consequently, these were estimated graphically from the work completed by Doherty (2014) and Jazra, Preller and Smart (2013).

It is for these reasons that a discrepancy in the boat tail is seen as well. Nonetheless, this result shows the successful completion of the second major objective – exercise the tool on the baseline geometry.

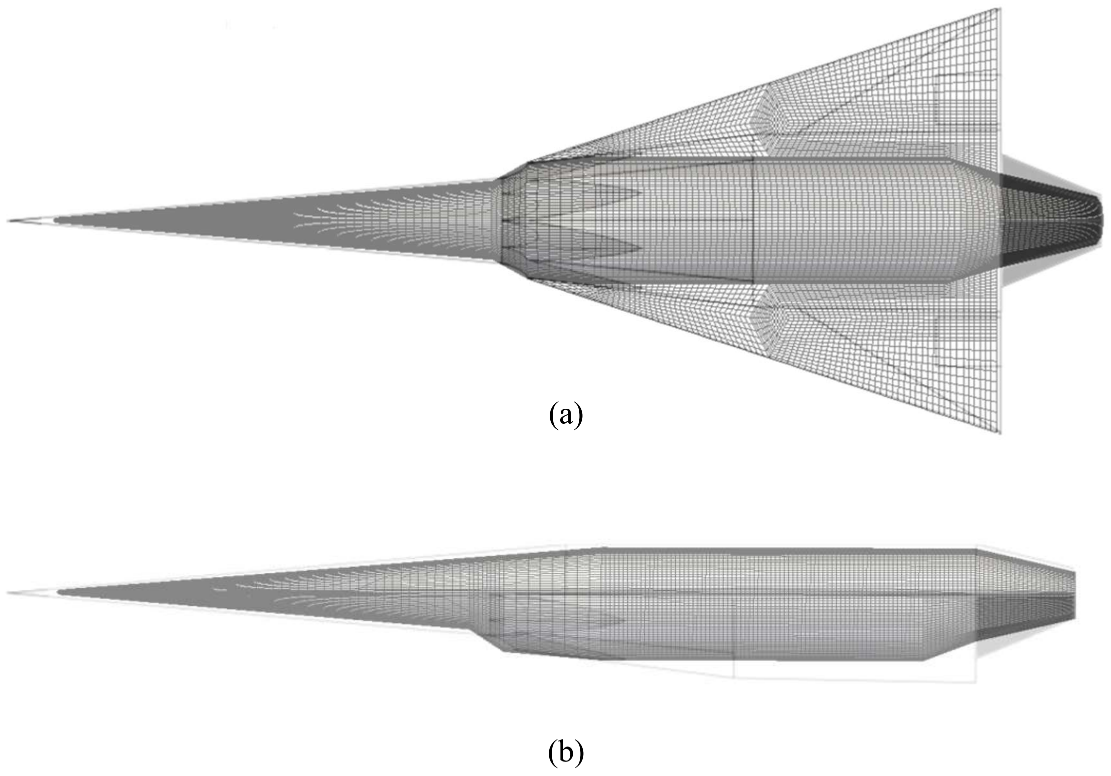


Figure 36. Top (a) and side (b) views of the baseline AMARA grid overlaid on a HYPAERO defined geometry.

5.1.2 Grid Quality

In general, the skewness of the grid was deemed acceptable, following a criterion of 0.85. This criterion was used as smoothness was generally quite good across the grid and may be somewhat adjusted by varying cell counts. Orthogonality was also quite good with the exception of the inlet ramp blocks – limitations caused by this area are discussed. The aspect ratio was generally good (< 4) but a small number of cells near the nose and wing far field are larger (< 9). There are several areas of concern which should be targeted in future work. These areas and the efforts made to improve quality will be addressed later.

Smoothness in the grid was controlled fairly well (80% > minimum of 0.25) by keeping the number of independent parameters controlling cell numbers to a minimum. Wherever possible the cell numbers were defined from neighbouring blocks such that the *cells/m* along the block boundary was constant. As labelled in Figure 34 however, the inlet ramp block exhibits poor smoothness. This is due to the downstream requirement of its longitudinal cell count matching that of the inlet blocks. The effect of this was minimised by averaging what the inlet and inlet ramp blocks should be and using then this for both.

Skewness is shown graphically in Figure 38 and 39. Figure 37 shows a histogram of cell skewness and shows a number of things. Firstly, the majority of the grid is of good quality (81.2% < 0.5) with 39% of cells at a skewness of 0.1 or less. A small percentage, 0.4%, of cells are highly skewed (> 0.75). The two peaks at 0.54 and 0.57 correspond predominantly to the blocks on the underside of the wing leading edge and fore of the wing diamonds as shown in the cloud of red in Figure 39. This identifies the first area of concern, the underside root fairing blocks (1-2).

Figure shows the locations of cells with a quality of 0.5 or greater. This corresponds to the worst 18.8% of cells. The poor quality areas are clearly shown in solid red. With reference to the block naming and numbering of Figure these are block 1-2 on the underside and block 4 on both sides. This identifies the second area of concern, the fore diamonds blocks 3 and 4.

Root Fairing (1-2)

The root fairing block on the underside is the worst block for the majority of geometric cases, consistently returning the worst cells. This is because of the highly skewed geometry caused by the combination of the high wing sweep and inlet ramp. In an attempt to improve this, the block construction method was changed to that of `AOPatch`. This used the twelve paths to define the six faces which in turn builds the volume. The results of an extreme case are shown in Figure 40. Although an improvement was noticed, this was only very slight at about 3% depending on geometry. This did however allow an increase in wing sweep at acceptable skew of around 2°.

Fore diamonds (3, 4)

The fore diamonds return very poor cells, only slightly better than the root fairing. This was because of the original blocking strategy and again the swept wings. An adjustment to the strategy was proposed. Add the ability to 'rotate' the diamond blocks in plane. The effects of this are shown in Figure 41. It is clear this relatively simple change greatly improves the quality of the grid.

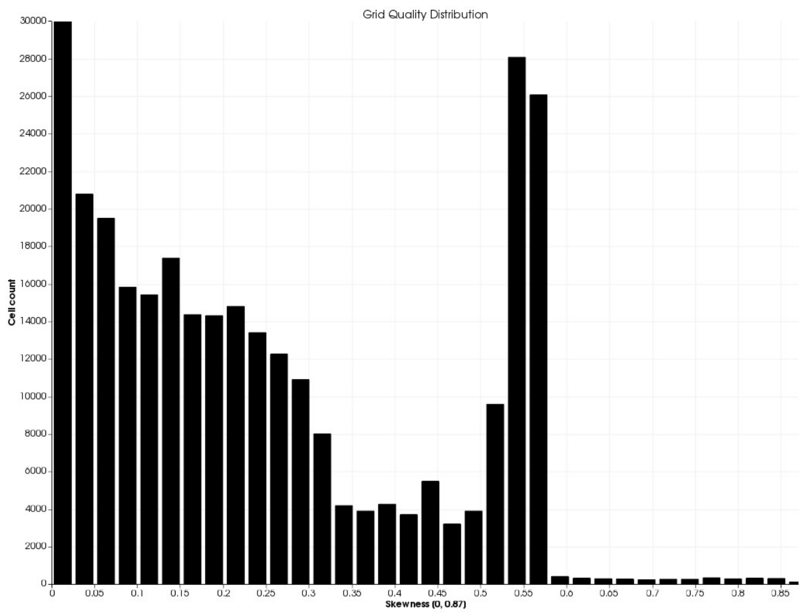


Figure 37. A histogram of the cell skewness distribution. There are 85257 cells (33%) in the first peak at a quality of 0.01.

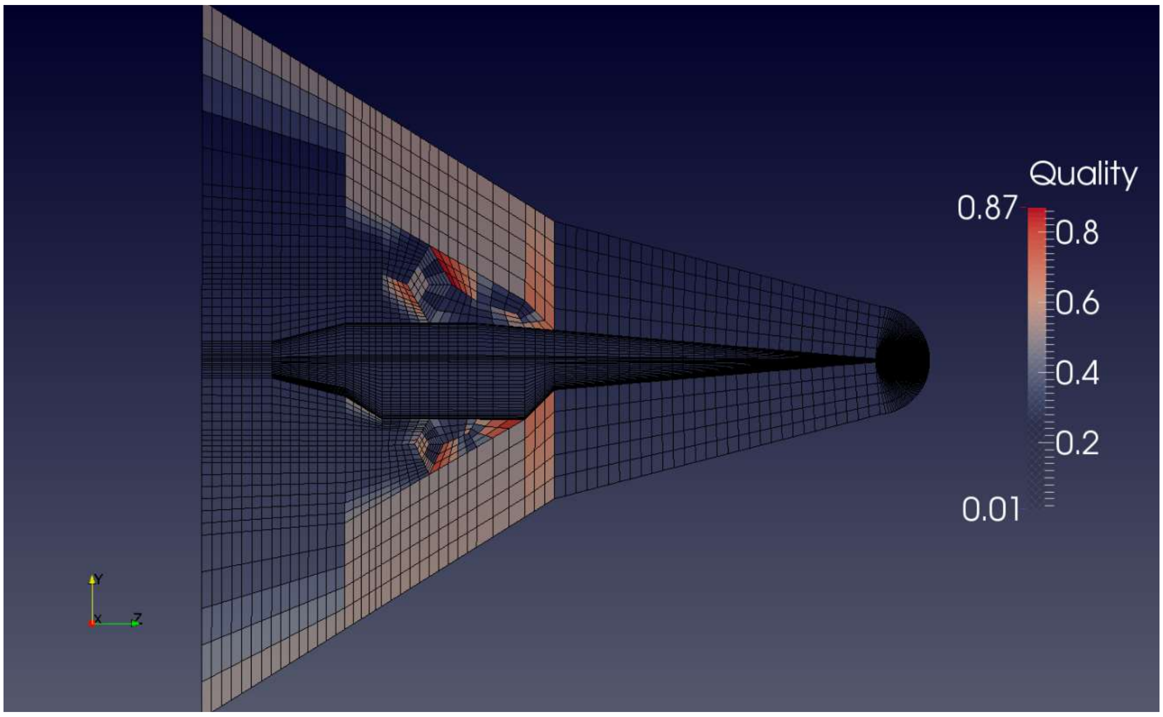


Figure 38. Skewness in the grid. Note that an opacity function has been applied - only cells with skewness greater than 0.5 are shown with increasing opacity.

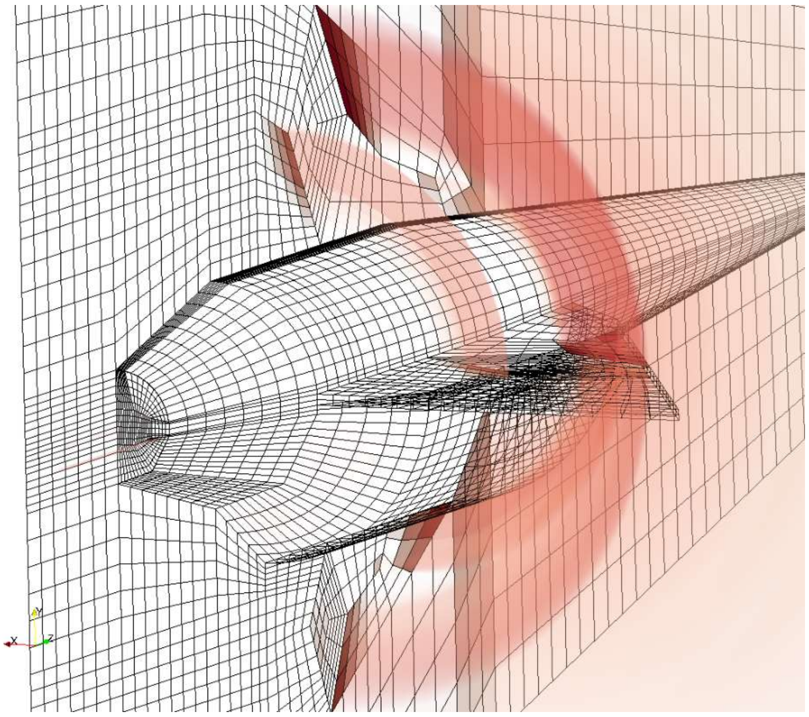
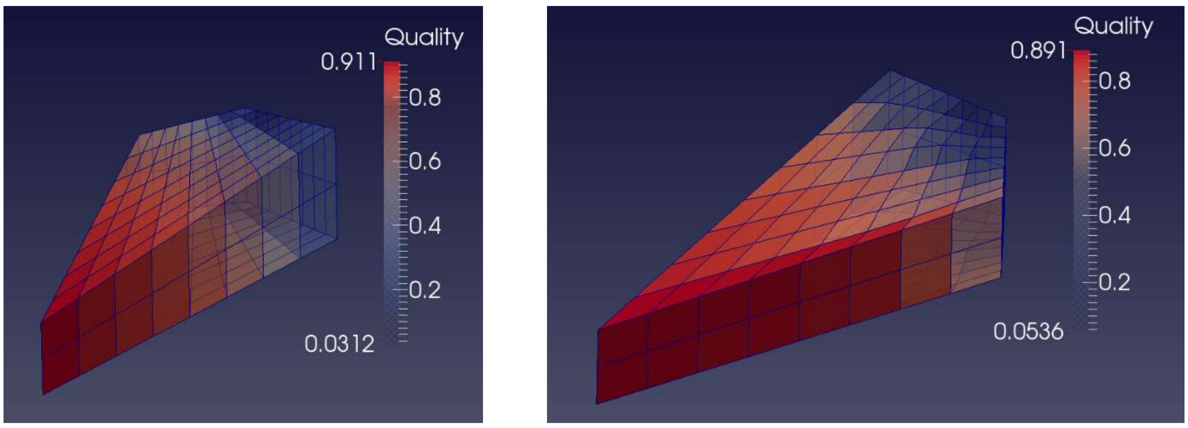
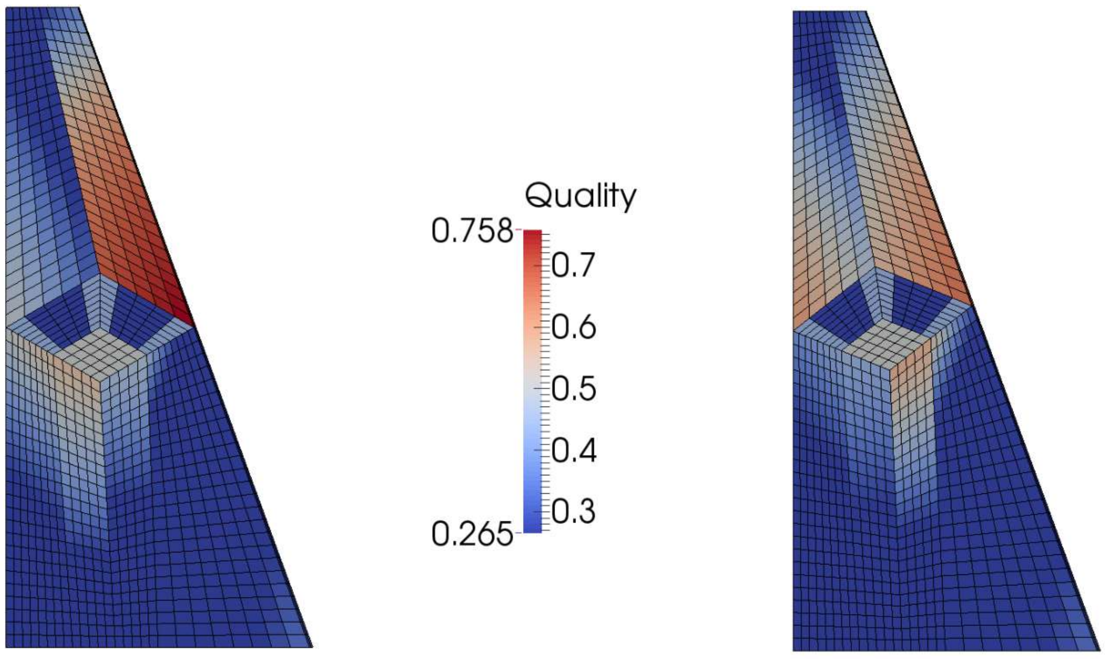


Figure 39. A volume rendering of the grid's skewness. An opacity function hiding skewness < 0.5) was applied to emphasise the location of the worst cells and how this propogates through the domain.



(a) (b)

Figure 40. The original (a) and AOPatch (b) block construction. The geometry was set to an extreme case where the wing sweep was 80° and the inlet angle was 40°. Note the views are from the opposite side and an opacity function as been applied to highlight the individual poor quality cells. Changes to the background mesh were investigated by varying cell numbers between 2 and 30. The differences were small but best results were found at approximately 25 × 10 cells.



(a) (b)

Figure 41. The positive effect of rotating the diamonds blocks. (a) shows the blocks at 0° and in (b) they have been rotated by 10°. The maximum skewness was improved from 0.758 to 0.657.

5.1.3 Limitations on Geometry

The areas of concern in terms of quality discussed in section 5.1.2 give rise to some important limitations on geometry. The most critical of which are the wing sweep and inlet ramp wall angle. The criterion used to define allowable geometry was a maximum skewness of 0.85.

The wing sweep angle may be varied between 60° and 70°. This is a problem given the desired optimisation domain considers sweep angles from 60° to 85°, according to the work completed by Jazra (2010). It does allow the baseline of Jazra (2010) but not the optimised geometry of Preller and Smart (2012).

The wall angle of the engine inlet ramp should be at most 5° at low sweeps and then 0° at high sweep. Although not written in the code, it is recommended that this angle be chosen by linearly interpolating between 0° and 5° for sweeps from 70° to 60° as follows:

$$\theta_{inletRamp} = 5 \cdot \frac{\phi_{sweep} - 60}{70 - 60}$$

The boat tail radius was limited to be greater than 0.45 m. This is dependent on the wing geometry so is a loose constraint depending on context.

During the forebody study (section 5.2), several forebody parameters were varied and limitations were found. In particular, the aspect ratio is limited to 2.5. This is because of high skew in the nose tip blocks. There is no reason however why these blocks may not be redefined for special cases. The longitudinal camber was approximately limited as in Preller (2016), that is to a maximum of ${}^{1}/_{4}\, r_{fb}$ at up to half chord from the nose. The transverse camber was also limited to less than about $0.15 r_{fb}$. These are quite generous and again, only limited in the extreme cases due to high skew in the nose tip blocks.

These limitations demonstrate a number of things. Firstly, the reasonable parametric ability of the tool. With the exception of the halfspan (wing sweep), the remainder of the parameters may be changed substantially (Figure 42Figure 42. Four examples of the variation in vehicle geometries. These examples also show how the initial grid will likely need some adjustment to achieve a high quality grid.). Furthermore, the robustness of the code (and lack thereof) is also highlighted.

In changing to extreme geometry the grid will generate successfully. However, in the first instance the grid will likely not be acceptable and modifications may have to be made. Several conditional statements designed to catch problematic geometry are included in the code. These print warnings when the above limitations are caught but do not interrupt grid generation.

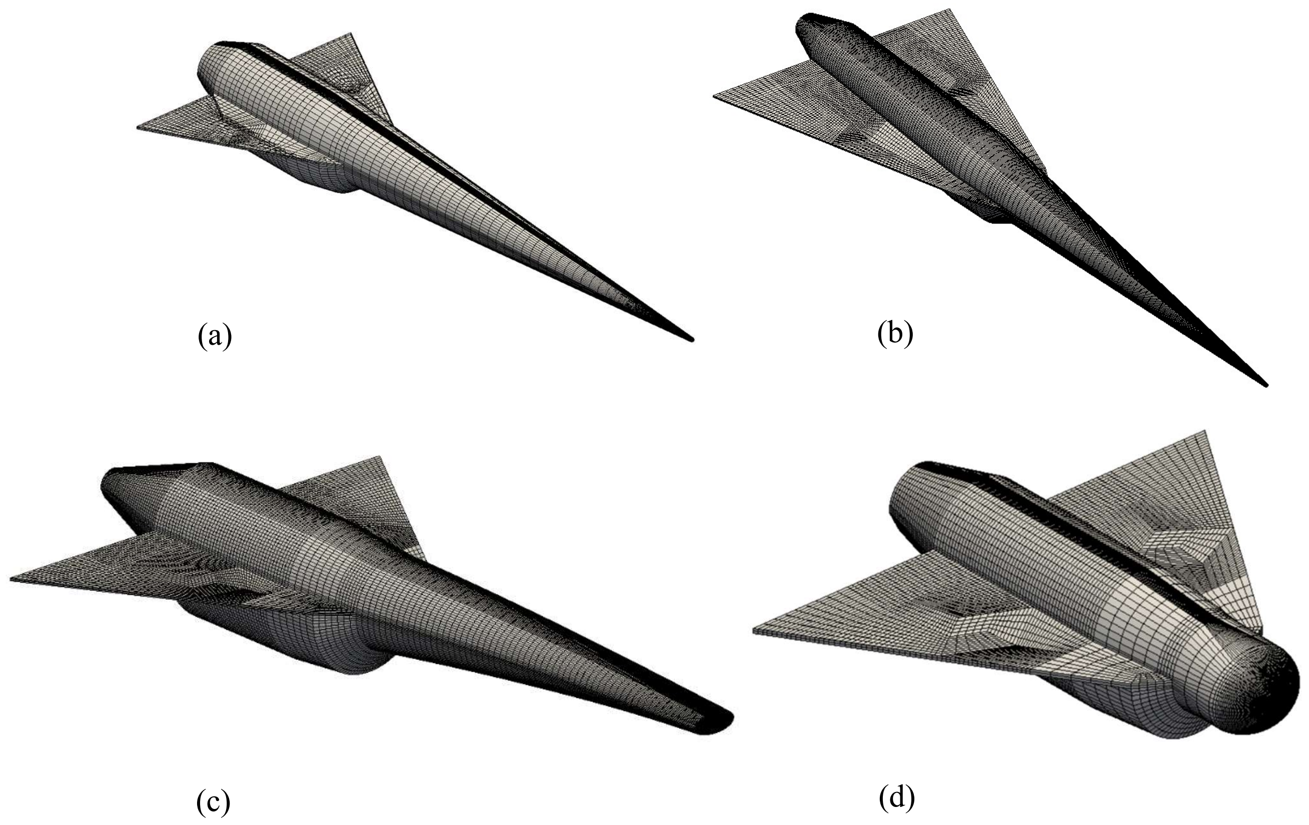


Figure 42. Four examples of the variation in vehicle geometries. These examples also show how the initial grid will likely need some adjustment to achieve a high quality grid.

5.2 Baseline SPARTAN

Flow Conditions and Simulation Parameters

All simulations were run at an angle of attack of 4.86° (corresponding to a negative rotation about *x*), a freestream Mach number of 6.0 and a static pressure of 1.94 kPa. This condition was chosen from an optimised trajectory to LEO and corresponds to the scramjet takeover point (Preller, Smart & Schutte, 2016). The grid used is that shown in Figure 34 with associated quality shown in Figure 37.

In order to reduce the computational time, viscous effects were neglected. This is a poor assumption, with the flow field around slender hypersonic vehicles dominated by viscous effects (Hirschel & Weiland, 2011). Although the viscous effects on lift at low angle of attack are small, on the order of 1% (Shaughnessy et al., 1990), they have a much stronger impact in the direction of travel. Skin friction drag makes up approximately 25%, or more, of the total drag (Jazra & Smart, 2010). However, considering the extra computational effort, in part caused by the extra finer cells to resolve viscous effects, this was an unfortunate but necessary approximation. Furthermore, without some form of grid independence study or comparison to existing data (HYPAERO for example), results should be considered provisional and simply a proof-of-concept.

Because of the regions of poor quality, several steps were taken to ensure the stability of the solver. The changes made were on the extreme side to ensure stability while accepting the increase in computational time. Firstly, the initial time step (`dt`) and CFL number (`cfl`) were reduced to 1.0×10^{-8} and 0.2 respectively. Secondly, the steps between the CFL check (`cfl_count`) was reduced to eight and the check was performed over the smallest cross-cell distance (`stringent_cfl=1`). Time history convergence of the energy and mass residuals is shown in Figure 43. The maximum residual in mass and energy were both less than 1.6e-4.

The simulation was run for 2.703e-3 seconds or 199,320 steps. This is equivalent to a characteristic flowtime based on vehicle length of only 0.28.

In an attempt to reduce the runtime required for a reasonable approximation to the solution, the simulation was initialised with conditions the same as the free stream. This required a runtime of 24 hours on two nodes of the Goliath cluster giving 32 CPU cores.

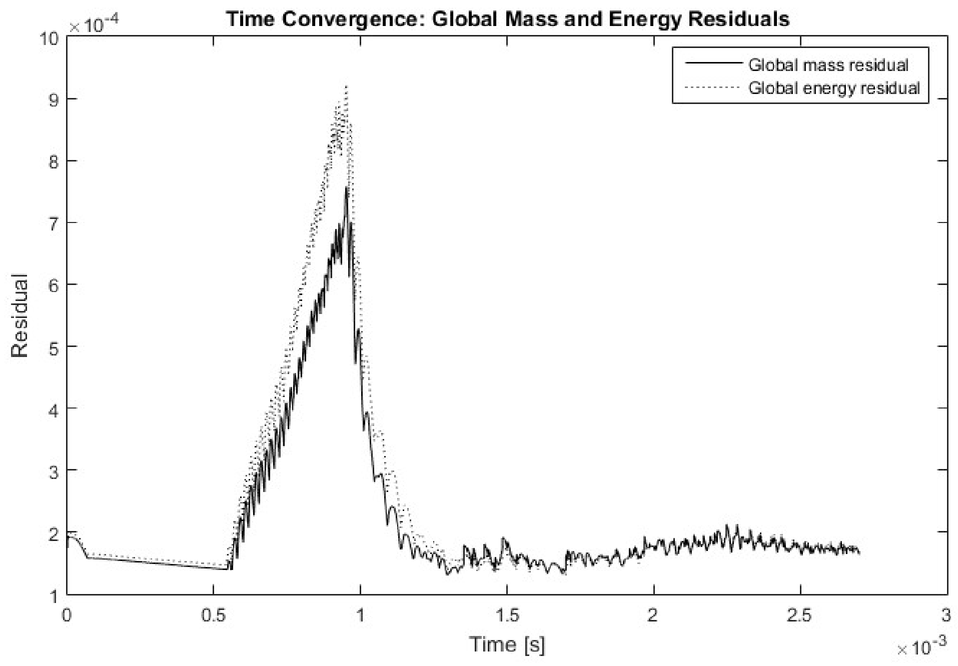


Figure 43. Time history of the global maximum energy and mass residual for the full vehicle simulation. The residuals were extracted from the Eilmer log files using the Python script in Appendix C.

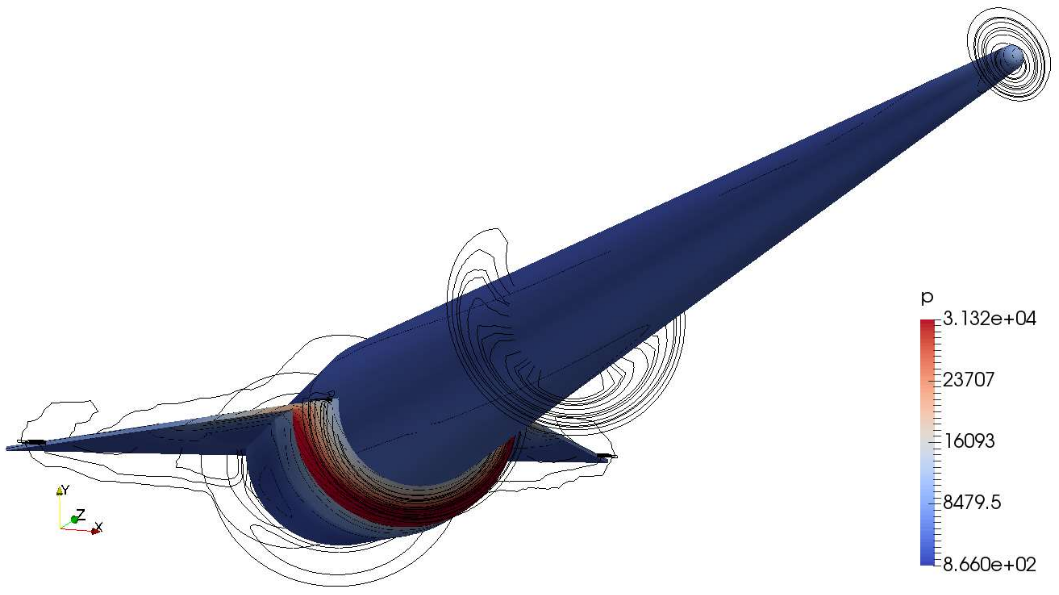


Figure 44. A 3D Paraview rendering of the full vehicle simulation. This surface of the vehicle is coloured by pressure and shows lines of constant pressure. The contours are uniformly distributed over the range shown. They are plotted at $z = 0, \text{-}5, \text{-}10, \text{-}15$ and $\text{-}20$ m from the nose.

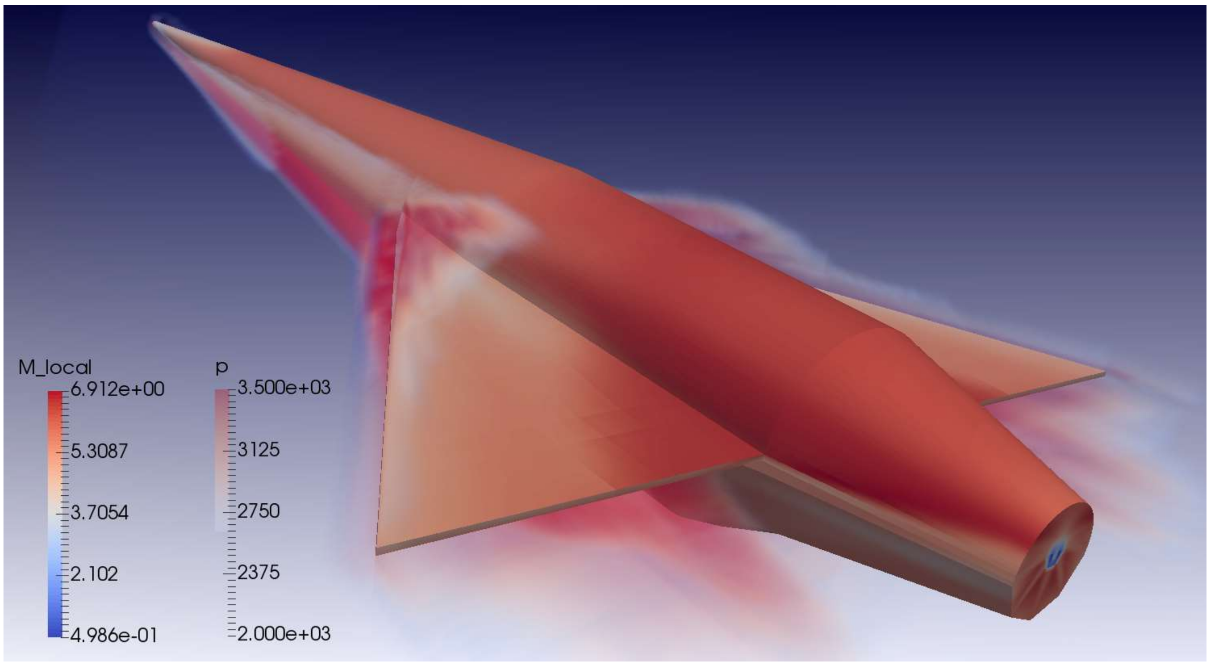


Figure 45. The vehicle surface is coloured by Mach number and the volume is coloured by pressure. In an attempt to visualise the near body pressure and shocks, an opacity function has been applied to hide the free stream pressure.

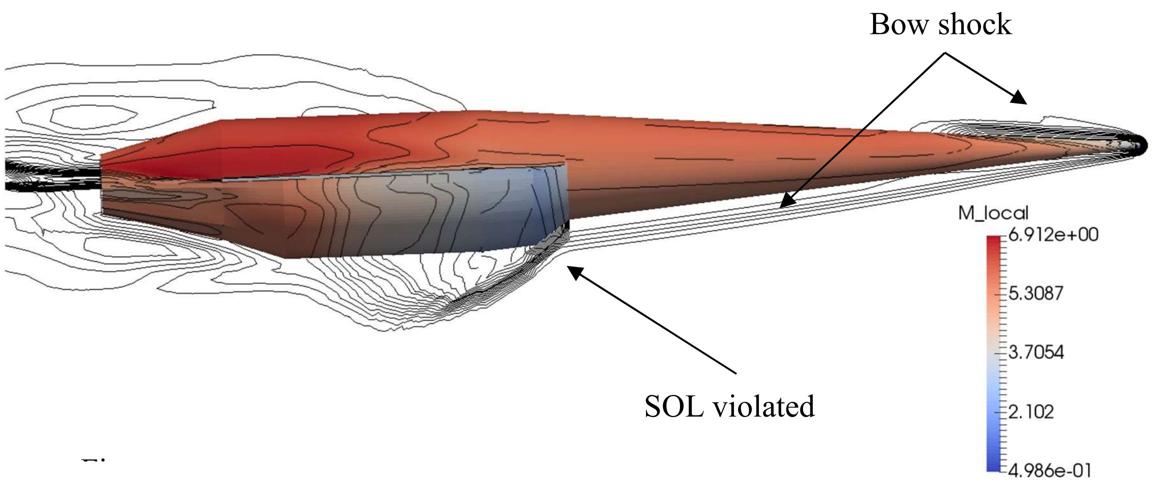


Figure 46. Contours of constant Mach number. The vehicle surface is also coloured by Mach number.

Figures 44, 45, and 46 display the results for the full vehicle simulation. Figure 44 shows the vehicle surface coloured by pressure and contours of constant pressure at cross sections down the vehicle. Pressure distributions over the surface of the vehicle are useful for the assessment of structural loads and venting regions (Gnoffo et al., 1999).

Figure 45Figure 45. The vehicle surface is coloured by Mach number and the volume is coloured by pressure. In an attempt to visualise the near body pressure and shocks, an opacity function has been applied to hide the free stream pressure. shows Mach number and a volume visualisation of the pressure distribution around the vehicle. An opacity function was applied here to hide the free stream pressure. This shows the approximate shock structure near to the vehicle. A small area of stagnant air is also visible behind the boattail as expected due to the blunt trailing edge. This will contribute substantially to the drag of the vehicle (Eggers et al., 1995).

The plot of constant Mach contours in Figure 46 further implies the vehicle has violated the shock-on-lip condition. This is shown in the tight collection of contours hitting the inlet and the second shock at a higher angle. This is in direct contradiction to the previous findings from HYPAERO results. However, as stated, these results must be considered provisional.

5.3 Forebody Results

Owing to the exceedingly long runtimes required for the full vehicle simulations, several simulations were run of just the forebody. The baseline and a proposed modified geometry were evaluated at the flow condition specified in section 5.2. A modified forebody was devised which drew inspiration from the recommendations in Bing, Chun-lin & Liang-xian (2015), Hirschel & Weiland (2009) and Varvill & Bond (2003). The result is a flattened body with a blunted nose and slight positive longitudinal camber. A summary of the baseline and modified forebody geometries is given in Table 5. These are illustrated in Figure 47.

Table 5. Geometric parameters of the baseline and modified forebodies.

Parameter	Baseline	Modified
AR	1.0	2.0
r_{nose} [m]	0.05	0.1
$C_{a,Long}$ [m]	0.0	0.03
$C_{a,Trans}$ [m]	0.0	0.15

Both simulations were run on the same grid of 177,045 cells. Run time was approximately 17 hours using 8 CPU cores for a simulation time of 2.581e-2 seconds (3 flow-times). As the grid quality was much improved, the stability control measures described in section 5.2 were relaxed. Global mass and energy residuals for the two simulations are shown in Figure 48.

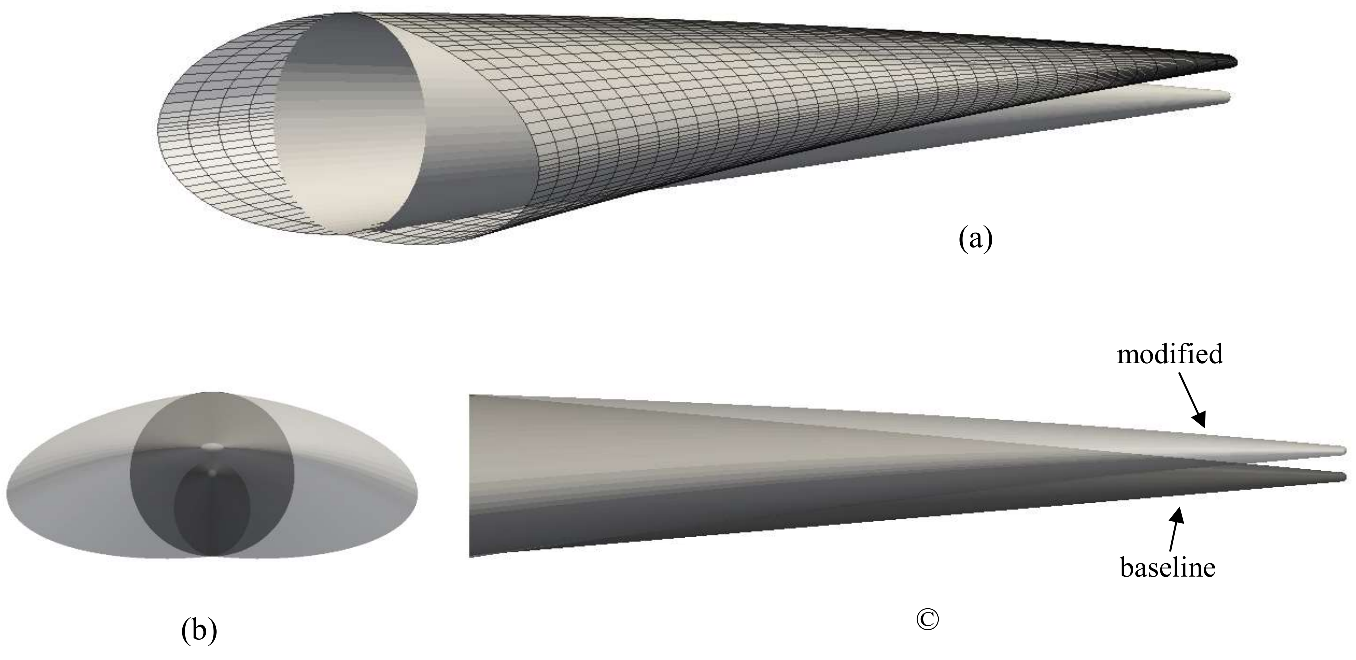


Figure 47. The baseline and modified forebody geometries used in this study. (a) isometric view, the modified shows the surface grid, (b) front and (c) side views. The baseline is shown in a slightly darker colour.

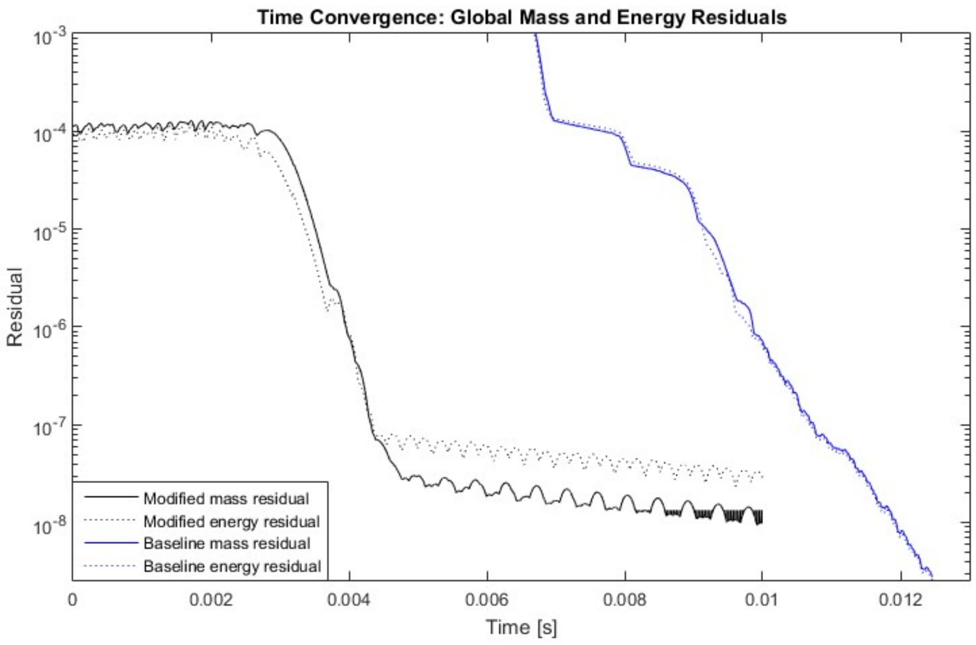


Figure 48. Global mass and energy residuals for the modified and baseline forebody simulations. Note the discrepancy in start time is due to slightly different initial CFL and time step values.

5.3.1 Baseline

Figure 49 shows the inlet coloured by horizontal velocity and vehicle surface coloured by pressure. The windward side (bottom) shows progressively higher pressure as the forebody increases in size, compressing the flow. The maximum pressure at the inlet plane was around 6.5 kPa. It should be noted that in this plot blue refers to the velocity of greatest magnitude (blue because of the forebody side defined) and the large areas just outside the symmetry plane imply strong divergence in the inlet onset flow. This is confirmed in Figure 50 where the velocity streamlines over the surface of the forebody are shown.

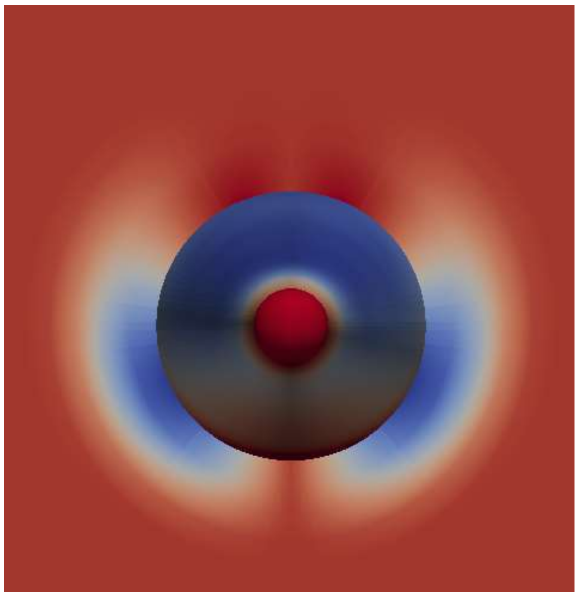


Figure 49. Horizontal velocity contours with the forebody surface coloured by pressure. Note blue corresponds to high velocity.

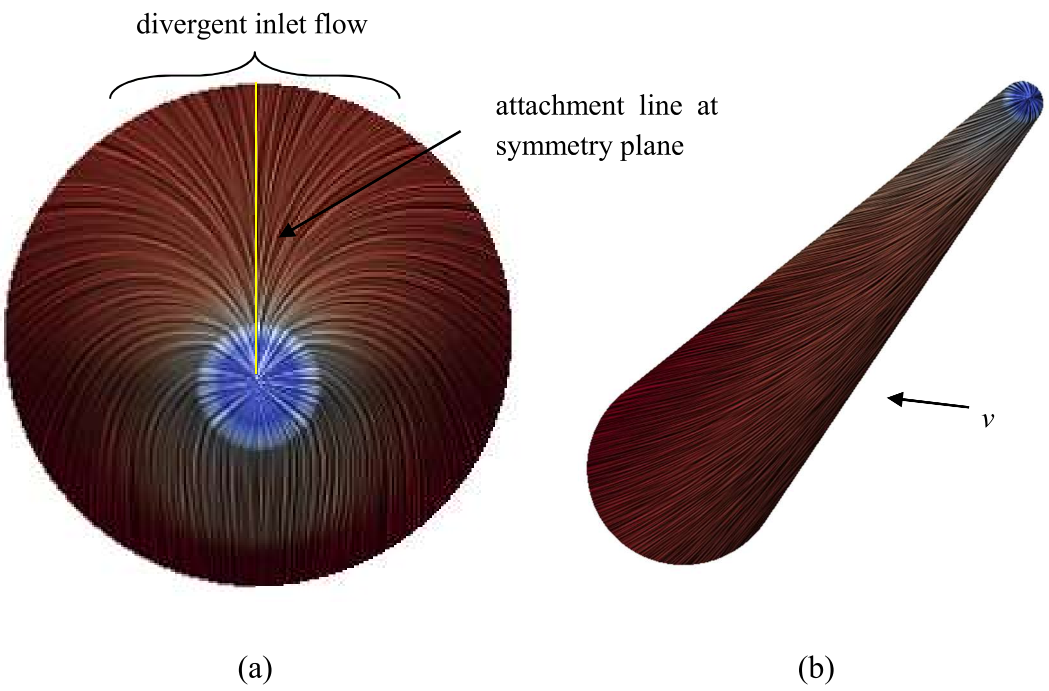


Figure 50. Surface streamlines coloured by velocity magnitude for the baseline inlet. Note that the forebody is inverted in (a), showing the windward side.

5.3.2 Modified Forebody

Figure 51 shows horizontal velocity plotted at the inlet capture plane. An investigation of the results showed the approximate horizontal velocity at the centre of the two middle engines ($x = \pm 0.263$) was reduced from over 120 m/s to around 55 m/s thus implying the engine capture rate is improved. Additionally, an increase in pressure from approximately 6.5 kPa to 9.5 kPa was observed for the modified forebody, likely from the stronger bow shock.

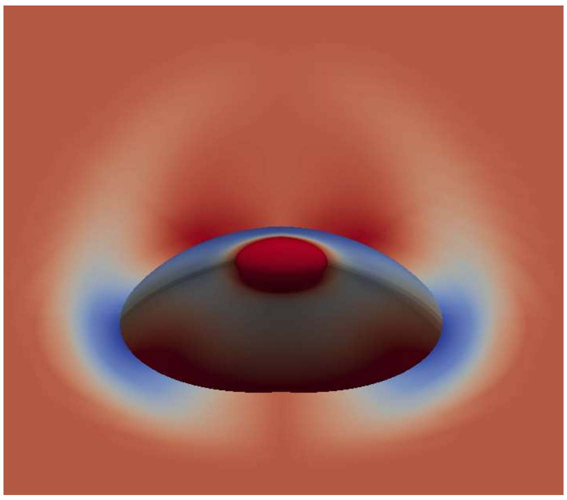


Figure 51. Horizontal velocity plotted at the inlet plane (z = -9.46 m) and the vehicle surface coloured by pressure. In this plot, the blue corresponds to high negative velocities.

The total internal volume of the modified forebody has increased potentially allowing for an increase in fuel given the packaging described by Preller (2016). This increase is on the order of 100%, to 16.25 m^3 (estimated from the aspect ratio of $AR = 2$). However without some rearrangement the useful volume has likely only increased by a small amount given the complexity of non-cylindrical fuel tanks.

Figure 52 may be used to further visualise the flow around the forebody. In particular, when compared to 50(a), the modified geometry clearly shows the improved straightness of the flow at the inlet. The streamlines are now reminiscent of Hirschel & Weiland's forebody design recommendations (2009) in Figure 6.

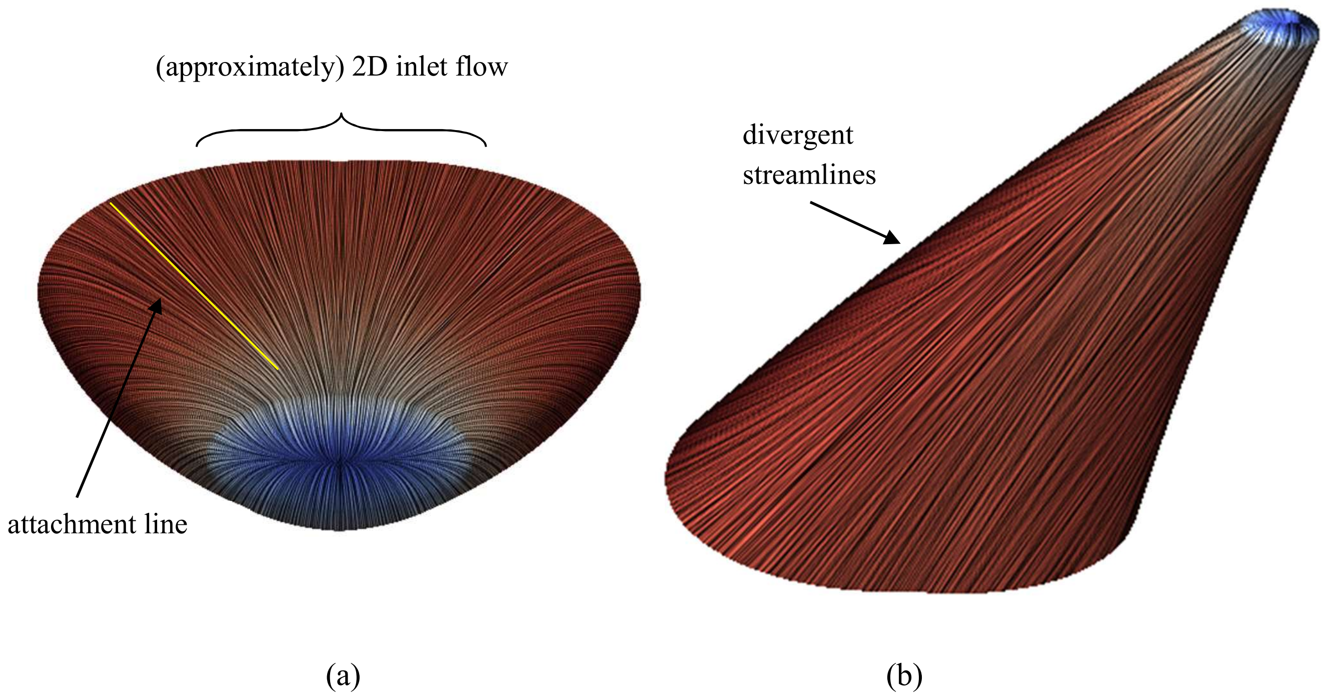


Figure 52. Velocity streamlines on the surface of the modified forebody. Note in (a) the forebody is upside down to show the windward side and straightness of the flow. This figure was generated using the SurfaceLIC filter.

6 Recommendations

Following the discussion in section 5, several shortcomings and limitations have been identified which will prevent the complete success of the tool. Recommendations for future work addressing each of these are given here.

6.1 Utilising a Patched Grid in Eilmer4[8]

As discussed, the root fairing block at the wing leading edge is the critical block in terms of quality and thus limits the geometry. This could be improved by using a section of unstructured mesh and patching it in with the remainder of the structured grid. A recent thesis at The University of Queensland has successfully developed an unstructured mesh generation tool of more than acceptable quality (Muir, 2016). This tool is currently restricted to 2D and there are significant obstacles before 3D generation is realised. On one hand, axisymmetric and extrusion methods for 3D blocks aren't expected to be overly complicated but a general 3D shape is considered exceptionally difficult.

This would however be an elegant 'best of both worlds' solution to the leading edge quality issue. A proof-of-concept is presented in Figure 53. This geometry is a 2D representation of the likely worst case scenario – a wing sweep of 85° and inlet ramp angle of 5°. The neighbouring blocks are included to show how unstructured cells would fit alongside the current structured grid.

The quality plots clearly show a vast improvement in the quality of the grid in the majority of the skewed block. The worst cell however is virtually the same, with a quality of 0.935. This is because the unstructured algorithm also uses quadrilateral cells and must still place at least one cell in the highly skewed point of the block.

[8] Gollan & Jacobs, (2015) *Implementation of a compressible-flow simulation code in the D programming language*. The University of Queensland.

However, it very quickly adds cells, in what's called the seaming process, improving the quality. Histograms of cell quality are also given for the two cases. The spikes for each case are predominantly due to the neighbouring blocks. What is clear is the improvement in average cell quality as shown in the reduction in cells above a skewness of 0.85 – 25 and 87 for the unstructured and structured cases respectively. Furthermore, it should be noted that the cell count for the unstructured block was increased, meaning the overall quality is better than the histogram implies.

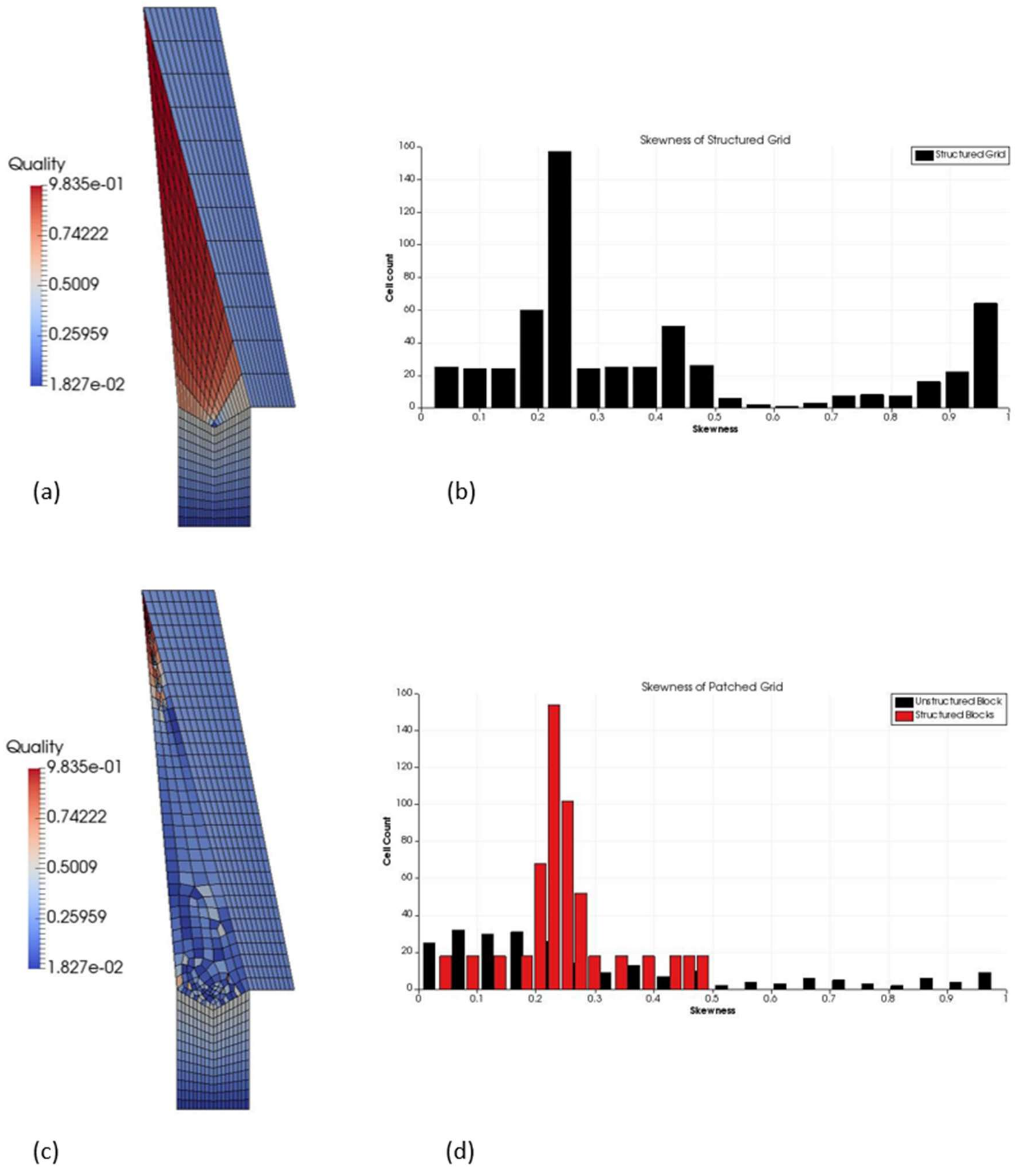


Figure 53. A 2D representation of the patched grid concept. Note in (d) the unstructured cells are shown in black and red refers to the structured cells.

In addition to improving the quality, a patched grid offers the distinct benefit of relaxing the requirement for cell counts to agree across blocks. If implemented into the grid this could also solve the inlet-inlet ramp smoothness issue. An example of the wing blocking structure showing the sections of unstructured grid is shown in Figure 54Figure 54. Suggested locations for unstructured blocks should a patched grid be used.. In addition to the root fairing blocks, those forming the elevons are selected.

The generation time however should be noted. While the structured grid was generated in a matter of seconds, the unstructured grid took over four minutes. Furthermore, the best results will be found for fine discretisation. A higher number of cells means the paving algorithm can construct better quality cells sooner improving overall quality. However this increases runtime even more. It is for these reasons (and the arguments in section 2.3.2 in favour of structured grids) that a patched, rather than entirely unstructured, grid is recommended. It should be noted that unexpected complications may arise as Eilmer has not yet been tested with unstructured grids.

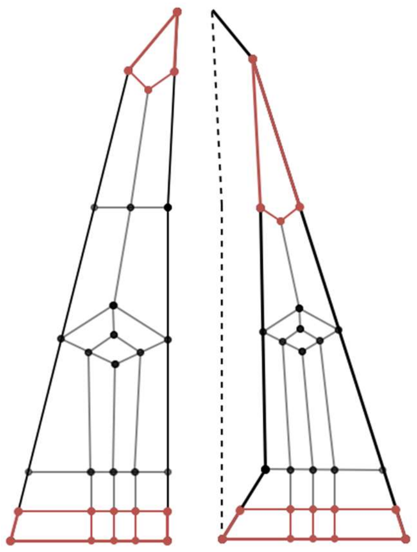


Figure 54. Suggested locations for unstructured blocks should a patched grid be used.

6.2 Improving the Grid Quality

There is always a balance between the time and effort spent on grid generation or generating a solution. With that in mind, Alter (2004) recommends generating a high quality grid as this is more time economical and allows a more efficient use of resources. With that in mind, further effort directed towards improving the grid quality are recommended in addition to the patched grid idea above. One such avenue lies in combining the parametric functionality of e3prep topology definition with a commercial grid generation program for grid generation. Advantages of this are as follows.

Firstly, e3prep was initially selected because of its programmatic nature. This allows the relatively easy parameterisation and subsequent alteration of the geometry and grid. Furthermore, it will facilitate running potentially hundreds of simulations at different flow conditions and configurations. In this scenario a master script could automatically define conditions setting up the simulations. GUI packages, in general, do not allow this level of control and functionality, favouring automation. Defining the block topology with e3prep will retain this ability.

Secondly, a grid generation program, such as GridPro, will generate a much higher quality grid. Grid generation is the focus of these programs and has been much further developed. Furthermore, smoothing, clustering and other local refinement tools may be used to control the grid. Other possible options which have not been researched include ICEM[9] or Pointwise[10] (previously GridGen).

So how would this actually work? This is a matter of getting e3prep to talk to GridPro and then GridPro to talk to e3prep. The second of these tasks has already been completed; e3prep has the ability to read and import external grids (see Eilmer user guide page 377, Jacobs et al., 2015) including GridPro (see `import_gridpro_grid`). Using e3prep to export topology is however a larger task. GridPro's surfaces and blocks are stored in TIL (section 2.3.4), similar to the C language. Before GridPro developed its GUI, its superior grids were generated by defining the topology in TIL.

[9] http://resource.ansys.com/Products/Other+Products/ANSYS+ICEM+CFD
[10] http://www.pointwise.com/pw/

Although a nontrivial task, a python program could be written to take the objects defined by e3prep and *translate* it to TIL before letting GridPro generate the grid. GridPro may also be completely run from terminal hence the automation of the process is not inhibited. After some experimentation with GridPro to generate simple 3D grids, runtime is expected to increase but this is more than acceptable given the expected improvement in simulation time. Further information on GridPro and TIL may be found in the reference manual by GridPro (2012).

6.3 Extending to Subsonic Simulations

It is likely not feasible to solve the flowfield around SPARTAN at subsonic velocities. This is because the simulation runtime will become prohibitive as the flow rebounds through the domain. ExtrapolateOutBC in particular will not handle subsonic flow leading to issues at transonic or high subsonic velocities. A better solution may be to transfer the solver duty to OpenFOAM[11]. OpenFOAM is a free, open source collection of CFD codes applied to a multitude of scenarios. The reason this is attractive is it will require very little modification. e3prepToFoam is python utility which converts e3prep generated meshes to the openFOAM foam format, including the generation of boundary patches. Details on the tool may be found in the technical report by Jahn and Qin (2015).

A small number of changes must be made in order to convert AMARA for use with OpenFOAM. Firstly, the correct OpenFOAM directory structure must exist. Secondly, setting the gas model and initial conditions is not necessary as these now reside in separate files (case/0/). Finally, the boundary conditions must be defined differently. This is done through setting all boundaries to ExtrapolateOutBC with keyword labels.

```
blk.bc_list[FACE] = ExtrapolateOutBC(label=”NAME”)
```

Where `NAME` refers to one of the following `OF_inlet_xx, OF_outlet_xx, OF_wall_xx, OF_symmetry_xx`. For example, all faces on the surface of the vehicle may be defined by the one boundary condition:

[11] http://openfoam.com/

```
blk.bc_list[FACE] = ExtrapolateOutBC(label=”OF_wall_00”)
```

Following this, the selection of a relevant solver (eg. simpleFoam, incompressible turbulent Navier-Stokes) and simulation control parameters, the simulation may be run.

6.4 Simulation Runtime

A severe impediment to this thesis was the runtime required for 3D simulations. To the author’s knowledge, AMARA represents one of the biggest grids developed using e3prep and solved with Eilmer. The full vehicle simulation, at a minimum, employs 265 blocks with the coarsest grid having over 350 000 cells. When viscous effects are included (imperative for drag and subsonic simulations), the cell count and runtime will increase substantially. In order to realistically compile an aerodynamic look up table, increased parallelisation is required such that the simulation may efficiently make use of 100+ or even more processes on Tinaroo.

Currently, the ramp blocks are many times larger thus representing a larger computational task. Therefore when distributing the load amongst more processes, simulation will reach a point where it is simply waiting on the large blocks’ calculation to finish before exchanging boundary data at the end of each time step. This will require blocks to be subdivided such that they may be distributed efficiently across more processes. Thankfully, e3prep supports the automatic subdivision through SuperBlock3D.

This was investigated however problems arose in defining the boundary conditions. The conditions do not carry over from the original block object and must be redefined individually as follows:

```
BLOCK_0 = SuperBlock3D(…)
# Iterate through the individual blocks within BLOCK_0 along k.
for blk in BLOCK_0.blks[0][0]:
   # For each block, ‘blk’ set the BCs. In this case a wall and inlet.
   blk.bc_list[SOUTH] = FixedTBC(Twall=wall_temp)
   blk.bc_list[NORTH] = SupInBC(inflow)
```

Further work is required to determine an efficient distribution of blocks. This may be investigated using the included load-balancing program. Here we may search for the most computationally efficient number of processes, in this case between 16 and 256.

```
e3loadbalance.py --job=AMARAgeom –n 16 –sweep-range=16:256
```

Generally after a point performance will plateau. By finding this point, say $n = m$, re-running e3loadbalance.py with `-n m` and then checking how the blocks are distributed, one may determine where to concentrate their efforts with SuperBlock3D. By splitting the ramp blocks into 16, e3loadbalance.py reported an improvement of over 50x for 30+ processes. Further information on load balancing may be found in Appendix K of the Eilmer user guide (Jacobs et al., 2015).

6.5 Extracting Coefficients

Although not an objective of this work, the later aerodynamic evaluation of SPARTAN and compilation of the aerodynamic look up table requires aerodynamic coefficients such as C_l C_d and C_m as well as stability derivatives. e3post includes a function to extract forces on specific block sides but this does not extend to 3D. The general methodology is as follows

1. Extract the area, pressure and vertices (points) of each cell on the surface of the vehicle.
2. Find the centre coordinates of the cell from the four points. This is where the pressure is acting, normal to the surface.

3. Form two vectors between three points and calculate the unit normal from their cross product. This assumes the cell is approximately planar. An alternative and more accurate version of this step is as follows:
    a. Form four vectors from the corners to the centre.
    b. Calculate four normals from the cross product of each neighbouring pair of points.
    c. Find the average of these, thus approximating the normal to the cell.
4. Find the force vector acting on this cell by multiplying its area by the local pressure. The direction is given by the normal.
5. Transform the cell force to a global coordinate system (wind axes).
6. Calculate the cell area in the direction required for the reference area.
7. Integrate these forces and cell areas across the entire surface of the vehicle.
8. Calculate the final coefficient.

For completeness sake, an unverified script to extract the pressure force is given in Appendix D as a starting point for a future script. This is a reasonable approximation for lift but will need to be updated to include viscous forces.

7 Conclusion

The main results and conclusions of the thesis will be reviewed against the original aims before a concise summary of the outcomes. The relevance of the outcomes to hypersonic vehicle design at UQ will also be discussed. Finally, a summary of the recommendations for future work is given.

7.1 Thesis Evaluation

Firstly the aims of the thesis will be addressed before final comments on the success of the project.

Develop an appropriate three dimensional blocking strategy
This aim was successfully met. The blocking strategy proposed was successfully implemented to allow the modelling of a generic hypersonic vehicle. Furthermore, with the work identified in section 5, the grid may be extended following the strategy suggested to include the vehicle tail, control surface deflection and further develop the wings. This strategy does however make

Construct a parametric grid for the analysis of generic hypersonic vehicles
A parametric grid was successfully developed which models a winged cone vehicle. The parametric ability was demonstrated in a range of vehicles with acceptable grid quality. However, this grid does have significant limitations and does not meet all objectives. Because of quality issues caused by the highly skewed geometry, the complete modelling of high wing sweep, control surface deflection and wing bluntness is deemed not possible with a structured grid. The use of unstructured blocks in a patched grid is suggested to overcome this problem.

Exercise the framework on SPARTAN

The grid was used to successfully model the SPARTAN baseline geometry. This was confirmed by checking dimensions against the published geometry. Simplifications and changed were made in terms of the engine and boat tail however. Further research is required to investigate the effect this has. The integration of the complicated engine inlet geometry is expected to be a significant if not impractical task, even for unstructured blocks.

Exercise the framework on a baseline and modified forebody

Two forebody geometries were successfully simulated and evaluated. The modified forebody, which was designed following recommendations from the literature, was qualitatively found to perform better than the baseline in terms of inlet flow and precompression.

The objective of this project was the development of a parametric grid generator for hypersonic vehicle analysis. The project is arguably successful given each aim has been met although with limitations and restrictions. Exhaustive recommendations on suggested future work to solve some of the issues present are given such that the knowledge and experience from this thesis is passed on.

7.2 Summary of Recommendations

Utilising a patched grid in Eilmer4[12]

A patched grid where highly skewed blocks make use of an unstructured grid generation method is recommended for the wing root and control surface blocks. This will allow the geometry to be formed.

[12] Gollan & Jacobs, (2015) *Implementation of a compressible-flow simulation code in the D programming language*. The University of Queensland.

Improving the grid quality

In order to preserve the parametric ability of python but greatly improve the grid quality, e3prep could simply be used to generate the block topology. A commercial grid generation package, such as GridPro, would then be used to generate and smooth the grid. This would give great improvements however a purely structured grid is not expected to be feasible.

Subsonic simulations

Research into a different solver code is recommended for simulation at subsonic or possibly transonic speeds. One such program which would require minimum effort and is known to be feasible is OpenFOAM. This would allow the current grid to be used with very little changes required.

Simulation runtime

In order to compile a comprehensive aerodynamic look up table for a full trajectory analysis, the simulation runtime must be improved. Increased parallelisation and taking full advantage of a cluster such as Tinaroo is recommended. This may be achieved through splitting the large blocks (ramp and outer wake blocks) and distributing with `e3loadbalance.py`.

Extracting coefficients

Finally, development of a new script to extract forces on a block boundary in 3D is required. A method has been proposed and a rough script presented without verification.

7.3 Implications

This thesis has delivered an adequate tool for the hypersonic vehicles. This may be applied to the SPARTAN geometry however the full range of geometric parameters is not possible with a structured grid. Aerodynamic evaluation of the baseline SPARTAN is feasible but will require work in the areas identified. Finally, compilation of a complete aerodynamic database across a full geometric and flight condition range is not feasible given the current methodology.

7.4 Closing Remarks

This project has required an immense amount of development and learning. It has lead to the development of new skills and a familiarity within grid generation and CFD analyses. Although not all objectives were met satisfactorily, it is satisfying to know a contribution has been made to the development of SPARTAN at UQ. It is hoped this work along with associated recommendations will foster the next generation of AMARA. Finally, a few realisations made along the way include:

The computer ***always*** *does what you* ***tell*** *it and not what you* ***want*** *it to do.*

It doesn't have to be perfect, simplifications and drawing the line somewhere are justifiable.

References

Ahmad, A. Maddock, C. Scanlon, T. Brown, R. (2011) *Prediction of the Aerodynamic Performance of Re-usable Single Stage to Orbit Vehicles*. Proceedings of Space Access 2011.

Alter, S. (2004) *A Structured Grid-Quality Measure for Simulated Hypersonic Flows*. 42nd AIAA Aerospace Sciences Meeting and Exhibit. January 5th-8th Reno, USA.

Anderson, J. (2003) *Modern Compressible Flow with Historical Perspective.* 3rd Edition. McGraw Hill Higher Education. New York, USA.

Anderson, J. (2006) *Hypersonic and High Temperature Gas Dynamics*. American Institute of Aeronautics and Astronautics. Vancouver, Canada.

Baker, D. Worden, S. (2008) *The Large benefits of Small-Satellite Missions*. Eos Trans. AGU. 89(33) p 301-302.

Berens, T. Bissinger, N. (1996) *Study on Forebody Precompression Effects and Inlet Entry Conditions for Hypersonic Vehicles*. Space Plane and Hypersonic Systems and Technology Conference, Norfolk, USA. AIAA Paper 96-4531.

Bertin, J. & Cummings, R. (2006) *Critical Hypersonic Aerothermodynamic Phenomena*. Annual Review of Fluid Mechanics. 38, 29-157.

Bing, C. Chun-lin, G. Liang-xian, G. (2015) *Design and Verification of Airframe/Propulsion Integration for Air-breathing Launch Vehicle*. 20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference. 6-9 July, Glasgow, Scotland.

Bowcutt, K. Smith, T. (2012) *Responsiveness and affordable launch of small satellites: A reusable air-breathing concept*. Reinventing Space Conference. 7-10 May, AIAA, Los Angeles, USA.

Chawner, J. (2013) *Quality and Control – Two Reasons Why Structured Grids Aren't Going Away*. Pointwise, The Connector. Viewed 23/09/16 at: <http://www.pointwise.com/theconnector/March-2013/Structured-Grids-in-intwise.shtml>

Doherty, L. Smart, M. Mee, D. (2014) *Experimental Testing of an Airframe Integrated 3-D Scramjet at True Mach 10 Flight Conditions*. 20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference, 6-9 July, Glasgow, Scotland.

Doherty, L. Smart, M. Mee, D. (2015) *Measurement of Three-Components of Force on an Airframe Integrated Scramjet at Mach 10*. 20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference. July 6-9 Glasgow, Scotland.

Dujjaric, C. Caporicci, M. Kuczera, H. Sacher, P. (1997) *Conceptual Studies and Technology Requirements for a New Generation of European Launchers*. Acta Astronautica. 41(4) p 219-228.

Duveau, P. Hallard, R. Novelli, R. Eggers, T. (1999) *Aerodynamic Performance Analysis of the Hypersonic Airbreathing Vehicle JAPHAR.* Office National D'etudes et de Recherches Aerospatiales Chatillon-Sous-Bagneux, France.

Eggers, T. Dittrich, R. Varvill, R. (2011) *Numerical Analysis of the SKYLON Spaceplane in Hypersonic Flow.* 17th AIAA International Space Planes and Hypersonic Systems and Technologies Conference. San Francisco, USA, April 2011.

Eggers, T. Strohlmeyer, D. Nickel, H. Radespiel, R. (1995) *Aerodynamic off-design behaviour of integrated waveriders from take-off up to hypersonic flight.* Proceedings of the 2nd European Symposium held in ESTEC, Noordwijk, The Netherlands. 21-25 November.

Ferguson, F. Dasque, N. Dhanasar, M. (2015) *Waverider Design and Analysis*. 20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference. 6-9 July, Glasgow, Scotland.

Gnoffo, P. Weilmuenster, K. Hamilton, H. Olynick, D. Venkatapathy, E. (1999) *Computational Aerothermodynamic Design Issues for Hypersonic Vehicles*. Journal of Spacecraft and Rockets, 26(1) p 21-43.

Gollan, R. Jacobs, P. (2013) *About the formulation, verification and validation of the hypersonic flow solver Eilmer*. Numerical Methods in Fluids. Vol 73(1), 19-57.

GridPro (2012) *Reference manual for TIL v5.5*. Program Development Corporation, NY, USA.

Hallion, R. (1998) *The Hypersonic Revolution: Case Studies in the History of Hypersonic Technology.* Vol. 2. U.S. Government Printing Office.

Hirschel, E. (2005) *Basics of Aerothermodynamics.* Springer Science & Business Media, Berlin, Germany.

Hirschel, E. H. Weiland, C. (2011) *Design of Hypersonic Flight Vehicles: Some Lessons from the Past and Future Challenges*. 16th AIAA/DLR/DGLR International Space Planes and Hypersonic Systems and Technologies Conference. 19–22 October. Springer, Bremen, Germany.

Hirschel, E. Weiland, C. (2009) *Selected Aerothermodynamic Design Problems of Hypersonic Flight Vehicles.* Springer Science & Business Media, Berlin, Germany.

Huang, W. Wang, Z. Luo, S. Liu, J. *An overview of research on engine/airframe integration for hypersonic waverider vehicles*. Journal of Solid Rocket Technology. 32(3) p 242-248.

Jacobs, P. Gollan, R. Denman, A. O'Flaherty, B. Potter, D. Petrie-Repar, P. & Johnston, I. (2012) *Eilmer's Theory Book: Basic Models for Gas Dynamics and Thermochemistry.* Mechanical Engineering Report 2010/09. The University of Queensland.

Jacobs, P. Gollan, R. Jahn, I. Potter, D. and others, (2015) *The Eilmer3 Code: User Guide and Example Book 2015 Edition*. Mechanical Engineering Report 2015/07. The University of Queensland.

Jazra, T. (2010) *Optimisation of Hypersonic Vehicles for Airbreathing Propulsion*. PhD Thesis. School of Mechanical and Mining Engineering. The University of Queensland.

Jazra, T. Preller, D. Smart, M. (2013) *Design of an Airbreathing Second Stage for a Rocket-Scramjet-Rocket Launch Vehicle*. Journal of Spacecraft and Rockets, 50(2) p 411-422.

Jazra, T. Smart, M. (2009) *Development of an Aerodynamics Code for the Optimisation of Hypersonic Vehicles*. 47th AIAA Aerospace Sciences Meeting. January 5-8 Orlando, USA.

Katz, (2009) *Meshless Methods for Computational Fluid Dynamics.* PhD Thesis, Stanford University, Stanford, USA.

Kinney, D. (2004) *Aero-Thermodynamics for Conceptual Design*. 42nd AIAA Aerospace Sciences Meeting and Exhibit. January 5-8, Reno, Nevada.

Knupp, P. (2008) *Measurement and Impact of Mesh Quality.* 46th AIAA Aerospace Sciences Meeting and Exhibit. January 7th-10th Reno, USA.

Mavriplis, D. (2008) *Unstructured-Mesh Discretizations and Solvers for Computational Aerodynamics.* AIAA Journal. 46(6) p 1281-1298.

Mehta, U. Aftosmis, M. Bowles, J. Pandya, S. (2015) *Skylon Aerodynamics and SABRE Plumes*. 20th AIAA International Space Planes and Hypersonic Systems and Technologies Conference. Glasgow, Scotland, July 2015.

Muir, H. (2016) *An Unstructured Mesh Generation Code for Eilmer4*. BE Thesis. The University of Queensland.

Papadopoulos, P. Venkatapathy, E. Prabhu, D. Loomis, M. Olynick, D. (1999) *Current grid-generation strategies and future requirements in hypersonic vehicle design, analysis and testing.* Applied Mathematical Modelling. 23, p 705-735.

Pezzela, G. (2012) *Aerodynamic and aerothermodynamic design of Future Launchers Preparatory Program concepts*. Aerospace Science and Technology 23, p 233-249.

Pirzadeh, S. (2010) *Advanced Unstructured Grid Generation for Complex Aerodynamic Applications*. AIAA Journal. 48(5), 904-915.

Preller, D. Smart, M. Schutte, A. (2016) *Dedicated Launch of Small Satellite.* SPACE Conferences and Exposition. September 13-16, California, USA.

Robinson, J. Martin, J. (2008) *An Overview of NASA's Integrated Design and Engineering Analysis (IDEA) Environment*. 6th Modeling and Simulation / 4th Liquid Propulsion / 3rd Spacecraft Propulsion Joint Subcommittee Meeting. December 8-12 Orlando, Florida.

Savino, R. Russo, V. Carandente, V. D'Oriano, V. (2014) *HyPlane for space tourism and business transportation.* 65th International Astronautical Congress. Toronto, Canada, September 2014.

Shaughnessy, J. Pinckney, Z. McMinn, J. Cruz, C. Kelly, M. (1990) *Hypersonic Vehicle Simulation Model: Winged-Cone Configuration*. NASA TM102610.

Smart, M. Tetlow, M. (2009) *Orbital Delivery of Small Payloads Using Hypersonic Airbreathing Propulsion*. Journal of Spacecraft and Rockets. Vol 46, No. 1, pp 117-125.

Tannehill, J. Anderson, D. Pletcher, R. (1997) *Computational Fluid Mechanics and Heat Transfer*. 2nd Edition. Taylor & Francis. Vermont, USA.

Varvill, R. Bond, A. (2003) *A comparison of propulsion concepts for SSTO reusable launchers*. Journal of the British Interplanetary Society, vol 56, pp 108-117.

A. AMARA on a Cluster

The following is some brief instructions to build and run the code on the Goliath or Tinaroo computing clusters. Aside from available resources, Goliath and Tinaroo differ in their respective queuing systems, slurm and qsub. Further information on these machines may be found at https://help.eait.uq.edu.au/compute/ and http://www2.rcc.uq.edu.au/hpc/guides/ respectively.

Firstly, the Eilmer3 codes must be retrieved and installed to your directory on the machine. Instructions on how to do this may be found in the Eilmer userguide or at http://cfcfd.mechmining.uq.edu.au/eilmer3.html. Information on terminal commands may also be found in the userguide, Appendix B. Be careful in setting the environment variables in the bashrc script correctly. Finally be aware of storage restrictions – a coarse vehicle simulation writing a solution 20 times amounts to around 550 MB.

Files may be easily transferred back to your local machine using WinSCP on Windows or FileZilla on Unix. Alternatively, the command `scb` may be used in a terminal. Be aware that if you’re off campus, you must first log into the UQ VPN or ssh into an on campus server (such as moss.labs.eait.uq.edu.au or remote.labs.eait.uq.edu.).

Both clusters require you to submit jobs to a scheduler which executes the job depending on resources available and requested. This is most easily done through a bash script. Two examples of bash scripts for Goliath and Tinaroo are presented below.

In this example, one compute node and its 16 available processes are requested. The grid is generated with e3prep before running e3loadbalance.py to generate the mpimap file. The MPI version of Eilmer is called and then AMARAsurf.py to extract the vehicle surface mesh.

e3post is used to convert to a Paraview friendly file, add the surface meshes and local Mach number. In each case the eilmer process is written to LOGFILE and may be viewed using the command `tail LOGFILE` to check on the process of a simulation.

amara_run.sh

```
#!/bin/bash
# set the number of nodes and processes per node
#SBATCH --nodes=1

# set the number of tasks (processes) per node.
#SBATCH --ntasks-per-node=16

# set max wallclock time
#SBATCH --time=24:00:00

# set name of job
#SBATCH --job-name=amara_Test

# mail alert at start, end and abortion of execution
#SBATCH --mail-type=ALL

# send mail to this address
#SBATCH --mail-user=<email@uq.edu.au>

module load mpi/openmpi-x86_64
e3prep.py --job=AMARAgeom
e3loadbalance.py –job=AMARAgeom –n 16
mpirun -np 16 e3mpi.exe --job=AMARAgeom --mpimap=AMARAgeom.mpimap –
run > LOGFILE
python AMARAsurf.py AMARAblock.py
e3post.py --job=AMARAgeom.py --vtk-xml --add-mach --tindx=last –
surface-list=”< AMARAblock_surfaceList.txt”
```

The above bash script, `amara_run.sh`, may then be queued using:

```
$ sbatch amara_run.sh
```

Before running on Tinaroo, several modules should be added to your .bashrc file. This may be edited using the vi or Nano programs.

```
module purge
module load intel_mpi
module load python
module load GCC
```

amara_run.qsub

```
#PBS -N amara
#PBS -m n
#PBS -A UQ-EAIT-MechMining
#PBS -l nodes=2:ppn=24,mem=120gb
#PBS -l walltime=24:00:00

cd $PBS_O_WORKDIR

echo "Start time:"
date

e3prep.py --job=AMARAgeom.py

e3loadbalance.py –job=AMARAgeom.py –n 48

```

```
mpirun -np 48 e3mpi.exe --job=AMARAgeom --mpimap=AMARAgeom.mpimap
--run > LOGFILE

echo "Finish time:"
date
```

The above qsub script may then be placed in the queue using:

```
$ sbatch amara_run.qsub
```

B. AMARAsurf.py

This script is used to extract all the surfaces of a specified boundary condition. The boundary condition is defined on line 45 and currently erroneously set to ‘BC’. The output format is “blk,surface-name;…” in the correct format for the –surface-list command of e3post.py. The SOUTH face of block 45 for example is written as “45,SOUTH”.

```
1    """ AMARAsurf.py
2
3    Alex Ward & Daniel Ward, 2016
4
5    This file iterates through the block looking for the boundary
6    condition keyword specified on line 45. It then returns a file,
7    AMARAblock_surfaceList.dat giving a list of block numbers and
8    face name (NORTH, EAST etc) for all surfaces using that
9    boundary condition.
10   Note it currently DOES NOT work correctly for SuperBlock3D
     objects.
11
12   Usage
13   # Run AMARAsurf.py, giving it your file specifying the blocks.
14   $ python AMARAsurf.py AMARAblock.py
15
16   # Paste content of AMARAblock_surfaceList.dat into the e3post
     command:
17   $ e3post.py --job=AMARAgeom --vtk-xml --surface-list="<output>"
18   """
19
20   import sys
21   import os
22
```

```
def parse(parseFilePath):
    #Lines to search after occurence of Block3D or SuperBlock3D
    searchDist = 15
    outputFilePath = parseFilePath.split('.')[0] + "_surfaceList.dat"
    if os.path.isfile(outputFilePath):
        os.remove(outputFilePath)
    with open(outputFilePath, 'a') as out:
        with open(parseFilePath, 'r') as f:
            lines = f.readlines()
            b3dCount = 0
            counts = []
            words = []
            linenums = []
            for i, line in enumerate(lines):
                if 'Block3D' in line:
                    if '#' in line: continue
                    startTBCSearch = i+1
                    foundTBC = False
                    for j in range(startTBCSearch, startTBCSearch + searchDist):
                        if j > len(lines) - 1 or 'Block3D' in lines[j]:
                            break
                        if '#' in lines[j]: continue
                        if 'BC' in lines[j]:
                            foundTBC = True
                            try:
                                w = lines[j].split('[')[1].split(']')[0]
                            except:
                                w = ''
                                print "Bad Bracket Split / KeyWord"
                            if w in ['BOTTOM', 'WEST', 'SOUTH', 'TOP', 'NORTH', 'EAST']:
                                counts.append(b3dCount)
                                linenums.append(j)
                                words.append(w)
                            else:
                                print "bad line:", j
                    b3dCount += 1
                    if not foundTBC:
                        continue
                        print "No 'FixedTBC' in", str(searchDist), "lines following 'Block3D' found at: line:", i
            print "Lines searched / Processed: ", len(lines)
        writeStr = ''
        for i,count in enumerate(counts):
            #Output file
            writeStr = writeStr + str(count) + ',' + str(words[i]) + ';'
            #Linenum debug
            #writeStr = writeStr + str(count) + ',' + str(words[i]) + ',' + str(linenums[i] + 1) + ';\n'
        writeStr = writeStr[:-1]
        out.write(writeStr)

if __name__ == '__main__':
    argv = sys.argv
    parse(argv[1])
```

C. getResiduals.py

```python
"""
getResiduals.py
    Retrieve and save the time history of the residuals in a friendly format for later printing.
Usage
    python getResiduals.py e3mpi.xxxx.log

A. Ward (original AWK script by M. Coombes)
"""

import sys
import os

def parse(parseFilePath):
    outputFilePath = parseFilePath.split('.')[0] + "_residuals.dat"
    if os.path.isfile(outputFilePath):
        os.remove(outputFilePath)
    with open(outputFilePath, 'a') as out:
        with open(parseFilePath, 'r') as f:
            lines = f.readlines()
            massResiduals = []
            massTimes = []
            energyResiduals = []
            energyTimes = []
            for line in lines:
                if 'mass global' in line:
                    words = line.split()
                    massResiduals.append(float(words[4]))
                    massTimes.append(float(words[8]))
                if 'energy global' in line:
                    words = line.split()
                    energyResiduals.append(float(words[4]))
                    energyTimes.append(float(words[8]))
        out.write(str(massResiduals))
        out.write('\n')
        out.write(str(massTimes))
        out.write('\n')
        out.write(str(energyResiduals))
        out.write('\n')
        out.write(str(energyTimes))
    f.close()

if __name__ == '__main__':
    argv = sys.argv
    parse(argv[1])
```

D. getForce.py

This script is used to extract the force data from a 3D Eilmer grid. PLEASE NOTE: This script is WITHOUT validation and problems are expected in its selection of the order to cross the vectors defining the cells. It is included here to provide the starting point for a future script should the opportunity arise.

```
getDrag.py
# A.WARD, D.KING, D.WARD 2016

"""
This is the start of a script to pull out lift and drag forces
for a three dimensional VTK grid. Harder than it seems at first!
"""

import os
import numpy as np

# SET DIRECTORY WHERE YOUR VTK GRID FILES ARE
directory = ''

os.chdir(directory)

'list of unknown length currently, summate at end to get total
force'
xforce_block_list = []
x = np.array([1.0, 0.0, 0.0], dtype=float)

for filename in os.listdir(directory):

```

```python
    if filename.endswith('.vtu'):

        with open(filename, 'r') as f:
            lines = f.readlines()

            '''points on line 2 = line 1 in python'''
            points_position_start = lines[1].index('NumberOfPoints="') + len('NumberOfPoints="')
            points_position_end = lines[1].index('" NumberOfCells')
            blocks_positon_start = lines[1].index('NumberOfCells="') + len('NumberOfCells="')
            blocks_positon_end = lines[1].index('">')
            points = int( lines[1][points_position_start : points_position_end ] )
            blocks = int( lines[1][blocks_positon_start : blocks_positon_end ] )
            '''
            create empty matricies
            3d array block length with 4 points stacked vertically
            '''
            blockData = np.zeros( (blocks, 4, 3), dtype=float )
            pointData = np.zeros( (points, 3), dtype=float )
            connectionData = np.zeros( (blocks, 4), dtype=int )
            directionVectors = np.zeros( (blocks, 3), dtype=float )
            rhos = np.zeros( (blocks), dtype=float)
            pressures = np.zeros( (blocks), dtype=float)
            velos = np.zeros( (blocks, 3), dtype=float)
            xforces = np.zeros( (blocks), dtype=float)

            '''get the  data on the point xyz position for corners of blocjs'''
            i = 4
            while i < (4 + points):
                data = lines[i].split()
                pointData[ (i - 4): ] = [ float(data[0]), float(data[1]), float(data[2]) ]
                i += 1

            '''add 4 to get to connectivity'''
            i += 4
            nLocal = i
            while i < ( nLocal + blocks ):
                data = lines[i].split()
                connectionData[ (i - nLocal): ] = [ int(data[0]), int(data[1]), int(data[2]), int(data[3]) ]
                i += 1

            '''This is not the best sol'''
            while i < 20000:
                if 'Name="rho"' in lines[i]:
                    nLocal = i
                    i += 1
                    break
                i += 1

            while i < ( nLocal + blocks ):
                rhos[ (i - nLocal ) ] = float(lines[i])
                i += 1


            while i < 20000:
                if 'Name="p"' in lines[i]:
                    nLocal = i
                    i += 1
                    break
                i += 1

            while i < ( nLocal + blocks ):
                pressures[ (i - nLocal ) ] = float(lines[i])
                i += 1


            while i < 20000:
                if 'Name="vel.vector' in lines[i]:
                    nLocal = i
                    i += 1
                    break
                i += 1

            while i < ( nLocal + blocks ):
                data = lines[i].split()
                velos[ (i - nLocal ): ] = [ float(data[0]), float(data[1]), float(data[2]) ]
                i += 1


            '''create these blocks'''
            for j in xrange(0, len(connectionData)):
                data = connectionData[j]
                blockData[j][0] = pointData[ data[0] ]
                blockData[j][1] = pointData[ data[1] ]
                blockData[j][2] = pointData[ data[2] ]
                blockData[j][3] = pointData[ data[3] ]


                '''Going to assume ADxAB would be the outwards direction for the moment'''
                AB = pointData[ data[1] ] - pointData[ data[0] ]
                AD = pointData[ data[3] ] - pointData[ data[0] ]

                vector = np.cross(AD, AB)

                Area = np.linalg.norm(vector)

                directionVectors[j] = vector/np.linalg.norm(vector)

                xforces[j] = Area*pressures[j]*-1*np.dot( directionVectors[j], x ) + rhos[j]*Area*( ( np.linalg.norm(velos[j]) )**2 )*-1*np.dot( directionVectors[j], x )

                j += 1

    xforce_block_list.append( np.sum(xforces) )

XFORCE = sum(xforce_block_list)
print XFORCE
```