Escola Superior d'Enginyeries Industrial, Aeroespacial
i Audiovisual de Terrassa

KREIOS - Development of a free molecular flow solver for ABEP intake optimization

By

Roger Jordi Martínez Pardell

Supervisded by

Manel Soria Guerrero Max Amer Viñas

Arnau Miró Jané

Academic Year 2023-2024

Abstract

Very Low Earth Orbit (VLEO) has emerged as a promising domain for advancing satellite technology, offering significant advantages in earth observation and telecommunications. To sustain operations at these altitudes, Atmosphere-Breathing Electric Propulsion (ABEP) systems, which utilize air intakes to gather and compress atmospheric particles for propulsion, could be crucial. This thesis focuses on the development of a collisionless flow solver tailored for shape optimization of the intake of KREIOS' ABEP engine. The main objective is to develop a Python-based computational tool that can model free molecular flow by simulating the trajectories of gas particles through various intake geometries. The methodology involves designing two computational approaches, with emphasis on a simpler approach to provide preliminary results. Although the integration of the solver into a comprehensive shape optimization loop and the implementation of full-scale computational efficiency improvements are beyond the current project's scope, this work establishes a foundational tool for future development. The results demonstrate the feasibility of collisionless flow modeling for ABEP engine intake optimization and provide insights into necessary steps for further refinement and implementation.

Keywords: *non-collisional flow, free molecular flow, ABEP, VLEO, CFD solver, Python, intake optimization*

Abstract

L'òrbita terrestre molt baixa (VLEO) ha sorgit com un camp prometedor per l'avenç en la tecnologia sateital, oferint avantatges significatius en l'observació de la Terra i les telecomunicacions. Per mantenir les operacions a aquestes altituds, els sistemes de propulsió elèctrica ABEP (de l'anglès Air Breathing Electric Propulsion), que utilitzen captadors d'aire per recollir i comprimir partícules atmosfèriques per a la propulsió, podrien resultar essencials. Aquest treball se centra en el desenvolupament d'un solucionador de flux no coisional dissenyat per a l'optimització de la forma del captador del motor ABEP de KREIOS. L'objectiu principal és desenvolupar una eina computacional basada en Python que pugui modelar el flux molecular lliure mitjançant la simulació de les trajectòries de les partícules de gas a través de diverses geometries. La metodologia implica el disseny de dos enfocaments computacionals, amb èmfasi en un enfocament més senzill per proporcionar resultats preliminars. Tot i que la integració del solucionador en un bucle complet d'optimització de forma i la implementació de millores d'eficiència computacional a gran escala queden fora de l'abast del projecte actual, aquest treball suposa una eina fonamental per al desenvolupament futur. Els resultats demostren la viabilitat de la modelització del flux no coisional per a l'optimització del captador del motor ABEP i proporcionen coneixements sobre els passos necessaris per a un major refinament i implementació.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Object

This project aims to develop a collisionless flow solver for the intake of KREIOS' Air Breathing Electric Propulsion (ABEP) engine. The work will focus in studying VLEO gas dynamics, and implementing a python code that can compute the parameters needed for shape optimization of the engine intake under such conditions. In order to do so, two computational approaches will be proposed, the simplest of which will be put into practice through by building a code capable of computing preliminary results. This prototype will serve as the foundation for the ultimate goal of implementing the solver into a shape optimization loop, which falls off this project's scope.

Gaining experience with Python programming, ray tracing libraries, and other essential tools for developing the non-collisional code is also an essential objective for the further refinement and scaling of the solver.

1.2 Scope

The scope of this thesis encompasses the development and preliminary validation of a collisionless flow solver specifically tailored for the intake optimization of KREIOS' Air Breathing Electric Propulsion (ABEP) engine. The project focuses on understanding the gas dynamics at Very Low Earth Orbit (VLEO) and creating a Python-based computational tool to facilitate the shape optimization of the engine intake.

In order to achieve the planted objectives, his thesis will:

- Study rarefied gas flow dynamics and determine for which range of altitudes can we assume

the flow to be collisionless

- Formulate two computational approaches capable of computing the parameters needed for shape optimization of the intake
- Develop a prototype solver that models free molecular flow by simulating the trajectories of gas particles through a mesh representing the intake geometry.
- Test the solver and obtain preliminary (non-representative) results on different geometries

However, this thesis will not:

- Fully integrate the developed solver into a comprehensive shape optimization loop for the ABEP intake. This integration is beyond the current project's scope and intended for future work.
- Address the full-scale computational efficiency improvements, such as parallel computing and GPU acceleration, although the potential for these enhancements will be discussed based on preliminary findings.

The outcome of this project will be a foundational tool that demonstrates the feasibility of collisionless flow modeling for ABEP engine intake optimization.

1.3 Requirements and limitations

Specifications or requirements:

- Free molecular flow must be studied to determine if the collisionless assumption is valid at the ranges where KREIOS' engine is expected to operate.
- The individual key functions of the code must be individually validated.
- Parameter flexibility is key. The user shall modify the composition, number of particles, particle velocity, etc. to simulate.
- Geometry (and mesh) flexibility: the code should work for any geometry that includes an inlet surface, some walls and an outlet surface. Ensuring so, we will be able to program the solver with a simple geometry and then implement it on a much more complex geometry.
- Initial and intermediate versions of the code have to be able to plot meshes, particle trajectories, reflections, etc. to facilitate debugging in the programming process.

- The designed computational approaches must be compatible with parallel computing for future implementation.

Basic limitations:

- Programming experience: the reduced previous knowledge in the field of CFD programming is set to be a limiting factor throughout the entire project.
- Little previous research in the field: the almost non-existent previous work done in the development of free molecular flow solvers will also be a limitation. It will be tough to define guidelines and determine which is the best approach to follow, given that it has not been done before.
- Computation time: without parallel computing and/or GPU computing, the execution of the solver on a single processor is likely to be slow, and a de-escalation of the real problem may be necessary (by reducing the amount of particles or limiting the possible reflections) in order to achieve results.
- Duration of the project: the time limitation of the Bsc Final Thesis is a constraint for achieving the broader goal of obtaining a fast and efficient solver that can be implemented into the optimization loop of the intake geometry.

1.4 Rationale

The rapid expansion of commercial activity in space over the past decade has highlighted the potential of Very Low Earth Orbit (VLEO) as a promising frontier for satellite deployment. VLEO, despite its challenges, offers a unique set of advantages that are increasingly attractive for various space applications. Historically, VLEO has been underutilized due to the significant atmospheric drag that makes maintaining orbit costly and inefficient with traditional propulsion technologies. However, recent advancements in novel propulsion systems, particularly Air-Breathing Electric Propulsion (ABEP), are poised to overcome these obstacles, making VLEO a viable and economically feasible option.

The allure of VLEO is multifaceted. Firstly, utilizing this orbit can substantially mitigate the problem of space debris. Satellites in VLEO can benefit from reduced collision risk due to the lower density of debris in these orbits, which simplifies both launch and operational phases. Additionally, VLEO's proximity to Earth enables significantly reduced communication latency, which is critical for real-time data transmission in telecommunication and Earth observation applications. Furthermore, Earth observation satellites operating in VLEO can achieve higher image resolutions due to their

closer vantage point, enhancing the quality and precision of data collected.

Market trends have shown a clear shift towards lowering the average altitude of commercial satellites, driven by the need for higher performance and more efficient use of payloads. This trend underscores the necessity for developing new renewable electric propulsion technologies capable of sustaining prolonged operations in VLEO. ABEP technology, in particular, represents a significant breakthrough. It leverages the sparse air present in the upper layers of the atmosphere, utilizing it as a propellant, thereby addressing the primary challenge of atmospheric drag. This innovative approach not only extends the operational life of satellites in VLEO but also reduces the dependency on onboard propellant reserves.

The need for developing a non-collisional solver is driven by the specific requirements and constraints of the ABEP engine's operating environment. While Direct Simulation Monte Carlo (DSMC) methods are a well-established and robust approach for simulating rarefied gas flows, they tend to be computationally intensive due to the necessity of simulating numerous particle collisions. In the very low-density conditions of Very Low Earth Orbit (VLEO), where the mean free path of gas molecules is much larger than the characteristic dimensions of the spacecraft intake, the frequency of collisions is significantly reduced. This makes DSMC methods less efficient for these specific applications, as the computational effort does not scale favorably with the sparse collision events.

Developing a non-collisional solver, on the other hand, leverages the hypothesis that in the VLEO environment, particle-particle interactions are minimal and the dominant interactions are between the gas particles and the surfaces of the spacecraft. This approach simplifies the computational model by focusing solely on particle-surface interactions, which is expected to reduce computational time and resources while maintaining accuracy. The non-collisional solver can thus provide faster simulations, allowing for more rapid iteration and optimization of intake designs. This increased efficiency is crucial for the iterative design processes in ABEP development, where multiple configurations need to be evaluated swiftly to identify the optimal design parameters.

Chapter 2

Background

2.1 VLEO

2.1.1 Atmospheric conditions

Very Low Earth Orbit (VLEO), generally defined as orbital altitudes below 450 km[1], presents unique challenges and opportunities for spacecraft propulsion and operation. To be more precise, the major potential of ABEP systems lies between 100km and 250km of altitude according to Schönherr, T. et al.[2], where the drag can still be compensated by ABEP engines while the gases are not yet too rarefied for these to be efficient. VLEO is an orbital range within the region of the atmosphere known as the thermosphere, which extends from 80km to 500km and gets its name from the drastic increase in temperature with altitude. The composition of the lower thermosphere is mostly molecular nitrogen (N_2) and oxygen (O_2) with trace of other gases such as helium (He) or hydrogen (H). At higher altitudes Ultraviolet (UV) radiation (wavelengths between 130 and 175 nm) are absorbed causing the dissociation of O_2 into atomic oxygen O, which then becomes the dominant gas [3]. The high reactivity of atomic oxygen poses significant challenges for spacecraft materials and propulsion systems, as it can cause erosion and oxidation [4]. The Sun's activity fluctuates over an 11-year solar cycle, heavily influencing the thermosphere and ionosphere's conditions. The former is contained within the latter and is composed of weakly ionized plasma that interacts dynamically with the neutral gases of the thermosphere. The ionosphere exists primarily due to the Extreme Ultraviolet (XUV) radiation from the Sun at wavelengths shorter than 103 nm [3]. During the cyclical periods of high solar activity, radiation from the Sun heats the Earth's upper atmosphere, causing it to expand and increasing atmospheric density at higher altitudes [5]. Conversely, during low solar activity, the atmosphere contracts, leading to lower densities at a given altitude. Intensity levels are indicated by the solar activity level, which accounts for the electromagnetic activity

from the sun that reaches the Earth's atmosphere. It is measured in Solar Flux Units (SFU) and it is tracked by the U.S. National Oceanic and Atmospheric Administration, which also presents predictions for the next solar cycle. Additionally, solar winds transfer energy to the Earth's magnetic field resulting in auroral processes, also affecting the ionosphere's conditions in the polar regions. Dynamic atmospheric processes that propagate upwards from the lower atmosphere also influence VLEO conditions. Theses include gravity waves, tides, planetary waves, pressure gradients and scattering processes. The combined effect of the forementioned factors result in the great variability existing in the thermosphere-ionosphere system.[3].

2.1.2 Atmospheric modeling

As seen, at such range of altitudes atmospheric conditions are highly variable, including density, temperature, conductivity and composition of the rarefied gases. Since the beginning of space travel, studies have been conducted to analyze and model the Earths atmosphere. With the advent of satellites and atmospheric probes, the empirical data obtained allowed for the development of modelling techniques of the atmospheric conditions with respect to temperature, total density, etc. The first standard to describe atmospheric properties as a function of altitude was the 1976 International Standard Atmosphere, based on studies that had started at the 1960s [6]. As space missions increased and atmospheric data became much more available, NASA Goddard SFC modeled the global temperature and composition of the residual atmosphere (partial densities of N_2, O_2, He, Ar, H) for altitudes between 120 and 150km. That was the MSIS model (Mass Spectrometer and Incoherent Scatter)[7]. Successive revisions and extensions of the model (MSIS-83, MSIS-86[4], MSISE-90, NRLMSISE-00) included more data from satellites and radars, expanding the altitude range of the model and including more species to the atmospheric composition.

Improved accuracy in thermospheric density prediction is provided by the Jacchia and Bowman model, JB2008, particularly in high solar activity times. It is especially helpful for missions in highly variable solar conditions since it makes use of solar and geomagnetic indices to predict variations in air density and composition [8]. Using information from the Space Surveillance Network, the High Accuracy Satellite Drag Model (HASDM) is a sophisticated model that offers real-time updates on air density. In VLEO, where atmospheric drag greatly affects satellite dynamics, this model is very helpful for tracking and forecasting satellite orbits [9].

Nonetheless, the NRLMSISE-00 model is frequently regarded as the most suitable for collecting data regarding the density, temperature, and composition of the atmosphere in VLEO for ABEP

system design[2]. This model incorporates data from multiple sources to account for both solar and geomagnetic activity, providing a compromise between accuracy and comprehensive data coverage. For the purpose of planning intakes and forecasting the performance of ABEP systems, its comprehensive outputs on atmospheric composition, including atomic oxygen and nitrogen levels, are very helpful [5].

2.2 Physics and governing equations

2.2.1 Dilute gas flow physics

Understanding the fundamental laws that describe rarefied gas flow behaviour is essential before delving into ABEP design. The rarified vacuum in VLEO features extremely low pressures and densities, which make viscous flows negligible or impossible[5]. Under such conditions, the traditional flow physics approach based on the continuum model is no longer applicable, as the particle-particle interactions are significantly reduced. Continuum assumes a regular distribution of mass with no empty space within the flow, allowing for macroscopic properties to be defined. This is no longer valid for insufficient molecular densities, and Navier-Stokes equations are no longer applicable. Instead, rarefied gas flow dynamics is described through the kinetic theory of gases. The state of these is detailed on a molecular level through statistical physics, as depicted by the Boltzmann equation. Such equation is also valid even if the gas gets denser and can be considered continuum. The diameter of gas particles of a rarefied gas d can be considered as the characteristic length for the process of collision between particles. In dilute gases, such value is much smaller than the average distance between particles, as depicted by the density parameter or volume fraction ϵ.

$$\varepsilon = n \cdot d^3 = \frac{N}{V} \cdot d^3 << 1 \tag{2.1}$$

where n is the number density of the gas (number of particles N divided by the total volume V) and d is the diameter of gas particles.

The mean free pass, mean collision time, collision frequency and the dimensionless Knudsen number are the main parameters used to determine the flow regime and the degree of rarefaction of gas particles.

The mean free path λ indicates the average distance a gas particle travels before colliding with another particle. The higher the number, the more dilute the gas will be.

$$\lambda = \frac{1}{n_\infty \sqrt{2} \pi d^2} \tag{2.2}$$

where n_∞ is the number density of the flow and d the average kinetic diameter of gas particles, as defined for the density parameter.

The collision frequency corresponds to the amount of particle-particle collisions per unit time. The lower the frequency, the more rarefied the gas will be.

$$\nu = n_\infty \pi d^2 g \tag{2.3}$$

where g is the relative velocity of particles $g = |v_1 - v_2|$.

The mean collision time indicates the time between two consecutive collisions. The higher it is, the more rarefied the gas will be.

$$t_c = \frac{\lambda}{\bar{v}} \tag{2.4}$$

where $\bar{v}$ is the average particle velocity, a function of the temperature T and the average particle mass m, $\bar{v} = \sqrt{\frac{8 \cdot k_b \cdot T}{\pi m}}$. k_b corresponds to the Boltzmann constant.

Lastly, the dimensionless Knudsen number indicates the predominance of collisions in a gas flow, making it the value used to define the flow regime of a rarefied gas. It is defined as:

$$Kn = \frac{\lambda}{L_c} \tag{2.5}$$

where λ is the above-mentioned mean free path and Lc is the characteristic length of the problem.

If there are a lot of particle collisions, the mean free path λ will be very low, resulting in small values for the Knudsen number. For $Kn < 0.01$ the flow is considered as a continuum. Furthermore, when the amount of collisions is reduced but not negligible the flow is defined as transitional, assigned to $Kn \sim 1$. Finally, for a $Kn > 10$ the amount of particle collisions can be considered nonexistent or negligible, defining the flow as collision-less, or free molecular flow (FMF).

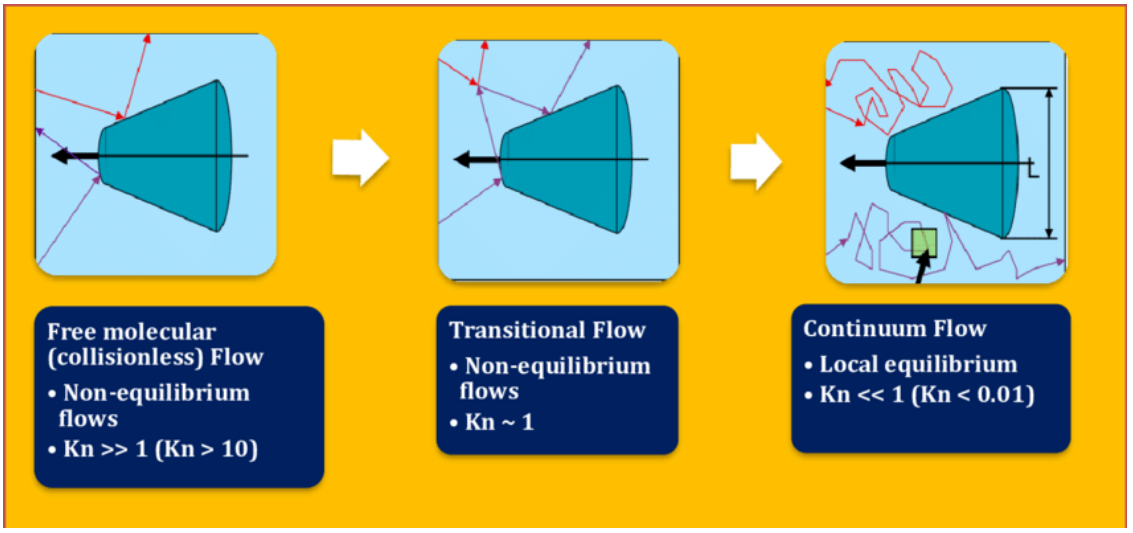

Figure 2.1: Flow regime depending on the Kn

Source: [10]

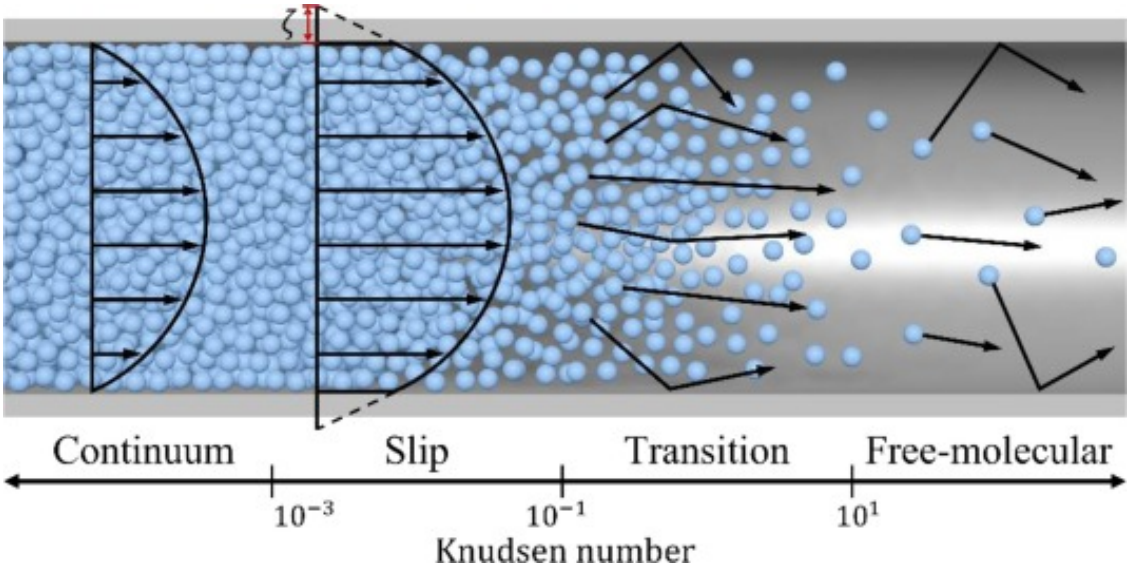

Figure 2.2: Flow regime depending on the Kn

Source:[11]

2.2.2 Gas flow regime at VLEO

With the objective of evaluating the flow regime at VLEO orbits, we will now calculate the Knudsen number Kn at different altitudes. The Knudsen number is a function of the mean free path of particles λ, which is a function of the average diameter of gas particles d and the number density of

the flow n_∞.

As seen, the most updated and validated atmospheric model is NRLMSIS-00[5]. Thus, the data used in this section has been computed with the instant run system for NRLMSIS-00 of NASA's Community Coordinated Modeling Center (CCMC)[12].

First, we will take the intake diameter as the characteristic length of the problem $L_c = 0.1m$.

Next, we need to find an approximate average diameter of the particles. To simplify the problem, we will only consider atomic oxygen (O), molecular oxygen (O_2) and molecular nitrogen (N_2) in the composition of atmospheric gas at VLEO. Their respective kinetic diameters are $d_O = 3.0\mathring{A}$, $d_{O_2} = 3.46\mathring{A}$ and $d_{N_2} = 3.64\mathring{A}$. At each altitude, we will compute the composition and calculate the average kinetic diameter:

$$d_{avg} = \%O \cdot d_O + \%N_2 \cdot d_{N_2} + \%O_2 \cdot d_{O_2} \tag{2.6}$$

Lastly, we have to obtain the number density of the flow, which is just a sum of the number densities of all existing species, obtained through the CCMC's instant run system for NRLMSIS-00.

The results obtained are shown in the following table, illustrating the flow regime at different altitudes.

Knudsen number at different altitudes							
Altitude (km)	**%O**	**%N2**	**%O2**	**d (m)**	**n (m^3)**	**λ(m)**	**Kn**
100	2,56	82,07	15,37	$3,48 \cdot 10^{-10}$	$8,20 \cdot 10^{18}$	$2,27 \cdot 10^{-1}$	**2,27**
110	6,90	81,67	11,43	$3,45 \cdot 10^{-10}$	$1,65 \cdot 10^{18}$	1,14	**11,4**
150	23,52	73,11	3,38	$3,36 \cdot 10^{-10}$	$4,30 \cdot 10^{16}$	46,48	**464,8**
200	40,97	57,42	1,61	$3,27 \cdot 10^{-10}$	$6,69 \cdot 10^{15}$	313,9	**3139**
250	57,50	41,61	0,89	$3,20 \cdot 10^{-10}$	$1,83 \cdot 10^{15}$	1149	**11490**

Table 2.1: Kn at different altitudes

As we can see, for the characteristic length of our problem, at around 110km of altitude the Knudsen number becomes greater then 10, making the collisionless assumption valid. The Kn then rapidly grows, indicating that the flow becomes more and more free-molecular.

2.2.3 Particle-surface interactions

When a gas particle suffers a collision with a solid surface, it can either be absorbed, scattered or experience a chemical reaction. However, the lack of specific data on surface properties (finish, absorbed gas layers, cleanliness, etc.) significantly complicates the modeling of such processes. Several approaches have been developed, the most widely used of which is the Maxwell model.

The Maxwell model is based on classical thermodynamics and simplifies the problem by considering only two types of interactions: specular and diffuse reflections[13]. In the following picture, both interactions are represented with the velocity vectors of particles in black and the resulting direction of the momentum exchange in grey.

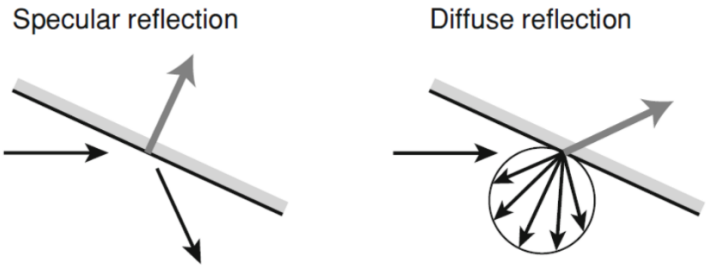

Figure 2.3: Particle-surface interactions in the Maxwell model

Source: [13]

In specular reflections, particles experience a fully elastic collision, with no energy exchange. The reflection angle is equal to the incident angle $\theta_r = \theta_i$, as in a mirror-like reflection, and the tangential component of particle velocity v_t remains constant, while the normal velocity v_n is inverted.

In diffuse reflection, upon colliding with a surface particles reach thermal equilibrium and are reflected following a half-range Maxwellian distribution corresponding to the wall temperature T_{wall} and particle velocity.

The Maxwell model establishes the tangential momentum or accommodation coefficient α_E, which indicates the fraction between diffuse reflection (α_E) and specular reflection ($1 - \alpha_E$). It is a parameter highly dependant on the surface material properties.

However, for free molecular flow and most surface materials, a full accommodation of diffuse reflections ($\alpha_E = 1$) can be assumed while obtaining good simulation results [14].

2.3 Air Breathing Electric Propulsion

Air Breathing Electric Propulsion (ABEP) engines represent a revolutionary approach to satellite propulsion in Very Low Earth Orbits (VLEO), typically below 450 km. Traditional propulsion methods rely on stored chemical propellants, which limit the operational lifespan and increase the launch mass and cost of satellites. ABEP engines, by contrast, utilize atmospheric particles as propellant, allowing for significantly extended missions without the need for onboard propellant storage.

ABEP systems are designed to intake atmospheric particles through a front-mounted inlet on the spacecraft. These particles, primarily composed of residual atmospheric gases at VLEO altitudes, are then ionized and accelerated using electric power, usually derived from solar arrays. This ionization and acceleration process generates thrust, compensating for atmospheric drag and maintaining the satellite's orbit.

One of the primary challenges in ABEP technology is the efficient collection and compression of atmospheric particles at extremely low densities. Atmospheric density at VLEO is highly variable, influenced by solar activity, geographic location, and time of day. Effective intake design must address these variations, ensuring a consistent and sufficient flow of particles to the propulsion system. Various intake designs have been proposed, including electrostatic and magnetic systems, which aim to enhance particle collection efficiency while minimizing aerodynamic drag [15, 16].

Electrostatic intake systems use electric fields to attract and compress atmospheric particles, providing a steady supply for ionization and thrust generation. Magnetic intake systems, on the other hand, employ magnetic fields to direct and compress ionized particles. Hybrid systems combining both electrostatic and magnetic elements have shown promise in achieving higher capture rates and better ionization efficiency [17, 18].

The potential applications of ABEP engines extend beyond orbital maintenance. By enabling long-duration missions in VLEO, ABEP technology can support high-resolution Earth observation, telecommunications, and scientific research missions. The enhanced resolution achievable at lower altitudes is invaluable for environmental monitoring, urban planning, and disaster response [13, 19].

Future research in ABEP technology is likely to focus on optimizing intake designs to improve efficiency and reduce drag further. Advances in materials science, electric field generation, and magnetic field manipulation will play crucial roles in developing next-generation ABEP systems. Continued innovation in this field promises to enhance the sustainability and capabilities of space missions, paving the way for more ambitious and cost-effective satellite operations.

Chapter 3

Methodology

3.1 Free molecular flow assumption

As seen in 2, the Knudsen number is significantly greater than 1 at the altitude ranges where ABEP engines would operate and thus the flow can be considered non-collisional. Neglecting particle-particle interactions notably simplifies the problem to solve, as solving the Boltzmann equation is no longer required. This has proven to be complex and highly time-consuming, which is why rarefied gas flow problems have been conventionally solved through the DSMC (Direct Simulation Monte Carlo) method. Direct simulation, presented by Bird, G.A. in 1963, focuses on solving the Boltzmann equation through statistical simulation. However, this method can still be notably time-consuming and computationally demanding due to the need to model both particle motion and collisions.

Instead, by assuming non-collisional behavior, we drastically simplify the problem. DSMC methods need to model both the motion and collisions of gas particles. This involves calculating collision probabilities, determining post-collision velocities, and updating particle positions iteratively, which is computationally demanding. In contrast, the non-collisional assumption eliminates the need for these complex calculations by assuming particles travel in straight lines and only interact with solid surfaces. This reduces the number of required computations, as each particle's trajectory can be calculated independently.

Furthermore, the free flow problem reduces the number of computational steps. DSMC requires iterative time-stepping to simulate gas flow evolution, involving moving particles, resolving collisions, and updating flow properties at each step. This iterative process can be slow, especially for large systems with many particles. The non-collisional assumption, however, allows for the direct

computation of particle paths and interactions with surfaces without the need for time-stepping. This significantly speeds up the simulation.

On top of that, the physics involved in the free flow problem is simplified. DSMC simulates detailed collision dynamics, including energy exchange and scattering angles, which require complex physics calculations. In the non-collisional approach, we only consider ballistic motion and reflections off surfaces. By avoiding the complexities of collision dynamics, we reduce the computational load and achieve faster simulations.

Additionally, the non-collisional assumption is inherently parallelizable. Since particles are treated independently, their paths can be computed simultaneously without needing synchronization. DSMC, while also parallelizable, requires synchronization at each time step to manage collisions and updates, adding computational overhead. By removing the need for synchronization, we further enhance the computational efficiency.

Finally, the memory and data management in the free flow problem are simpler. DSMC methods track detailed state information for each particle, including positions, velocities, and collision histories, leading to significant memory usage. The non-collisional assumption only requires tracking initial positions and velocities and their interactions with surfaces, reducing data management complexity and contributing to faster computation.

To illustrate, consider a simulation with one million particles. In DSMC, each time step involves moving particles, checking for collisions, updating velocities and positions, and synchronizing data across computational nodes if parallelized. In the non-collisional approach, each particle moves in a straight line until it hits a surface and then reflects according to simple rules, with no need for synchronization or iterative updates beyond initial and boundary interactions.

In short, the speedup of the non-collisional CFD solver over DSMC methods stems from eliminating collision handling, reducing iterative computations, simplifying physical interactions, enhancing parallelization, and minimizing data management requirements. This approach provides approximate results quickly, making it ideal as a low-fidelity solver for the first phases of shape optimization.

3.2 Geometries and meshes

3.2.1 Simple cubic mesh

To begin the developpement of the code, a much simpler geometry and mesh was needed, to ensure easier computation and easier visualization for detecting mistakes and debugging. The simple mesh designed was a combination of 3 *.stl* meshes that constituted a 10x10x10 cube. Each face of the cube was divided into two *trimesh* triangles, from vertex to vertex.

The following figure depicts the simple cubic mesh, plotted by Python library *Matplotlib*

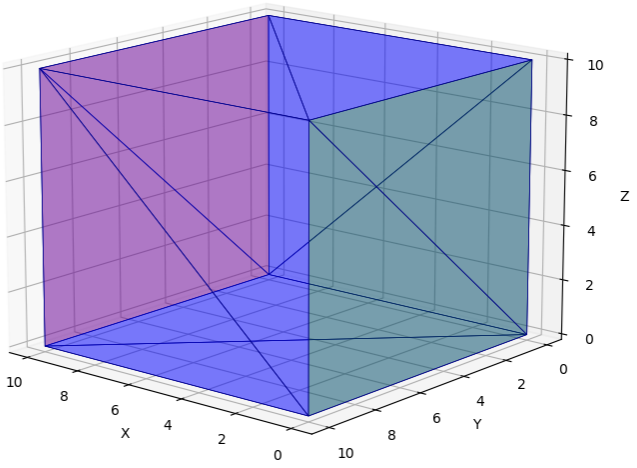

Figure 3.1: Simple cubic mesh

The green surface on the right corresponds to the inlet mesh, the red surfaces on the left corresponds to the outlet mesh and the blue surfaces on the sides of the cube correspond to the walls mesh.

3.2.2 Intake mesh

After developing the code on the simple mesh, it will then be run on the much more complex geometry of the engine intake. Such geometry consists on a circular intake with a honeycomb grid at the inlet and is based on Amer, M.'s Msc Thesis [20]. A non-structured mesh has been generated on the walls of the intake, and an outlet and inlet virtual surfaces (also meshed) have been included.

The green surface on the left corresponds to the inlet, while the red surface on the right corresponds to the small outlet; the rest are the blue walls of the intake.

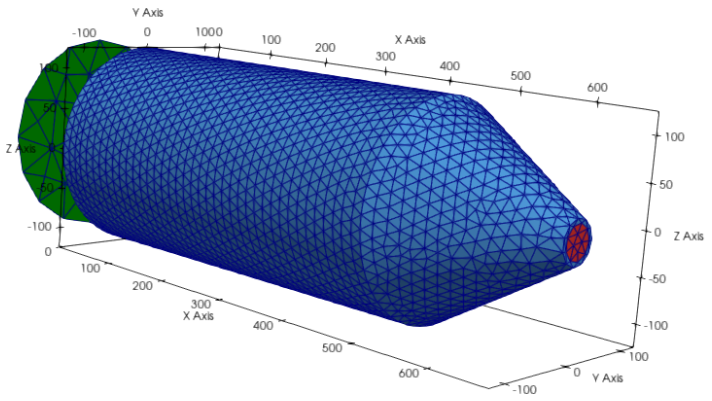

Figure 3.2: Intake mesh side view 1

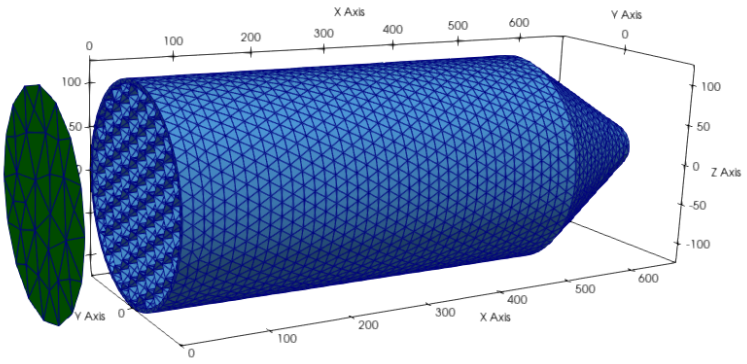

Figure 3.3: Intake mesh side view 2

Finally, the following image corresponds to the view of the intake mesh from the inlet towards the outlet.

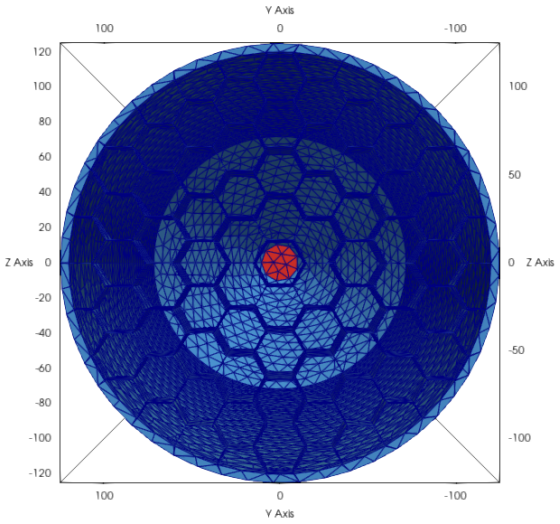

Figure 3.4: Intake mesh view from the inlet

3.3 Computational approaches

An optimal intake design is one that provides maximum inflow (mass flow at the outlet of the intake) while maintaining a minimum total drag. The thrust that the ABEP engine will be able to produce is a function of the amount of particles that the intake can collect, which we want to maximize. Additionally, the lower the drag, the greater the resultant propulsion force, once compensated by thrust. Essentially, this means that our solver must be able to compute the total drag force and the mass flow rate at the outlet. In this project, two approaches have been proposed to reach that goal.

3.3.1 Single wave approach

This first approach, that we will name the single wave approach, consists in 'launching' a set of particles from the inlet surface all at the same time, forming one single 'wave' of particles. This is the simplest approach to implement into a code, and thus the one we will program within this project, as we will see in future sections.

In free flow, we consider that particles do not interact with each other. By neglecting these particle-particle interactions, we no longer need to discretize the time domain or track each particle over time. Instead, we only care about the origin of the particle within the inlet, its trajectory and collisions with the intake walls and the position where it crosses the outlet. By computing the momentum exchanges of each particle-wall collision as well as the amount of particles crossing the outlet, we can calculate the intake mass flow and its aerodynamic drag.

However, these two magnitudes are time-dependant. On one hand, the mass flow rate $\dot{m}$ describes

the mass of particles crossing the outlet per unit time $\dot{m} = \frac{\Delta m}{\Delta t}$. Besides, the force exerted by gas flow on the walls F_f, the balance of which is the total drag, describes the change in momentum per unit time $F_f = \frac{\Delta p}{\Delta t}$.

It may seem as we need to somehow trace particles over time to obtain these two magnitudes, but in this approach the time domain is implicitly included in the amount of particles simulated n. Firstly, we determine the amount of time that we want to simulate the flow for Δt (physical time, not computational time). Now, taking the air density ρ_∞, satellite velocity v_∞, inlet area (considering it perpendicular to the flow) A_{inlet} and the mean mass of gas particles $m_{mean} = \frac{M_{mean}}{NA}$, we can easily calculate n:

$$n = \frac{\rho_\infty \cdot v_{\infty \cdot A_{inlet}}}{m_{mean}} \Delta t \tag{3.1}$$

This corresponds to the total amount of particles that would enter the inlet during Δt. In a collisional flow solver, we could not launch all particles at the same time, as there would occur multiple collisions and particle-particle interactions that would not happen in continuous flow. Regardless, in free molecular flow simulation, these interactions are ignored, and 'launching' all particles together has no effect on the problem output. Following the same rule, a greater Δt, which implies a greater n, will have no additional effect due to particle-particle interactions and the result will simply be more averaged.

One could argue that there is a misconception in this approach: the particles that enter the inlet at the last instant $t = \Delta t$ do not reach the outlet, but we are considering as if they were. That person would be right on the mark. These 'trapped particles' are in the intake domain when $t = \Delta t$, and they are included in n, which essentially means they will be simulated until they reach the outlet.

However, the flow velocity v_∞ is significantly greater than the length of the intake L_{intake}, therefore the time it takes for a particle to travel through the entire intake t_{travel} is very small. For a Δt significantly greater than t_{travel}, which we can express as $\frac{t_{travel}}{\Delta t} \ll 1$, the amount of 'trapped particles' is significantly smaller than the total amount of particles simulated n, and so will be their effect on the output of the problem ($\dot{m}$ and F_f).

In essence, with this approach there is a small miscalculation that can be neglected if Δt has a magnitude of 10^{-1} seconds or more: given a satellite velocity of magnitude $10^3 m/s$ and an intake length of $10^{-1} m$, the travel time will have a magnitude around $10^{-4} s$.

Finally, once all generated particles n have been simulated and left the domain through the inlet or the outlet, we can proceed to calculate the total drag force and mass flow rate. Having computed the momentum exchange $\Delta\vec{p}$ at each particle-surface collision, as well as the mass of each particle crossing the outlet Δm, we can calculate:

$$\vec{D} = \frac{\sum \Delta\vec{p}}{\Delta t} \tag{3.2}$$

$$\dot{m} = \frac{\sum \Delta m}{\Delta t} = \frac{n_{outlet} \cdot M_{mean}}{NA \cdot \Delta t} \tag{3.3}$$

where NA is the Avogadro constant, being $\frac{M_{mean}}{NA}$ the average mass of a gas particle.

3.3.2 Continuum approach

The continuum approach is based off the principle that each time a particle that exits the simulation domain (either through the inlet or the outlet), a new one is randomly generated at the inlet mesh. The main purpose of such idea is to maintain a constant level of particles being computed to keep the computer always occupied.

On the contrary, with the single wave approach, the computer may be still calculating reflections for the last few particles to exit through the inlet or outlet, while the rest are already out of the domain and fully computed. This can be inefficient computationally, as we are not using the total computing capacity all the time, but just at the beginning of the simulation (when we still have all particles in the domain). Instead, we propose to consistently leverage the full computing capacity by constantly have the same amount of particles in the domain.

Firstly, unlike in the single wave approach, the amount of particles simulated n_c is not analytically calculated to correspond to any physical magnitude. Instead, an optimal n_c has to be found to ensure the most efficient computation possible. Under the possibility of running the solver through the GPU, n_c would have to be calculated and optimized considering the GPU's specifications (number of Streaming Multiprocessors (SM), number of threads per SM, global memory of the processor...). Note that we express n_c to differentiate it from n, the amount of particles that enter the inlet in Δt, seen in the single wave approach3.1.

A set n_c number of particles is generated with a random origin within the inlet surfaces and an

initial direction. For each particle, the following data has to be stored: identifying number, position, index of the mesh element, direction and relative time ($t_i = 0$ for the particles initially generated).

Iterations of a loop are then started to trace particles and determine what we call 'events'. An event is an intersection between a particle trajectory and a mesh surface, and can be classified as a 'wall' event (reflection), an 'inlet' event or an 'outlet' event (departure of the particle from the domain).

For each event, the number of event, type of event, number of particle, position, particle direction (reflection) and time t_{event} will be stored. If an 'inlet' or 'outlet' event occur, that particle 'dies' and its identifying number will not be included in the next iteration of the function. Besides, a new particle will be generated randomly at the inlet with the data previously mentioned, and a relative initial time of $t_i = t_{event}$.

Now, the number of 'outlet' n_{outlet} and 'inlet' n_{inlet} events is computed in order to determine what we call the Trapped Particle Ratio (TPR), which is defined as:

$$TPR = \frac{n_c}{n_{outlet} + n_{inlet}} = \frac{n_c}{n_e} \tag{3.4}$$

where $n_e = n_{outlet} + n_{inlet}$ expresses the particles that have left the intake.

The TPR is an indicator of the number of particles that have entered the domain but have not left it yet. It is crucial for the TPR to be notably low so that we can compute with certain accuracy the mass flow rate. If most of the particles simulated are still trapped in the intake, the As the simulation progresses, the number of particles in the intake $n_c = ct.$ becomes smaller and smaller in comparison with the amount that have completed their path and departed through the inlet or outlet n_e, which keeps growing. When the TPR becomes smaller than a defined convergence criteria δ, the iterations end. At that point, we have

$$TPR \leq \delta \rightarrow n_e = \frac{n_c}{\delta} \tag{3.5}$$

We then have a total of particles simulated N that can be calculated as:

$$N = n_c + n_e - n_{elast} = n_c + \frac{n_c}{\delta} - n_{elast} = n_c \cdot (1 + \frac{1}{\delta}) - n_{elast} \tag{3.6}$$

where n_{elast} stands for the number of particles that exit the intake in the last iteration. They are subtracted from the expression as they will not be regenerated. If n_c is great enough and δ is small enough, n_{elast} will almost be negligible.

We can now think as if N particles were simulated following the single wave approach, but in a much more efficient way, computationally. Setting the relative times of events on an absolute time scale, we can compute the drag force and the mass flow rate the same way we did in the previous approach.

First, relating N with n, defined in the previous approach and described by equation 3.1, we can obtain the physical simulation time Δt it would be equivalent to:

$$\Delta t = \frac{N}{\frac{n}{\Delta t}} = \frac{N}{\frac{\rho_\infty \cdot v_\infty \cdot A_{inlet}}{m_{mean}}} = \frac{N \cdot m_{mean}}{\rho_\infty \cdot v_\infty \cdot A_{inlet}} \tag{3.7}$$

Finally, we just need to set all events on the same time scale by considering their relative time instead of their absolute time.

$$t_r = t_{event} - t_i \tag{3.8}$$

where t_r is the relative time of an event, t_{event} its absolute time, as stored, and t_i the initial time of the particle who suffered that event.

To compute momentum exchanges and mass flow rate, we initiate a loop to only consider the events with relative time is smaller than Δt. Finally, we calculate the drag force and mass flow rate in the same way as in the previous approach, as described in equations 3.2 and 3.3.

3.4 Particle velocity

When the rarefied gas particles enter the intake, their velocity v_{in} is a combination of the satellite's velocity and the typical velocity of their thermal motion.

The thermal velocity v_th, understood as the velocity for which a gas particle has the average translational kinetic energy, can be obtained as follows:

$$\frac{1}{2}mv_{th}^2 = \frac{3}{2}k_B T \rightarrow v_{th} = \sqrt{\frac{3k_B T}{m}} \tag{3.9}$$

where m is the average mass of a gas particle, k_B is the Boltzmann constant and T is the temperature of the gas.

The term on the left of the first expression corresponds to the definition of the kinetic energy of a particle, while the term on the right corresponds to the average translational kinetic energy, defined by the Maxwell-Boltzmann particle velocity distribution (applicable with accuracy for monoatomic and diatomic gases, such as the ones we can find in VLEO). The obtained thermal velocity is defined as the root mean square of the total velocity.

Finally, this thermal velocity component is applied on a random direction and added to the satellite's velocity, which is only applied on the X-direction (the longitudinal axis of the intake).

For each individual particle, we compute:

$$\vec{v}_{in} = v_{satellite}\vec{i} + v_{th}\vec{u} \tag{3.10}$$

where $\vec{u}$ is a random unit vector.

3.5 Sub-functions and validation

In this section we will take a look over some of the key functions of the code. We will discuss their purpose, their algorithmic methodology and, if possible, their validation. Most of these functions have been developed separately, independent from each other. They have then been progressively added together to constitute the final code, that we will discuss later on.

Random Point Generation in a Mesh

The function `random_point_in_mesh` generates a random point within a mesh by first selecting a random triangle based on its area and then generating a random point within that triangle. It is essential for the emission of particles from the inlet. If this function was somehow conditioned and did not provide random points, but instead had a tendency to one side of the intlet, it could result in more collisions on one side of the intake which would imply an imbalanced and asymmetric drag vector.

Algorithm 1 Random Point in Mesh

Input: Mesh
Output: Random Point, Selected Face Index
Calculate areas of all triangles in the mesh
Select a triangle based on the area-weighted probability
Extract vertices of the selected triangle
Generate a random point within the triangle using `random_point_in_triangle`
return Random Point, Selected Face Index

The sub-function `random_point_in_triangle` is used to generate a random point within a given triangle defined by its vertices. Once a triangle has been selected, with a probability proportional to its area, a random point within it is selected.

Algorithm 2 Random Point in Triangle

Input: Vertices $v1$, $v2$, $v3$
Output: Random Point
Generate two random numbers $s[0]$ and $s[1]$ uniformly distributed between 0 and 1
Compute $\sqrt{s[0]}$
Calculate $u = 1 - \sqrt{s[0]}$
Calculate $v = s[1] \cdot \sqrt{s[0]}$
return $u \cdot v1 + v \cdot v2 + (1 - u - v) \cdot v3$

In order to validate the function, it has been run to generate 5000 random points in a very simple mesh consisting of 5 triangles. One of which has a different area, prove that the function selects a triangle with an area-weighted probability. If it did not, we would see the small triangle with a significantly greater point density. With such a large number of points in a small mesh, the result observed can be an almost homogeneous or uniform distribution.

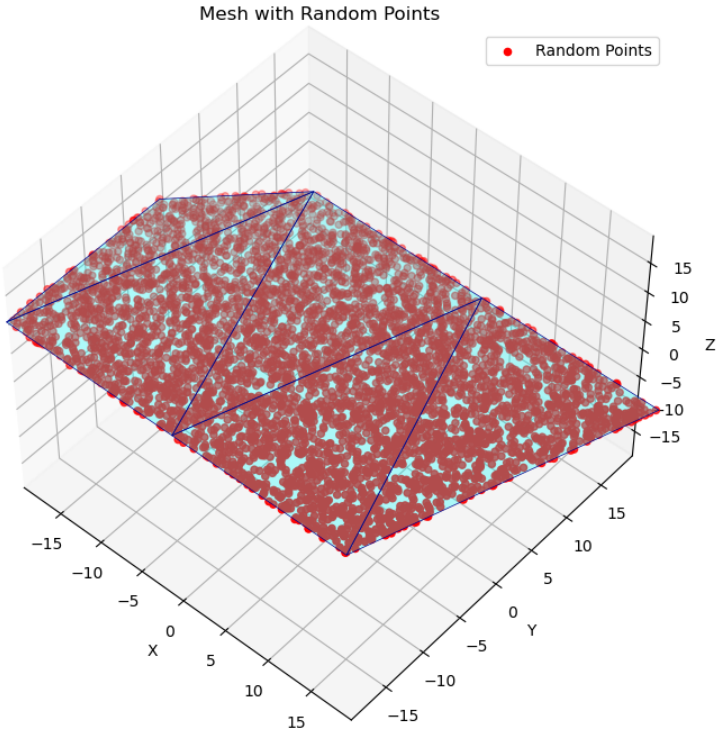

Figure 3.5: 5000 random points within a mesh

Reflection Calculation

The function `calculate_reflection` computes the resulting vector after a particle-surface collision with a completely diffuse reflection.

In the beginning of the development of this function, some particles crossed the surface and were deviated, as if they were a refracted ray. To prevent refraction, it was imposed for the reflecting vector to form an angle smaller than 90 degrees relative to the surface normal.

Algorithm 3 Calculate Reflection

Input: Direction, Normals
Output: Reflections
Generate random angles θ and ϕ for each normal
Compute reflection vectors based on θ and ϕ
Calculate dot product of reflections and normals
for each reflection **do**
    if dot product ≤ 0 **then**
        Reflect in the opposite direction
    end if
end for
return Reflections

The logic behind the validation of this function is similar to the previous one: 1000 rays particles with the same origin and direction have been launched to reflect on a surface at the same point. As picture 3.6 illustrates, the result is a semi-sphere of uniformly distributed reflection vectors at the same side of the surface as the incident ray.

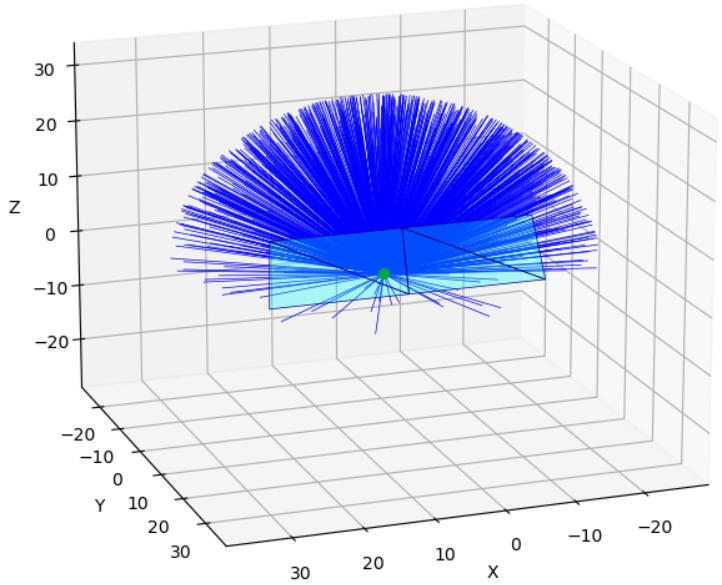

Figure 3.6: 1000 diffuse reflections

The previous rays, as well as the meshes and points were plotted using the `plot_mesh_and_rays` function, explained below.

Mesh and particle trajectory plotting function

In the early phases of the development of the code, it has been crucial to visually verify if meshing, particle generation, reflections, etc. were being computed correctly. The simplest way to do so is to plot the mesh and the particle trajectories, as well as their intersections. To do so, the `plot_mesh_and_rays` has been designed, based on Matplotlib library for Python. Here is a detailed explanation of the function's components:

- **Initialize the Plot:** A 3D plot is created using Matplotlib with a specified figure size.
- **Ray Extension:** An arbitrary length is set to extend the rays for better visualization.
- **Plot the Mesh:** The mesh is plotted using a `Poly3DCollection` with specified transparency (`alpha`) and edge colors.
- **Adjust Axis Limits:** The axis limits are set based on the coordinates of the mesh vertices to ensure that the entire mesh is visible within the plot.
- **Plot the Rays:** Each ray, which represents a particle trajectory, is plotted from its origin in the specified direction, extended by a factor for visualization purposes.

A ray in this context represents the trajectory of a particle, indicating the path it would follow from its origin, the inlet, until the next reflection point or the outlet.

Algorithm 4 Plot Mesh and Rays

Require: mesh, rays_origins, rays_directions, intersections (optional), reflections (optional)

```
fig ← plt.figure(figsize=(10, 8))
ax ← fig.add_subplot(111, projection='3d')
length ← 30
mesh_collection ← Poly3DCollection(mesh.vertices[mesh.faces], alpha=0.3, facecolor='cyan', linewidths=0.5, edgecolors='darkblue')
ax.add_collection3d(mesh_collection)
scale ← np.array([coord for vertex in mesh.vertices for coord in vertex]).flatten()
ax.auto_scale_xyz(scale, scale, scale)
for all (origin, direction) in (rays_origins, rays_directions) do
    ray_end ← origin + direction × length × 10
    ax.plot([origin[0], ray_end[0]], [origin[1], ray_end[1]], [origin[2], ray_end[2]], 'r-', linewidth=0.5)
end for
```

The validation of the function is immediate, as its execution is already visual. Picture 3.7 serves as

an example of the purpose of the `plot_mesh_and_rays` function. Here, 500 rays are generated at the inlet surface of the simple mesh and travel towards the outlet.

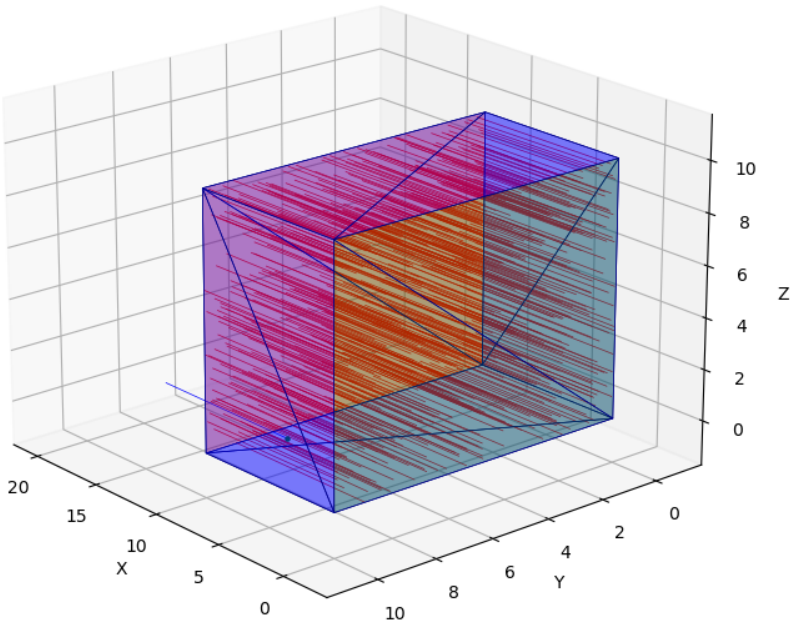

Figure 3.7: Mesh and rays plot

Counting Functions

The functions `count_reflections_per_triangle` and `count_crossings_per_triangle` count the number of rays reflecting on each triangle of the "walls" mesh and crossing through each triangle of the "outlet" mesh, respectively.

The program considers the particle-surface collision is fully elastic, and the reflection counter function applies a change in momentum equivalent to the mass of the particle and its velocity in the opposite of the vector between the incident vector and the reflected vector.

In the case of the crossings counter function, the program simply computes the number of particles

travelling through the outlet and multiplies it by their mass.

Algorithm 5 Count Reflections Per Triangle

```
Input: Mesh, Ray Origins, Ray Directions
Output: Triangle Momentum
Initialize triangle_momentum to zeros
for each ray do
    Perform ray-mesh intersection
    if intersection found then
        Calculate momentum exchange
        Update triangle_momentum for intersected triangle
    end if
end for
return triangle_momentum
```

Algorithm 6 Count Crossings Per Triangle

```
Input: Mesh, Ray Origins, Ray Directions
Output: Triangle Mass Flow
Initialize triangle_mass_flow to zeros
for each ray do
    Perform ray-mesh intersection
    if intersection found then
        Update triangle_mass_flow for intersected triangle
    end if
end for
return triangle_mass_flow
```

VTK Generator Function

The function `save_to_vtk` writes the mesh data and associated calculated values (such as momentum exchange, mass flow, and origins count) into VTK format files. VTK files are widely used for visualization in tools like ParaView.

- For each mesh (walls, inlet, outlet), the function extracts the mesh vertices and faces.
- Cell data includes the normals of the faces and additional metrics based on the mesh type:
  - **walls**: Momentum exchange per triangle.
  - **outlet**: Mass flow per triangle.
  - **inlet**: Number of origins per triangle.
- Point data includes vertex normals if available in the mesh.
- The function uses the `meshio.write_points_cells` method to write the data into VTK files named based on the provided `filename` and mesh name.

This approach enables detailed visualization and analysis of the flow dynamics within the mesh using ParaView or similar software.

Algorithm 7 Save to VTK

1: **Input:** Filename, Meshes, Momentum Exchange, Mass Flow, Origins Per Triangle
2: **for** each mesh **do**
3: Extract points and cells
4: Create cell data for normals and relevant metric (momentum exchange, mass flow, origins count)
5: Write data to VTK file
6: **end for**

3.6 Final code

The previously explained functions are added together and put into action in this final code, which implements the first approach, or the single wave approach. Particles are generated at the inlet mesh, travel through the geometry in the direction computed in ref, and may reflect off the walls or exit through the outlet mesh. The code simulates free molecular flow through ray tracing methods. By assuming collisionless flow, gas particle dynamics reduces to linear trajectories from surface to surface. Thus, each particle trajectory can be considered a "ray", which reflects off the geometry's walls.

The core function of the solver is `trace_rays`, which calculates all reflections and particle trajectories until they reach the outlet. At each iteration, it computes the intersection between all particles that are still in the domain and the intake's meshes. If the intersection occurs with the 'walls' mesh, the `calculate_reflection` function is applied, as well as the `count_reflections_per_triangle`, to compute the momentum exchange. In the contrary, if the intersection occurs with the inlet or outlet surfaces, the particle will not be included in next iteration (as it has exited the intake's domain), and in the case of the outlet the `count_crossings_per_triangle` is applied to compute the mass flow.

As mentioned earlier, the main flaw of this approach is that the program has to iteratively compute reflections until every single simulated particle has exited the domain (through the inlet or the outlet). This is can be computationally inefficient, given that in the end of the simulation, when very little particles are still confined reflecting, the program is still occupied. This is especially true if computing with the GPU, where one can think that there are a set number of computational slots. Each of these is assigned to a particle and traces its trajectory and reflections until they exit the domain. With this approach, if there is one single particle left reflecting, only one 'slot' will be in use while the rest are not in use.

The following picture is an example of a ray trajectory, as computed by the `trace_rays` function. An initial random origin is generated at the inlet, as well as an initial direction vector. The trajectory of the particle from the inlet until impacting on the first wall is plotted in red for visualization purposes. Afterwards, it reflects multiple times at the different walls of the cubic mesh until reaching the outlet.

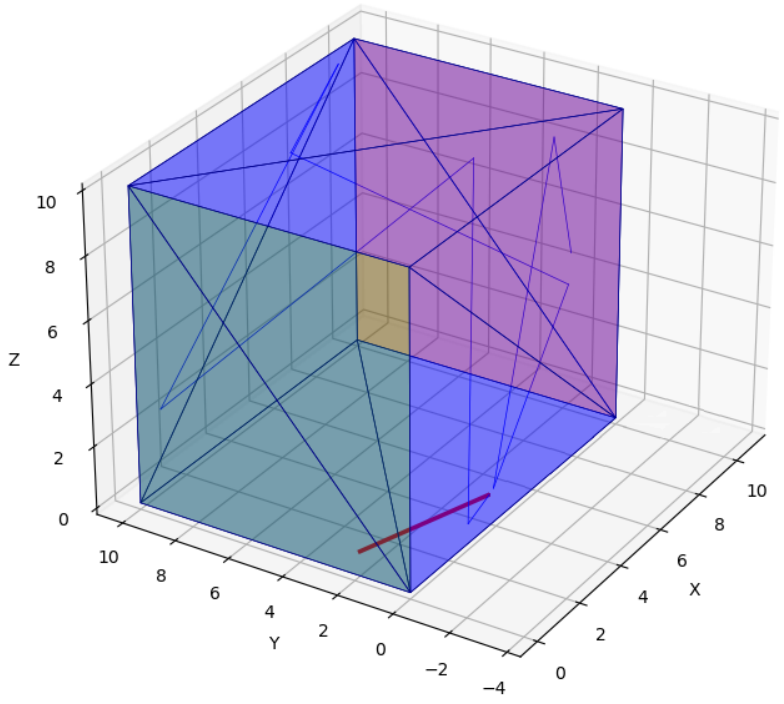

Figure 3.8: Complete trajectory of a particle from the inlet to the outlet, as computed by the `trace_rays` function

The algorithm of the `trace_rays` function is detailed below:

Algorithm 8 trace_rays

Require: Meshes, Ray Origins, Ray Directions, Max Reflections
Ensure: All Origins, All Directions, Reflections, Intersection Points, Intersection Indices

Initialize empty lists: all_origins, all_directions, reflections, intersection_points, intersection_indices
Set current_origins to ray_origins and current_directions to ray_directions
Initialize empty arrays: old_origins, old_directions
for $i = 1$ to max_reflections **do**
    Ensure current_origins and current_directions are 2D arrays
    Find intersections of rays with the mesh walls
    if no intersections found **then**
        break
    end if
    Get normals of intersected faces
    Calculate new reflections based on the current directions and normals
    Append locations to intersection_points and ray indices to intersection_indices
    if old_origins is not empty **then**
        Update current_origins and current_directions by removing previously processed points
    else
        Set current_origins to locations and current_directions to new reflections
    end if
    if current_origins is empty **then**
        break
    end if
    Append current_origins and current_directions to all_origins and all_directions
    Append new reflections to reflections
    Update old_origins and old_directions
end for
Return all_origins, all_directions, reflections, intersection_points, intersection_indices

3.6.1 Detailed Steps

The following is a description of the final code's structure:

1. **Initialization:** Initialize empty lists to store origins, directions, reflections, intersection points, and indices.

2. **Load the Meshes:** Load the `.msh` file using the `GMSH_reader_surftri` function, and separate faces based on `zoneid` to create trimesh objects for the walls, inlet, and outlet meshes.

3. **Generate Random Origins:** Generate random origins for the rays within the inlet mesh using the `random_point_in_mesh` function. Store these origins and their indices.

4. **Compute Particle Velocity:** Calculate the satellite velocity and the speed of gas particles using the ideal gas constant.

5. **Generate Random Directions:** Generate random directions for the particle speed vector

and calculate the ray directions by combining satellite velocity and particle speed. Normalize the ray directions.

6. **Ray Tracing Loop:**
   - For a specified number of reflections, find intersections of rays with the mesh walls.
   - If no intersections are found, exit the loop.
   - Calculate new reflection directions based on face normals at the intersection points.
   - Update current origins and directions by removing previously processed points.
   - Append results to the corresponding lists.
7. **Momentum Exchange Calculation:** Count the number of rays reflecting on each triangle of the walls mesh and calculate the momentum exchange for each triangle.
8. **Mass Flow Calculation:** Count the number of rays crossing through each triangle of the outlet mesh and calculate the mass flow for each triangle.
9. **Origins Count:** Count the number of origins generated at each triangle of the inlet mesh.
10. **Save Results:** Save the calculated data (momentum exchange, mass flow, origins count) to VTK files for visualization in ParaView.
11. **Return Results:** Return the collected data including origins, directions, reflections, intersection points, and indices.

Chapter 4

Results

4.1 Simple mesh simulation

The previously described code has been run with the simple cubic mesh and generating 100.000 particles at the inlet. Despite that number being very significantly lower than the total amount of particles we would simulate at VLEO, the aim of running the code is demonstrating the correct functioning of the program, verifying that it is capable of computing results.

4.1.1 Origins distribution at the inlet

The following picture is a visualization of the distribution of particle origins at the inlet surface of the cubic mesh through Paraview. The completely uniform placement of origins further validates the `random_point_in_mesh` function.

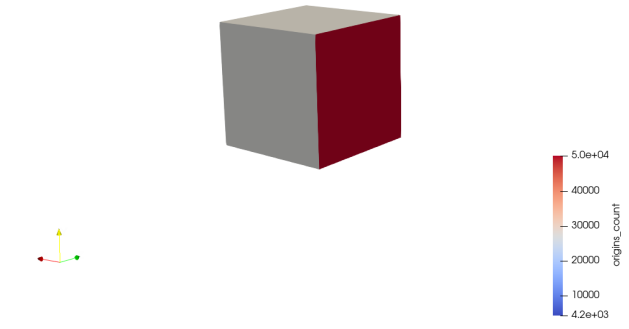

Figure 4.1: Origins distribution for the simple mesh simulation

4.1.2 Mass flow rate distribution at the outlet

4.2 displays the also uniform distribution of mass flow between the two triangles that form the outlet surface mesh.

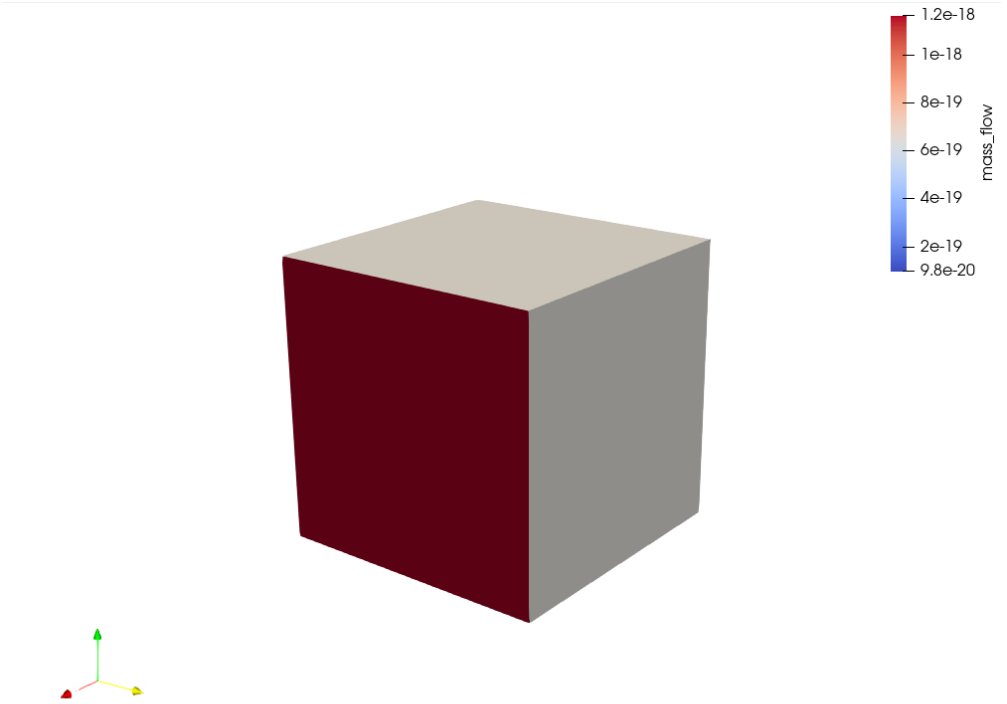

Figure 4.2: Mass flow rate distribution for the simple mesh simulation

4.1.3 Momentum exchange distribution at the walls

The image below represents the momentum exchange distribution at the walls of the mesh. As seen in the scale legend, the variation in Δp between the 8 triangles that constitute the walls mesh is very small. Divided by Δt and the area of each element $A_{element}$, we would obtain the pressure distribution.

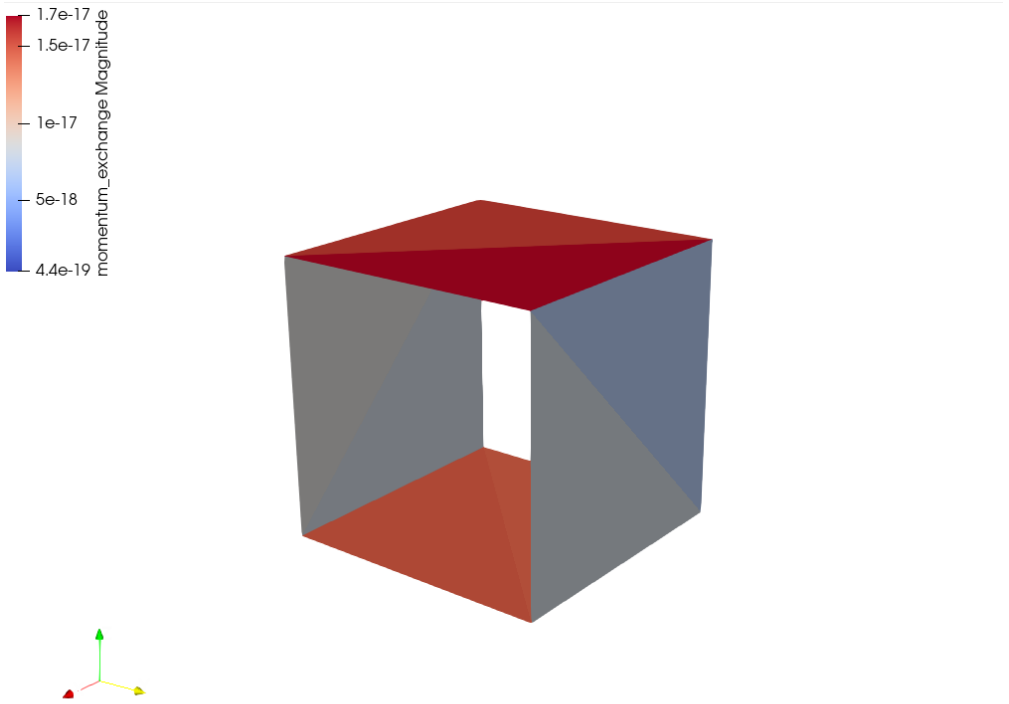

Figure 4.3: Momentum exchange distribution for the simple mesh simulation

4.1.4 Mass flow rate and drag force

The program also prints the drag force vector ($\vec{D}$) and the total mass flow rate at the outlet ($\dot{m}_{total}$), which are shown below:

$$\vec{D} = \frac{\sum \Delta \vec{p}}{\Delta t} = [1.49938979 \cdot 10^{-17} 1.95669413 \cdot 10^{-18} - 3.07187815 \cdot 10^{-18}]$$

$$\dot{m} = 2.321313852758445 \cdot 10^{-18}$$

As explained, the number of particles simulated n is very significantly smaller than the number of particles that the intake would collect at operation. Thus, the resulting mass flow rate and drag force's magnitudes are extremely small. However, the obtention of these results serves the purpose of demonstrating the non-collisional solver as a concept.

4.2 Intake mesh simulation

Now, the solver is run on a much more complex geometry and mesh: the intake. As seen in 3, the intake has a funnel shape, which complicates the exit of particles through the outlet surface,

specially considering reflections are diffuse. To reduce computation time and prevent a long loop of particles reflecting in the interior of the intake, which would take significant computational time, a reflection limitation has been set on the `trace_rays` function. Again, the purpose of running the code is to demonstrate loop closure and to prove that the code can obtain results. However, in order to obtain real-scale results (with a high amount of particles and without limiting the number of reflections), a more efficient approach is needed (such as the continuum approach) as well as better computation capacities (through parallel computing and/or making use of the GPU).

The following results are obtained after simulating 100 particles limited to 2 reflections.

4.2.1 Origins distribution at the inlet

The next picture illustrates the distribution of particles generated at the intake's inlet surface. As it has been simulated with very little particles, the distribution is not as homogeneous as in the previous section, with the simple mesh.

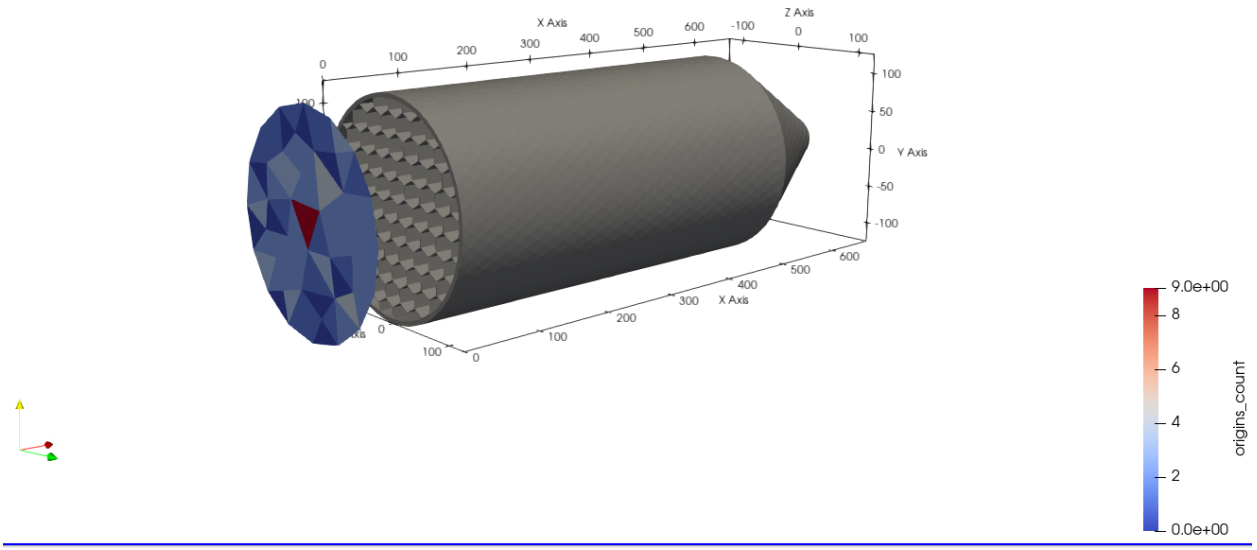

Figure 4.4: Origins distribution for the intake simulation

4.2.2 Mass flow rate distribution at the outlet

4.5 represents the mass flow rate for each triangle of the outlet mesh. It must be noted that the amount of particles crossing the outlet is significantly lower than the ones generated at the inlet, for this simulation, as the reflection number has been strictly limited and a big percentage of

the generated particles have not reached the outlet when the `trace_rays` function terminates its iterations.

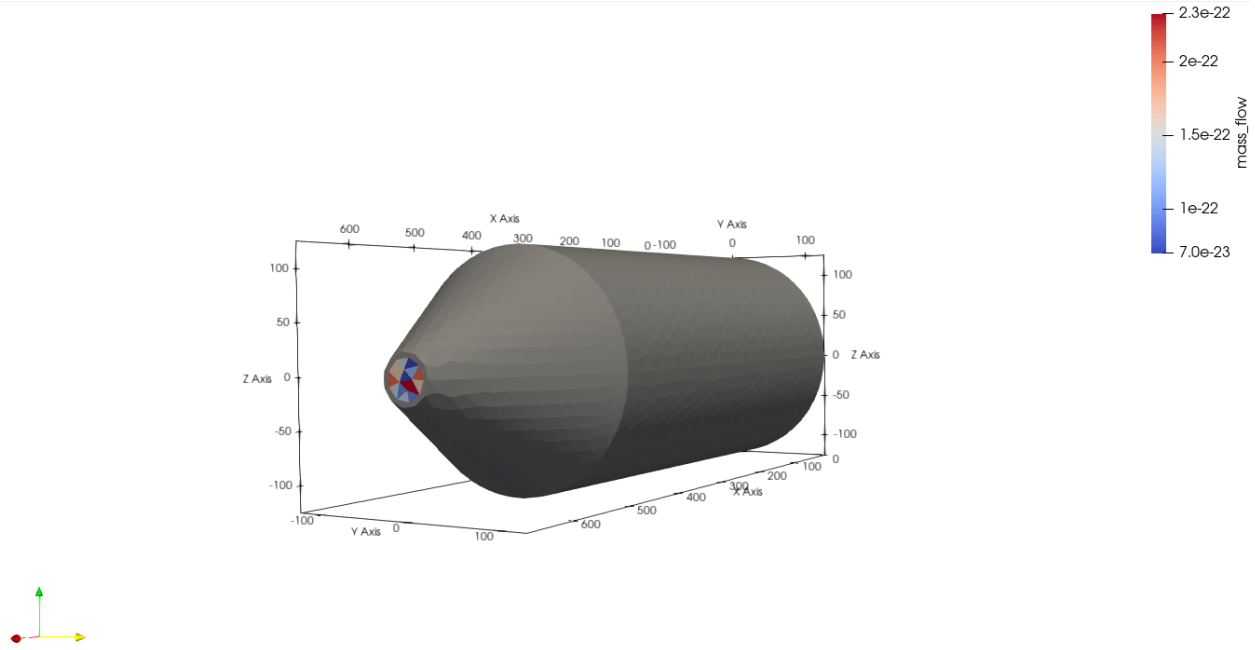

Figure 4.5: Mass flow rate distribution for the intake simulation

4.2.3 Momentum exchange distribution at the walls

The following image depicts the distribution of change in momentum due to gas flow at the interior walls of the intake. Its funnel shape causes most of the particle-wall collisions to occur on the final cone of the intake, contributing more significantly to the drag in the X-direction.

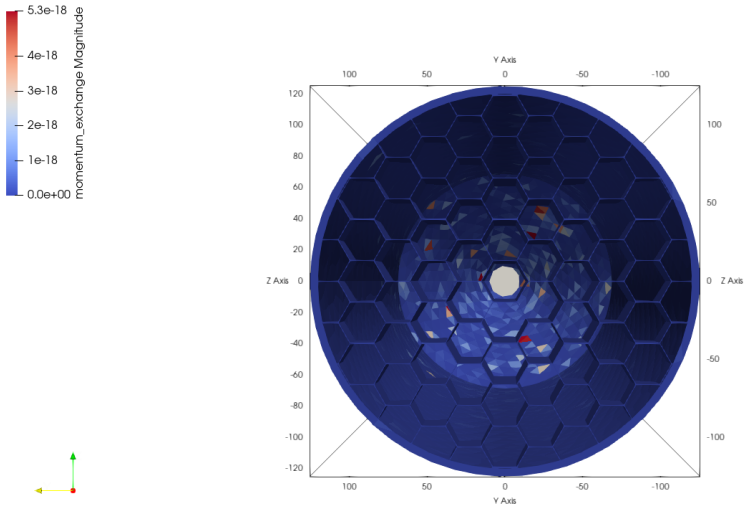

Figure 4.6: Momentum exchange distribution for the intake simulation

4.2.4 Mass flow rate and drag force

Finally, the solver computes the drag force vector ($\vec{D}$) and the total mass flow rate at the outlet ($\dot{m}$):

$$\vec{D} = \frac{\sum \Delta \vec{p}}{\Delta t} = [1.40114723 \cdot 10^{-15} 1.87800113 \cdot 10^{-17} - 1.97950002 \cdot 10^{-17}]$$

$$\dot{m} = \frac{\sum \Delta m}{\Delta t} = \frac{n_{outlet} \cdot M_{mean}}{NA \cdot \Delta t} 2.231764309824 \cdot 10^{-21}$$

As mentioned, these values are not representative, but rather a proof of concept of the code run with the actual intake geometry. Scaled to a notable amount of particles and without limiting reflections, the code could output real approximate values mass flow rate and drag, useful for the optimization of the intake.

Chapter 5

Budget summary

The only cost of this project is the workload of the engineer developing it. For an approximate total work time of 300h and taking the average hourly-salary of an entry-level engineer in 2024 in Spain (16€/h)[21], we have a budget summary shown in the table below.

Budget breakdown			
Details	Hourly cost (€/h)	Amount (€)	Total (€)
Engineer salary	16	300	4800

Table 5.1: Budget breakdown

Chapter 6

Analysis and assessment of environmental and social implications

The development of Air-Breathing Electric Propulsion (ABEP) engines holds significant environmental and social implications. Designed for very low Earth orbits (VLEO), ABEP engines can transform satellite propulsion, enhancing sustainable space operations and societal applications. The major advantage of ABEP technology is its facilitation of satellite operations in VLEO, defined as orbits below 450 km. These orbits offer improved resolution for Earth observation and telecommunications satellites, achieving a resolution 16 times greater than higher orbits. This boost allows for more detailed data collection, valuable for environmental monitoring, urban planning, and disaster management.

High-resolution satellite imagery enhances tracking of environmental changes like deforestation, glacier melting, and ocean pollution. This supports conservation strategies and informs policy decisions to reduce environmental degradation. Urban planners can use this data to optimize infrastructure, monitor air quality, and manage resources more effectively.

ABEP engines represent a leap in environmental sustainability. Unlike traditional propulsion systems that rely on chemical propellants, ABEP technology uses air and solar power to produce thrust. This eliminates the need for fuel, reducing the environmental impact associated with chemical propellants. ABEP engines can continuously generate thrust by taking in atmospheric particles, making them a renewable and environmentally friendly option. ABEP technology also addresses the critical issue of space debris. Traditional propulsion systems contribute to space debris by leaving behind components in higher orbits. In contrast, ABEP engines operate in underutilized VLEO orbits, avoiding the congested regions where most space debris accumulates. This reduces the risk

of collisions with existing space junk and contributes to a cleaner orbital environment.

From a social perspective, ABEP technology has far-reaching implications. The enhanced resolution provided by VLEO satellites benefits various sectors. In agriculture, high-resolution imagery helps monitor crop health, optimize irrigation, and manage land use. Detailed satellite data aids in urban planning, infrastructure development, and traffic management, facilitating better planning and development decisions.

ABEP technology also enhances disaster response capabilities. Near real-time environmental data provides critical information for emergency responders during natural disasters such as floods, hurricanes, and wildfires. High-resolution satellite imagery can identify affected areas, assess damage, and coordinate rescue and relief efforts efficiently, improving response times, saving lives, and reducing economic losses.

In conclusion, ABEP engine technology offers significant environmental and social benefits. By enabling access to VLEO, ABEP engines enhance satellite capabilities, providing higher resolution data for multiple applications. By avoiding space debris production and utilizing renewable resources, ABEP engines promote a sustainable approach to space exploration. The societal benefits, such as improved agricultural practices and enhanced disaster response, highlight ABEP technology's potential to positively impact various aspects of life on Earth. As the space industry progresses, eco-friendly and socially responsible technologies like ABEP engines will be crucial in shaping a sustainable future.

Chapter 7

Conclusions

Achievement

This thesis presents the development of a free molecular flow solver specifically designed for optimizing the intake of Air-Breathing Electric Propulsion (ABEP) systems. The primary objective of developing a Python code capable of simulating molecular flow in very low Earth orbit (VLEO) conditions has been accomplished. Moreover, invaluable experience in the field of CFD programming and free flow modeling has been gained.

First, a detailed literature review of VLEO conditions and their modeling has been done. Understanding the physics behind dilute gas flows like the ones we want to simulate has been crucial for the latter implementation of the solver. A specific study and calculation of the flow regime at VLEO has demonstrated the fasibility of assuming the flow as non-collisional, they key premise of this thesis. Lastly, a general review of ABEP technology has given context on the state of the art and basic principles of what we are achieving to optimize.

Furthermore, an analysis of how collisionless flow can be significantly easier and faster to simulate has been made, in order to understand the need and the advantages of the solver we aimed to develop. Subsequently, two different computational approaches to the solver have been formulated with detail, understanding their flaws and advantages. The simplest of them has been implemented as a Python code, capable of simulating the flow of rarefied gas particles through any meshed geometry, obtaining the mass flow rate and the total drag. In the process of doing so, intermediate functions with simpler objectives have been programmed and validated.

Finally, the developed solver's execution has been demonstrated by simulating flow on a simple cubic geometry and on a prototype of KREIOS' intake. Preliminary non-representative results have

been obtained for both, including mass flow rate and momentum exchange distributions, visualized through Paraview.

Overall, this thesis has successfully developed and validated a novel non-collisional computational fluid dynamics solver, potentially useful for the optimization of ABEP intake systems. The work has provided a comprehensive framework for simulating free molecular flow in VLEO conditions, demonstrating the feasibility of the proposed methods.

Future Work

This thesis has taken place within the broader goal of optimizing the intake of KREIOS ABEP engine, as mentioned. The future work proposed to further help achieve that goal consists in the improvement of the developed flow solver to achieve real-scale fast results, so that it can be added to an optimization loop.

It is suggested to implement the continuum approach as a Python code. This includes deeper delving into the physics behind the approach, the possible limitations and solutions. As we have seen, it is a much more efficient approach capable of supporting parallel computing. Thus, the proposal is to compute gas particles individually in multi-core GPU processors (through the exportation of NumPy arrays through CuPy, for example). Additionally, accurate atmospheric models should be implemented as inputs for the solver (density, temperature, composition, etc.), as well as particle-surface interaction models, guaranteeing more accurate results. Finally, the increase in density at the engine intake should be assessed, as it would result in a higher number of particle-particle collisions. This would include the simulation with conventional DSMC methods to determine the flow regime at the interior of the intake.

Bibliography

1. BECEDAS, J. et al. Aerodynamic Technologies for Earth Observation Missions in Very Low Earth Orbit. *16th Reinventing Space Conference (RISpace)*. 2018.
2. SCHÖNHERR, T. et al. Analysis of Atmosphere-Breathing Electric Propulsion. *33rd International Electric Propulsion Conference*. 2013.
3. SPEIGHT, J.G. Environmental Organic Chemistry for Engineers. 2017.
4. HEDIN, A. E. MSIS-86 thermospheric model. *Journal of Geophysical Research: Space Physics*. 1987, vol. 92, no. A5, pp. 4649–4662.
5. PICONE, J. M. et al. NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *Journal of Geophysical Research: Space Physics*. 2002, vol. 107, no. A12.
6. JACCHIA, L.G. *Static diffusion models of the upper atmosphere with empirical temperature profiles*. Vol. 8. Smithsonian Contributions to Astrophysics, 1965.
7. HEDIN, A.E. et al. *A global thermospheric model based on mass spectrometer and incoherent scatter data MSIS, 1. N2 density and temperature*. Vol. 82. Journal of Geophysical Research: Space Physics, 1977. No. 16.
8. BOWMAN, B. R. et al. A new empirical thermospheric density model JB2008 using new solar and geomagnetic indices. *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008.
9. STORZ, M. F. et al. High accuracy satellite drag model (HASDM). *Advances in Space Research*. 2005, vol. 36, no. 12, pp. 2497–2505.
10. SADREHAGHIGHI, I. Lattice Boltzmann Method (LBM). *CFD Open Series*. 2022.
11. LEE J. Kim, J.H. Statistical assessment of tangential momentum accommodation coefficient using internal flow rate model based on rarefied gas conditions. *Results in Physics*. 2022, vol. 43.
12. *CCMC Instant Run system - NRLMSIS-00* [`https://kauai.ccmc.gsfc.nasa.gov/instantrun/nrlmsis/`]. [N.d.]. Accessed: 06/2024.
13. DOORNBOS, E. *Thermospheric Density and Wind Determination from Satellite Dynamics*. Springer Berlin Heidelberg, 2012.

14. BIRD, G. A. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Clarendon Press, Oxford, 1994.
15. ROMANO, F. et al. RF helicon-based inductive plasma thruster (IPT) design for an atmosphere-breathing electric propulsion system (ABEP). *Acta Astronautica*. 2020, vol. 176, pp. 476–483.
16. PARODI, P. et al. Study of a collector-intake system for VLEO air-breathing platforms. In: *International Conference on Flight vehicles, Aerothermodynamics and Re-entry Missions and Engineering (FAR)*. 2019.
17. HISAMOTO, Y. et al. Design of air intake for air breathing ion engine. In: *63rd International Astronautical Congress*. 2012.
18. CARA, D. D. et al. RAM electric propulsion for low earth orbit operation: an ESA study. In: *30th International Electric Propulsion Conference*. Florence, Italy, 2007.
19. HOHMAN, K. *Atmospheric breathing electric thruster for planetary exploration*. 2012. Tech. rep. Busek Co. Inc.
20. AMER, M. *Optimisation of proposed intake geometries for ABEP systems through DSMC CFD analysis*. 2022. MA thesis. Cranfield University.
21. Engineer Average Salary in Spain 2024 - The Complete Guide. 2024. Available also from: `http://www.salaryexplorer.com/salary-survey.php?loc=198&loctype=1&job=9&jobtype=1`. Accessed: 2024-06-13.

Appendix A

Codes and meshes

The different Python codes and meshes used in this project have been uploaded to a GitHub repository. These include programs used to develop individual functions, programs to visualize or validate such functions, intermediate programs that are result of the process of combining functions, etc. A ReadMe file has been included to explain the purpose of each mesh or program in the repository.