Research and Design Document
Flight Simulator

How does flight physics work and how does one implement it in a game.

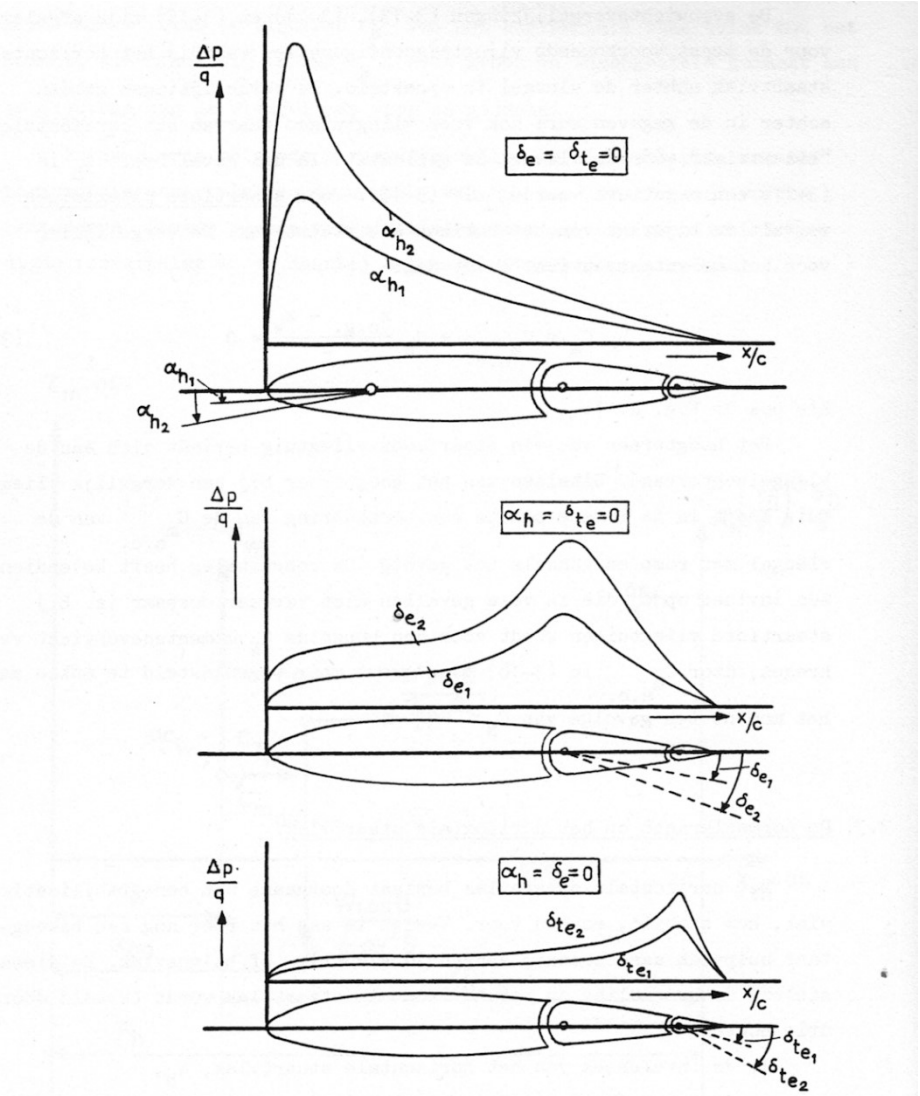

Fig. 3.5: Reduction of the pressure distribution over the stabiliser chord due to the change in subsequently resp. α_h, δ_e en δ_{te}.

Table of Contents

Introduction

Why am I doing this?

For my course Personal Branding & Portfolio I have decided to try and create a flight simulator with real-time realistic physics.

I have in the past 2 years really come to like a game called WarThunder, where you can fly, drive and control multiple forms of vehicles used in a military setting, the vehicles ranging from the biplanes of the inter-bellum to the iconic M1 Abrams MBT from the cold war times.

The game consists mainly of 3 different modes: Arcade, Realistic and Simulator. The mode that I currently play the most frequent is Simulator mode. This is due to the way that you have the realism of the mechanics; flat spins, wing stall, the need to trim and the overall scale, while still maintaining the casual playstyle due to not having to know the entire startup sequence of that vehicle, unlike a game such as DCS for example, where you have to memorize the entire start-up sequence, which can take up to 5 minutes at times. My main vehicle type that I have interest in is as you might have been able to guess by the title of the document, planes.

I have had quite an interest in the physics behind flight, and even more how that is implemented in games. Therefore, since starting the study CMGT, I have wanted to try and create my own flight simulator, even if just purely as a simulator, and not specifically a game with a play loop and all else that's necessary for a game.

How will I do this?

The project is made from two parts, both targeting one of my learning goals from this project.

The first part is learning about general flight physics, as well as how it's implemented in other solutions/games. Questions can be, but are not limited to; What is the formula for lift, drag, thrust and gravity? How do certain properties of a wing, engine, fuselage change the overall characteristics of the aircraft they are a part of? How do well known games implement the physics? If they simplified a formula or system, how and why? How could I best go about implementing the systems and physics? How would I go about designing the systems and simulator to be able to expand it in the future? If possible, how were a few aircraft designed, what was kept in mind or focused on?[1]

The second part of the project is the actual making of the simulator.[2] I will be making use of the Unity Engine. This will be the main part of the Dev-log, as it will be the part of the project where I will run into the most situations where I have to make choices, try to solve issues that come up during development for whatever reason. And generally, the easier part to reflect and log as it's actual development, instead of just writing stuff down on a piece of paper or in a document.

[1] This question is mainly something for if I have time and am able to easily find information on it, as it's more a question out of interest then necessity.

[2] As noted in my devlog and reflection, this part of the item has been cut for this specific course because of a lack of time, which is due to me underestimating the scope of this document.

What this document is

As the title goes, this is a research and design document for the Flight Simulator project. In this document I go over the things I've researched in regard to flight, the physics involved and how current games implement it. After that there are a few pages where I go into how I would approach the actual creation of the project, this is the Design Document.

Technical Document

The Technical documents will be separated into 2 sections: Real life physics and game implementations. The first part I will try to form an understanding of the basic principles behind flight, what important variables there are when it comes to calculating different forces, and how to calculate a few of the dynamic constant's e.g. Lift/Drag Coefficient. The second part of the research will be about current implementations of real time flight physics in established games. I will be looking at the open-source repository SimpleWings and the SDK from Microsoft Flight Simulator as that can give me quite deep insight into how they have their model's setup, what variables they don't dynamically calculate but instead hard set, and possibly things that they tried and found wouldn't work. The latter will help me avoid getting stuck with a problem or issue that is not actually resolvable, or at least not in the time frame that I have.

Design Document

The design document is where I go over what I want to implement in broad terms. With a few of those things I might mention how I plan to implement them as well.

This will give me an idea on what and how I want to do things once I come to the actual development phase. I plan to put most of my thoughts/plans in there as text, however I might sometimes add an illustration to make my idea come across easier, or for things that are complex to explain but easily shown through a diagram. This might be something like a flight-model loop (see Figure 1), or connections between systems/classes.

The main idea behind this document, however, is once again, to determine the scope of the prototype. I discuss what I plan to implement, and why I want to have those features/additions part of the prototype.

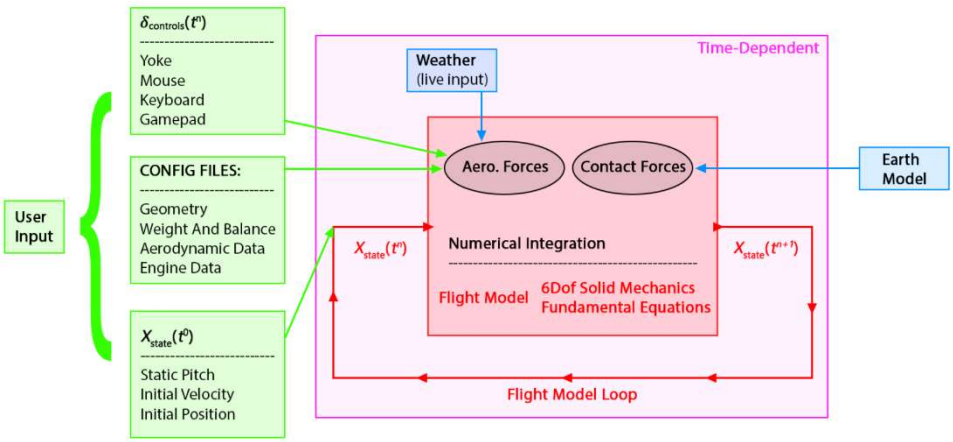

Figure 1 - High-level Flight-Model loop diagram as seen on the MFS SDK page about Flight-Model physics.

Research - Fundamental Aerodynamics

Introduction: Physics simplified

To understand how an aircraft flies, you need to understand what air is. Air consists of thinly spread molecules. This means that as you move through air, you are going through, and against, molecules. These molecules provide resistance, as well as pressure. This resistance, as well as the presence of molecules in air, is what flight is dependent on. Flying can be broken into 4 forces; **Lift**, **Gravity**, **Drag** and **Thrust**.

Gravity should be a something you are familiar with, so I will give a simple explanation. Gravity is a force that's applied to anything with a mass. On earth the force applied is about 9.8 N/Kg. This means that for every kilogram of mass an object has, it will experience 9.8 N of force directed to the ground.

Lift is the force that counteracts gravity. Lift is created by the wings of an aircraft as air flows past them. There are a few different explanations that try to explain this simply, but they are all lacking one way or another, so provided will be the one which is the easiest to explain in a paragraph.

As airflow passes by the wing, at the front of the wing the airflow gets split in half. The top half of the airflow goes over the wing, when it does so it first faces upwards, and when it leaves the wing, the air flow is directed downwards. According to the 2nd law of newton, this requires the wing to exert force on the air pulling it down, and according to newtons 3rd law, every action has a reaction, so the air puts a force on the wing, pulling it up in a sense. The airflow under the wing is diverted downwards, pushing the wing up. This is the pressure/force you feel if you stick your hand out the window while driving in a car. If you hold your hand at a 45° angle, you will feel the wind pushing your hand up.

Thrust is the name for the force that the engine(s) on an aircraft applies in a forward direction. The way that thrust creates force depends on the type of engine (propellor, jet, rocket). A propellor creates thrust by pushing air backwards, the same way a boat screw pushes water back to propel a boat forwards. An aircraft propellor does this at a higher RPM (rotations per minute) due to the lower density of air compared to water. A jet engine creates thrust by making a lot of air move quicker out of the engine than it comes into the engine causing the air to push back on the engine, moving it forwards. A rocket engine creates thrust by burning a fuel, which leaves the engine at high speed, pushing the engine forwards. These are very simple explanations, of which only the propellor engine I will dive deeper into.

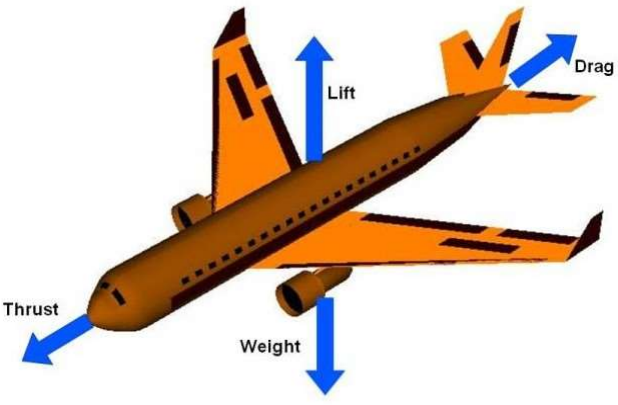

Figure 2- The different forces applied on a plane. Parallel to the fuselage are Thrust and Drag, while perpendicular are Gravity (Weight) and Lift.

Drag is the resistance that air provides against the aircraft and counteracts thrust. Molecules inherently try to stay close to each other, the degree to which they achieve that decides the density. When air passes the aircraft, it will try to stick to the aircraft, this will create a form of friction with the other air molecules.

These four forces together decide how an aircraft behaves in flight. As seen in Figure 2, the forces are in the horizontal and vertical planes. As long as these two directions forces are balanced, the aircraft will not change speed or altitude. If, however, the lift force goes up for example, then the aircraft will have a positive force directing up, meaning that the aircraft will gain altitude. The unbalance of these forces is how a pilot can control the altitude and speed of an aircraft.

Control of an aircraft

Now we have a basic understanding of how the 4 forces applied during flight work, but not how a pilot could actually change the forces in a way that allows them to manoeuvre the aircraft. This is where the control surfaces and other systems and mechanics come into play. There is a barrage of controls in modern aircraft that allows a pilot to change course, speed, altitude and attitude. However, seeing as the scope of this project is not so big as to simulate all the bells and whistles that modern airliners and fighter jets have, we will keep it simple and stick to common control types which is present on nearly every aircraft.

Control surfaces

The pilot tends to have access to three different control surfaces, each influencing a certain axis of rotation (see Figure 3). The three control surfaces are ailerons, elevators and the rudder(s). Ailerons are present on the wings, usually favouring the wing tips compared to the wing root (where the wing goes 'into' the fuselage), and they control the roll of the aircraft. The tail of the aircraft hosts the remaining two control surfaces. On the horizontal stabilizers you will find the elevators, which control the pitch of the aircraft. Finally, you have the rudder which controls the yaw of the aircraft.

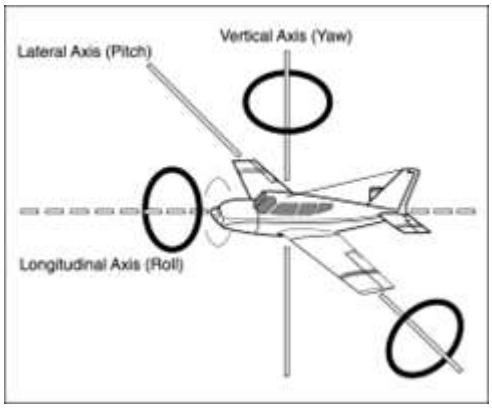

Figure 3 - The three axes of rotation of a plane.

Ailerons

Ailerons are one of the more important control surfaces, namely on larger aircraft, as they control the roll of the aircraft, and thereby the direction that you fly to. Ailerons are mounted on the trailing edge of each wing, near the wingtips and deflect in opposite directions. When a pilot moves the aileron control to the left for example, the left aileron deflects up and the right aileron deflects down. A raised aileron lowers the lift produced by the wing and a lowered aileron raises the lift, so in the example the right wing will drop, and the left wing will rise. This causes the aircraft to roll and therefore turn to the right. Centring the aileron controls then returns the ailerons to the neutral position, maintaining the roll angle. The aircraft will remain in that roll until a opposite aileron motion is induced to return the aircraft back to a level flight.

Elevators

Elevators are the easiest way for the pilot to control the pitch of the plane. The easiest way to explain how the elevators control the pitch is the following: Imagine that a plane has a rod going all the way through the wings—in Figure 3 that would be the axis indicator—making the plane swivel around that line. Deflecting the elevators upwards

will make the back of the plane push down. As we know, push something down on one side of a rotation line, then the other side will go up, like a seesaw.

Rudder

The rudder is a primary control surface hinged to the trailing edge of the vertical stabiliser, used to control yaw (rotation about the aircraft's vertical axis) by redirecting airflow and generating a side force that swings the nose left or right. In coordinated turns, deflecting the rudder into the turn counteracts adverse yaw—the tendency of the aircraft to yaw opposite the roll direction due to differential drag on the wings—ensuring the turn is smooth without slipping or skidding. Pilots also use rudder inputs during crosswind take-offs and landings to align the fuselage with the runway centreline, and to maintain directional control at low speeds when aerodynamic forces on the wings are insufficient.

Forces of Flight

As said on page 6, there are 4 forces that together keep an object in the air: Lift, Drag, Thrust and weight. We will be diving into the first 2, as those are the more mathematical and theoretical interesting topics for us, as well as the more challenging concepts to implement in the eventual prototype. I will however, shortly

Lift

"A fluid flowing around the surface of a solid object applies a force on it. Lift is the component of this force that is perpendicular to the oncoming flow direction."

Basic Principle

The correct explanation for lift is more complex than the simplified version I gave in the Introduction. Many basic explanations rely solely on either Bernoulli's principle or Newton's laws of motion, the second being the way I explained it in the introduction, while in reality both play a role.

Lift is generated through a combination of Newton's 2nd and 3rd laws and the forces caused by pressure differences around the wing. As air flows over the wing, the angle of attack and shape of the airfoil deflect the air downwards. According to Newton's third law, this downward force on the air results in an upward force on the wing – this is one component of lift. At the same time, the wing alters the speed of the 2 airflows around the wing; the air above the wing speeds up, while the air below slows down. This difference in speed creates a pressure difference, low above, high below, which results in a net upwards force. These pressure differences however are not limited to just the surface – they extend into the space around the wing, forming what's known as a pressure field (see *Figure 4)*.

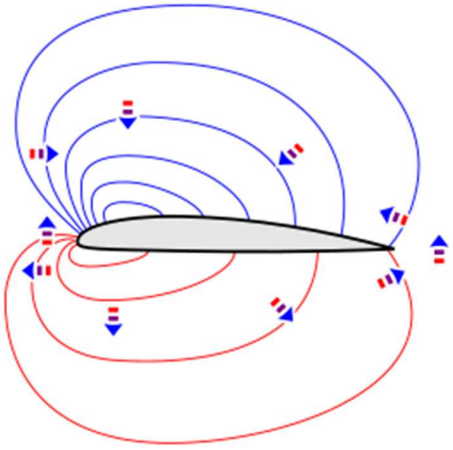

Figure 4 - Pressure fields around an airfoil. The lines are isobars of equal pressure along their length. Arrows indicate the acceleration of air in that direction, due to pressure differences between high (red) and low (blue).

For instance, incoming air is slowed and deflected downwards by the high-pressure region under the wing that it collides with. Meanwhile, air above the wing accelerates as it's drawn into the low-pressure region. These changes in velocity and direction are not passive, they actively maintain the pressure field, which in turn keeps redirecting and altering the speed of the airflow. This feedback loop, where pressure gradients accelerate the flow and the flow sustains those same gradients, is fundamental to maintaining lift in a real aerodynamic system.

Airfoil Geometry & Camber

An airfoil—the two-dimensional cross-section of a wing or propeller blade—is a streamlined shape engineered to produce significantly more lift than drag by deflecting airflow and creating pressure differentials across its surfaces.

Camber, the curvature of the mean line between the upper and lower surfaces, shifts the lift-coefficient (C_L) curve and zero-lift angle: positively cambered airfoils generate lift at 0° angle of attack, moving the zero-lift point into negative AoA and increasing the lift-curve slope, whereas symmetric airfoils require positive AoA to develop lift and exhibit more benign behaviour when inverted. The trade-off is that cambered profiles, while lowering stall speed and boosting cruise efficiency, introduce stronger pitching moments and can lead to sharper stall onset if over-cambered; symmetric sections, conversely, stall more gently and simplify structural design but demand higher speeds for the same lift. Finally, stall characteristics vary with camber shape: higher camber raises the maximum lift coefficient (C_{LMax}) yet concentrates flow separation more abruptly at the leading edge, causing a sudden loss of lift when the critical AoA is exceeded, whereas lower or reflexed cambers delay separation and yield more gradual stall behaviour at the cost of reduced peak lift.

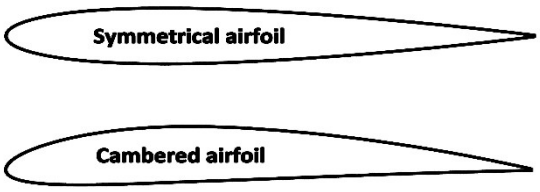

Figure 5 - A symmetrical and asymmetrical (cambered) airfoil. Note how the cambered airfoil has a flatter bottom then top. This difference in thickness from the middle line is how you can identify a cambered airfoil.

High-Lift devices (Flaps and Trim)

Wings often include movable surfaces besides the control surfaces I mentioned on page 7, being the flaps and trim. These movable surfaces are used to boost the lift created during low-speed manoeuvres or actions, such as take-off and landing. Flaps—such as plain, split or Fowler type—extend or deflect downwards to increase the camber of the wing, the surface of the wing or both, raising the maximum lift coefficient (C_{LMax}) and raising the stall angle. This lets the aircraft fly safely at lower speeds. However, Flaps don't only have positive influence on the flight characteristics. Deploying flaps also increases drag—from both the larger frontal area and larger disturbance of airflow—so they are retracted once the extra lift is no longer needed.

Trim tabs are small adjustable tabs on the trailing edge of control surfaces. By deflecting a trim tab, the pilot sets a continuous small force that counteracts control-surface load. As seen in Figure 6, in practice the trim tabs alter the net camber on the tail/elevator, allowing hands-off steady flight at a desired pitch without constant input from the pilot.

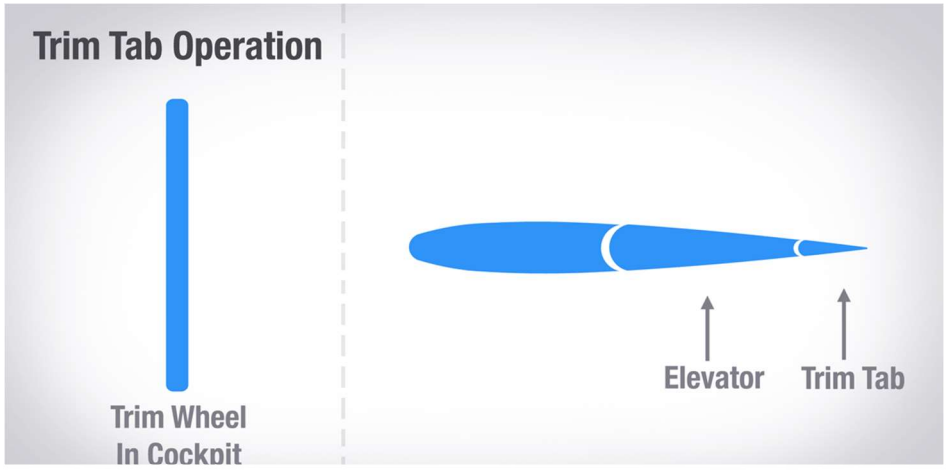

Figure 6 - Animation showcasing how the elevator trim tab works. Because the elevator is on the tail, an increase in lift will pitch the nose down, due to an imbalance in lift between front and back of the plane.

Quantitative Model - Equations

Now, the mathematics behind lift is—thankfully—simpler than the explanation behind lift. To be able to calculate lift you need only one formula, being the following:

$$\boldsymbol{L = \frac{1}{2}\rho v^2 S C_L}$$

where

- $\boldsymbol{L}$ is the generated lift force.
- $\boldsymbol{\rho}$ is the air density.
- $\boldsymbol{v}$ is the velocity of the air over the wing, also known as true airspeed (TAS)
- $\boldsymbol{S}$ is the planform wing area.
- $\boldsymbol{C_L}$ is the lift coefficient at the desired angle of attack and Mach number.

With this formula you can calculate the lift generated by a specified surface area, assuming you also have the air density, velocity of the air over the surface area, and the lift coefficient graph.

The formula to get air density at a specific altitude is:

$$\boldsymbol{\rho(h) = \rho_0 e^{-h/H}}$$

where

- $\boldsymbol{\rho}$ is air density as a function of the altitude.
- $\boldsymbol{h}$ is the altitude.
- $\boldsymbol{e}$ is Euler's number, an irrational constant.[1]
- $\boldsymbol{\rho_0}$ is the density at sea level constant (1.225 kg/m^3).
- $\boldsymbol{H}$ is the scale height of the atmosphere (≈ 8,500m).[2]

1 An irrational constant cannot be represented as a ratio of integers. This means you can't get the number by dividing one whole number with another whole number.

2 As further clarification, this means that with standard weather (temperature and gravity) conditions, which we will assume for our case, the air density reduces 63% every 8500m.

Drag and Fluid dynamics

"In aerodynamics, aerodynamic drag, also known as air resistance, is the fluid drag force that acts on any moving solid body in the direction of the air's freestream flow."
- *Wikipedia.com*

"In fluid dynamics, drag, sometimes referred to as fluid resistance, is a force acting opposite to the direction of motion of any object moving with respect to a surrounding fluid."
- *Wikipedia.com*

Basic Principle

Drag is the aerodynamic force opposing an object's motion through a fluid—air, in our case—acting parallel to the flow and directly against the direction of travel. As an aircraft advances in the air, air molecules collide with its surface and must be pushed out of the way. By newtons third law—mention before on page 9—imparting energy on those molecules forwards, leads to a reactionary force in the opposite direction, manifested as drag. Drag scales quadratically—just like lift—with the object's speed, and linearly with the object's shape and the density of the liquid the object is moving through.

There are two main components contributing to the parasitic drag[1] during flight, being friction and pressure drag.

Friction drag

Skin friction is a drag induced by the viscosity of the liquid—air, in our case—a object is moving through. Due to the viscosity of the liquid, the molecules closest to the surface of the object are subjected to the no-slip condition, which is a principal from fluid dynamics that states that liquids with a viscosity lead to molecules closest to the surface of the object having a relative speed of 0—meaning that they don't move. You can see this as a form of stickiness that a liquid has on a molecular level. This stickiness also then applies to itself, so the molecules that stick to the surface of the object try to pull forwards—in the opposite direction of their original travel—the molecules that go past them. Using the previously discussed 3rd law of newton, we know that the molecules then pull back on the molecules on the surface, leading to a pull back on the object.

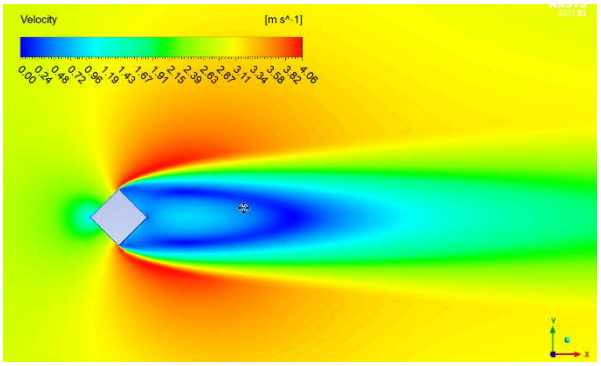

Figure 7 - A CFD[1] computed plot showing the velocity of air around a diamond shaped object. You can see high pressure on the top and bottom, and low pressure behind the object.

[1] Parasitic drag is not a result of a positive force, and therefore not useful—hence the term parasitic—unlike lift induced drag, which is made during the generation of lift by an airfoil.
[2] Computational Fluid Dynamics (CFD) is a numerical technique that uses computer algorithms to approximate the equations governing fluid flow—allowing engineers to predict airflow patterns, pressure, and velocity around objects without physical wind-tunnel testing.

The rougher a surface is the more 'pockets' of air get stuck in those imperfections, leading to a bigger boundary layer—the layer of still or slower moving liquid molecules. This in cause leads to more friction drag. Imperfections have the same effect; they lead to more surface area that can be affected by friction drag.

How strong friction drag is, is partly linked to the Reynolds number, which is a dimensionless number that dictates the type of flow that the liquid is exhibiting. The Reynolds number is used to predict the transition from laminar to turbulent flow (see Figure 8).

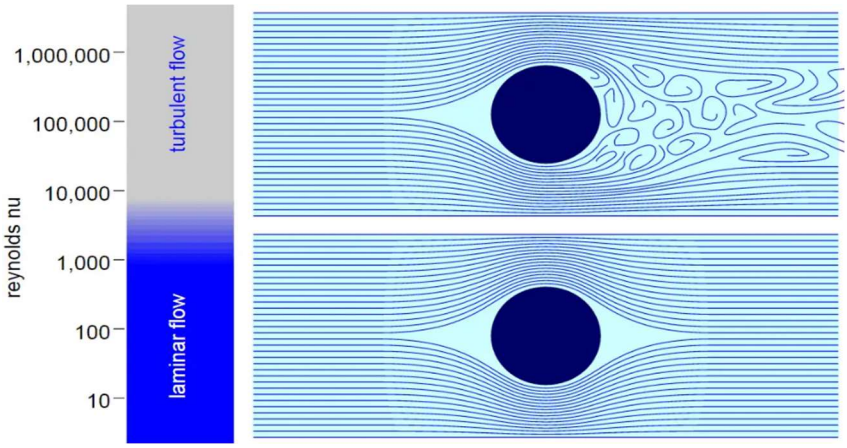

Figure 8 - Streamline patterns around a circular cylinder at different Reynolds numbers. The left panel shows the Reynolds-number regimes. The right panels depict flow over a cylinder The right panels depict flow over a cylinder: in laminar flow (bottom) the streamlines remain entirely attached around the cylinder, producing a steady, symmetric wake, whereas in turbulent flow (top) the boundary layer transitions and separates earlier, producing an unsteady wake with vortex shedding.

Pressure Drag

Pressure drag—also called form drag, since it directly depends on the body's form—arises from the pressure differential between the forwards-facing and rear-facing surfaces of a body as air flows around it.

When a bluff object—one with a non-streamlined contour—moves through laminar free-stream flow (see Figure 9), the boundary layer cannot navigate the abrupt curvature at the trailing edge of the body. As a result, the flow separates from the object and forms a turbulent wake of low pressure behind the body. This wake "suction" created a net retarding force equal to the pressure difference integrated over the body's surface.

Streamlined shapes (e.g. teardrop profiles or raindrop forms) delay the separation of the boundary layer by maintaining a attached flow over a greater surface length, thereby reducing the wake size and pressure drag, though at the expense of increased skin-friction drag due to the larger surface in contact with the free-stream flow.

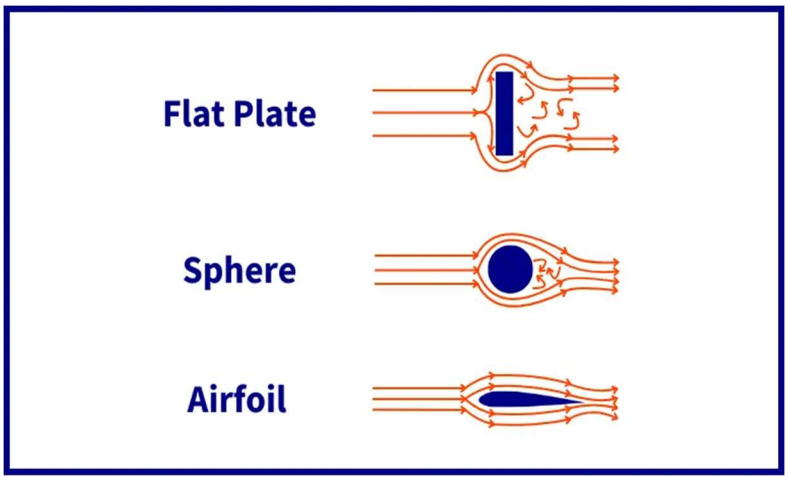

Figure 9 - Freestream flows around a bluff object, a sphere, and a airfoil. Note the reducing wake size as the object becomes more streamlined.

Quantitative Model - Equations

The formulas for calculating drag are just like the formulas for lift, simpler then the theory. The formula for the overall drag is as follows:

$$F_D = \frac{1}{2}\rho v^2 C_D A$$

where

- F_D is the drag force,
- ρ is the density of the fluid,
- v is the speed of the object relative to the fluid (TAS),
- A is the cross-sectional area and
- C_D is the drag coefficient.

The drag coefficient depends on the shape of the object and on the Reynolds number.[1] As you can see from the formula, the total drag is quadratically linked to the speed of the object relative to the fluid.

This formula accounts for both parasitic drag as well as lift-induced drag.

1 There is a formula for calculating the Reynolds number, and from there one can determine the CD through 2 different methods—the one you use being dependent on the Reynolds number. However, since this is by far out of the scope of the project that this research document is meant for, I will not be going into how the Reynolds number is calculated.

Thrust

For heavier-than-air objects, there needs to be a form of thrust to be able to generate the lift needed to get the object into the air. This can be achieved in multiple ways. The three most common methods of creating thrust are through the use of a propeller engine, a jet engine or a rocket engine. All three of these use fuel to power a mechanism that provides a forward's momentum—assuming the engine is facing forwards.

Due to the fact that I will and cannot implement all three methods of thrust into the prototype, I will only be delving further into only one of the three aforementioned methods of creating thrust, namely, the propeller engine, as it has the most potential for further development and addition of physics.

All engines share the same principle for creating thrust, which is pushing matter backwards to propel the engine—and with it the affixed body—forwards.

Propellers

In aeronautics, an aircraft propeller, sometimes referred to as an airscrew converts rotational motion from an engine or other power source into a swirling slipstream which pushes the propeller forwards or backwards depending on the configuration. The propeller comprises a rotating hub, to which several radial airfoil-section blades are attached. The pitch of the propellers can be fixed, manually changed to a few set positions, or of the automatic variable "constant-speed" type.[1]

The way that the propellers create thrust is practically the way wings create lift, which is also why you can see similarities between propeller blade airfoils and wing airfoils (see Figure 10).

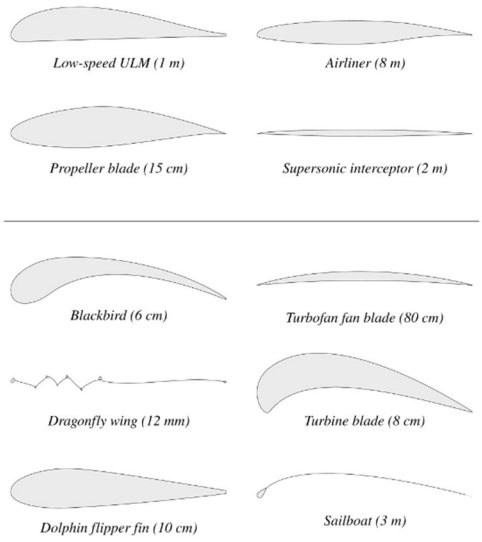

Figure 10 - Different airfoils, showcasing the similarity between wing airfoils and propeller blade airfoils.

There is no reliable formula for the calculation of the thrust that a propeller produces, even if the principle behind the generation of thrust is the same as the generation of lift. This is due to the angle of the airfoil relative to the direction of the free-stream flow. However, there is an empirical method of calculating thrust, which is what I will be using in the prototype. The equation goes as follows:

$$\boldsymbol{T = C_T \rho n^2 D^4}$$

where

- $\boldsymbol{T}$ is the thrust generated by the propeller,
- $\boldsymbol{C_T}$ is the thrust coefficient,
- $\boldsymbol{\rho}$ is the density of the liquid,
- $\boldsymbol{n}$ is the rpm of the propeller, and
- $\boldsymbol{D}$ is the diameter of the propeller (in meters)

With this one can make an estimation of the thrust made by a single propeller, and the C_T value can be tweaked in the simulator to adjust for other changes compared to real-life physics.

1 Constant speed variable pitch propellers are propellers that change the pitch of the blades to maintain a constant engine rpm, optimizing the engine performance during flight even with changing flight conditions.

Stability derivatives

An aircraft inherent stability depends on its design. The aircraft's ability to correct its attitude in response to changes in environment or control determines how stable it is. A stable design resists or damps out these perturbations without the need for constant pilot input.

An aircraft has stability in the three axes of the plane, which are the following:

- "X" or "x" axis runs from back to front along the body, called the *Roll Axis,*
- "Y" or "y" axis runs from the left wing to the right wing and is called the *Pitch Axis,*
- "Z" or "z" runs from the top to the bottom, called the Yaw Axis.

These three axes all run through the *centre of gravity*, called the CG, of the plane. There are two slight variants of these axes, depending on the situation and context: ***Body-fixed axes*** and ***stability axes***.

Terms

There are a significant amount of terms/parameters introduced with the formulas required for calculating the momentum and torque forces made by the natural stability of an aircraft.

Body-Fixed axes

Body-Fixed axes, or body axes, are defined and fixed relative to the body of the aircraft. This in coordinate space terms means they are object-space axes.

- X body axis is aligned along the fuselage and is usually directed towards the direction of motion.
- Y body axis is perpendicular to the X body axis and is oriented along the wings of the vehicle, usually directed to the right (left wing is negative, right wing is positive).
- Z body axis is perpendicular to the XY plane and usually points downward.

Stability axes

Aircraft usually operate at a constant angle of attack (as mentioned in the Lift section). The angle of the nose does not align with the direction of the oncoming air. The difference in these directions *is* the angle of attack (AoA). In a way the stability axes are the body axes rotated about the Y body axis by the trim[1] AoA, and then re-fixed to the body of the aircraft.

- X stability axis is aligned into the direction of the oncoming air in **steady** flight.
- Y stability axis is the same as the y body-fixed axis.
- Z stability axis is perpendicular to the XY stability axes plane.

Forces and Angular momentum

There are a few more terms, namely for the forces and velocities along each of the axes and the moments and angular rates around each of the axes.

---

1 The trim is the state of the plane where all the aerodynamic forces and moments on the aircraft are balanced out, so no accelerations or rotations are occurring. That means that Lift = Gravity, Thrust = Drag and no moments are present (M, L, N = 0).

For the forces along each of the axes you have the following:[1]

- X, or F_X, is used to indicate forces along the X axis, and
- Y, or F_Y, is used to indicate forces along the Y axis, and
- Z, or F_Z, is used to indicate forces along the Z axis.

For the velocities of the oncoming flow on the axes you have the following:[2]

- u is the speed along the X body axis, and
- v is the speed along the Y body axis, and
- w is the speed along the Z body axis.[3]

For the moments and angular rates around each of the axes—whether to use body axis or stability axis depends on the context, such as the subscript—you have the following parameters:

- L is used to indicate the **rolling** moment, which is around the X axis.
- M is used to indicate the **pitching** moment, which is around the Y axis.
- N is used to indicate the **yawing** moment, which is around the Z axis.
- P, or p, is used for angular rate about the X axis—*Roll rate about the roll axis*.
- Q, or q, is used for angular rate about the Y axis—*Pitch rate about the pitch axis*.
- R, or r, is used for angular rate about the Z axis—*Yaw rate about the yaw axis*.

Linearised Moment Equations

The pitching, rolling and yawing moments all derive from the same base formula, differing only by their specific stability derivatives and characteristic lengths. This base formula can be described as follows:

$$\boldsymbol{dynamic\ pressure * conext\ area * length *}$$
$$\boldsymbol{(sum\ of\ pertubation\ coefficients * their\ variables)}$$

We will start with the pitching moment, as it slightly deviates more than the other two.

Pitch (Longitudinal) Stability

The pitching moment M about the lateral axis is linearised around the trim as:

$$\boldsymbol{M = qSc(C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{qc}{2V})}$$

where

- $\boldsymbol{q}$ is the dynamic pressure: $q = \frac{1}{2}pV^2$. You can regard it as the wind strength, and
- $\boldsymbol{S}$ is the planform wing area (m^2), and
- $\boldsymbol{c}$ is the mean chord length—a characteristic length of the wing, and
- $\boldsymbol{C_{m_0}}$ is the pitching-moment coefficient at the trim condition (where $M = 0$). If the aircraft is perfectly balanced at its trimmed angle, C_{m_0} is zero; if not, it shifts the whole curve up or down, and
- $\boldsymbol{C_{m_\alpha}}$ is the AoA term, it tells you how much the moment coefficient changes per radian of angle change, and

1 These forces are along the body axes and are therefore called Body-Axis Forces.

2 The velocities are of the oncoming flow projected onto the body axes. It is easier to think of these speeds as projections of the relative wind vector on that respective axis, rather than in terms of the motion of the aircraft relative to the fluid. As the body rotates relative to the direction of the relative wind, these components change, even though there might be no net change in the actual speed of the aircraft.

3 The terms are purposely in lower case.

- $\boldsymbol{C_{m_q}\frac{qc}{2V}}$ is the pitch-rate damping term, it tells us how the moment changes per unit of a non-dimensional rate—this term resists pitching motions, providing dynamic damping.

Putting this all together you get the following order of operations:

1. Compute q, S and c.
2. Measure current α and pitch rate q.
3. Plug the values into the bracket.
4. Multiply the sum by qSc to get M in Newton-metres.

Roll (Lateral) & Yaw (Directional) Stability

The equations for the rolling moment L and yawing moment N are closer related. This is due to them both not needing the C_{x_0} that the pitch moment equation uses, they are therefore also slightly shorter/smaller. Besides that, the other change is that they use the span—the distance from one wingtip to another—b instead of the mean chord length c.

Rolling Moment L

$$\boldsymbol{L = qSb\left(C_{l_\beta}\beta + C_{l_p}\frac{pb}{2V}\right)}$$

where

- $\boldsymbol{q}$ and $\boldsymbol{S}$ are the same, dynamic pressure and wing planform area respectively, and
- $\boldsymbol{b}$ is the wingspan, the distance from one wingtip to the other, and
- $\boldsymbol{C_{l_\beta}\beta}$ is the sideslip term, it gives the restoring roll moment when the aircraft sideslips by angle β, and
- $\boldsymbol{C_{l_p}\frac{pb}{2V}}$ is the roll-rate damping, it resists the roll rate p.

Yawing Moment N

$$\boldsymbol{N = qSb\left(C_{n_\beta}\beta + C_{n_r}\frac{rb}{2V}\right)}$$

where

- $\boldsymbol{q}$ and $\boldsymbol{S}$ are the same, dynamic pressure and wing planform area respectively, and
- $\boldsymbol{b}$ is the wingspan, the distance from one wingtip to the other, and
- $\boldsymbol{C_{n_\beta}\beta}$ is the sideslip term, it realigns the nose, and
- $\boldsymbol{C_{n_r}\frac{rb}{2V}}$ is the yaw-rate damping, it resists the yaw rate r.

From these formulas we can see that in the brackets the first coefficient term gives the static stability of the aircraft, while the second term gives the dynamic (damping) stability of the aircraft.

Research - Current Implementations

In this section of the research paper, I will be looking into and discussing how a few current games/simulators simulate their flight physics, how they structure their data, and how they simplified the physics—if they even did—to lower the load on the hardware.

I have 3 different titles I will be looking at: SimpleWings, Microsoft Flight Simulator (MFS) and finally WarThunder. That will also be the order I will go over them. This is due to the amount of information they release regarding how they implement the physics and systems.

SimpleWings is an open-source repository on GitHub meant for unity, I can go through the code myself to try and grasp how things are done, along with any explanations found.

MFS has a website for their SDK (Software Development Kit), where they go over how they calculate the future positions of the planes (simulating the physics), how they declare their data and possibly more I don't yet know about.

WarThunder has the least information about how they simulate their physics or the architecture of their flight models. Any information I do find will most likely be through unofficial channels, such as data mining or rigorous testing ending in assumptions/hypotheticals.

The 3 main points I will try and discuss with each of these sources are the following:

- How do they simulate their physics
- Did they do any simplifications of the physics?
    - If so, why?
- What is the architecture of the flight models?
    - How is data stored?
    - Why that way?

For SimpleWings I will also look into how the overall software architecture is done, as it will be a good reference for when I go into the design document.

Simple Wings

Project Overview

The SimpleWings project is made to give the framework for a simple aircraft with lift generating wings, using predefined curves to dictates the amount of lift and drag the lifting body creates. The project is made using Unity 5.6.7f1, and is under the MIT license, meaning that anyone can use, modify, distribute, sublicense and/or sell copies of the software, so long as the same copyright notice is passed along with further distributions.

The core components of the project/unity package are the SimpleWing.cs and WingCurve.cs files. These contain the main logic regarding the 'flight model', the calculations of lift and drag and the applying of those calculated forces. In the demo folder there are scripts made for basic flying operation using the two core components. This includes classes/code for control surfaces, weaponry, thrust and HMD/HUD.

Data Models

The way that SimpleWings stores the lift and drag coefficient data is with a SciptableObject called WingCurves—the core component. This scriptable object has two Unity AnimationCurves, one for lift, one for drag (see Figure 12).

These AnimationCurves can be set both manually by hand—to recreate the data of a specific airfoil for example—or semi-automatically through the variables that the user can set in the inspector. These auto-creation variables are introduced by the custom editor made for the WingCurves class (see Figure 12). The user can indicate certain important data points, such as the critical AoA point, which will then be interpreted into a lift curve (see Figure 11). The drag curve can only be set manually however it is usually a simpler curve, higher AoA means more drag.

The way the wings can get the data from the curve is by using the GetLiftAtAoA and GetDragAtAoA methods. These require the AoA to be passed as a float, and returns the corresponding value from the curve.

Figure 12 - Screenshot of the inspector for a WingCurves scriptable object. Note that only the Lift and Drag AnimationCurves are the only variables of the WingCurves class itself, the other fields are introduced by the custom inspector for the WingCurves class.

```
List<Keyframe> keyList = new List<Keyframe>(9)
{
    // Wing at positive AOA.
    new Keyframe(0.0f, neutralLift),
    new Keyframe(criticalAngle, maxLiftPositive),
    new Keyframe(fullyStalledAngle, minLiftPositive),

    // Flat plate, generic across all wings.
    new Keyframe(45.0f, kflatPlateMax * flatPlateMult),
    new Keyframe(90.0f, 0.0f),
    new Keyframe(135.0f, -kflatPlateMax * flatPlateMult),

    // Wing at negative AOA.
    new Keyframe(180.0f - fullyStalledAngle, -minLiftPositive * negativeAoaMult),
    new Keyframe(180.0f - criticalAngle, -maxLiftPositive * negativeAoaMult),
    new Keyframe(180.0f, neutralLift)
};
```

Figure 11 - The interpretation formulas for the generation of lift curves using user set values. Keyframes are ordered from left to right (0 to 180).

Implentation of physics

Most of the physics calculations are done in the second core component, the SimpleWings component. This is a MonoBehaviour script that can be added not only to the wing, but also to the fuselage and control surfaces. In the example prefab—Fighter Jet—that is in the demo, the fuselage has two objects both with a SimpleWing component, one object is for the vertical and one for the horizontal plane. The rudder, elevator and ailerons have both the SimpleWing component as well as the ControlSurface component. This means that the deflection of the control surface influences the AoA of the wing relative to the oncoming air (linear velocity of the plane).

Every single SimpleWing component applies a force to the RigidBody attached to the parent—why only the parent I do not know, that is not explained. This means that if you do not check the box to apply the force at the centre of gravity of the RigidBody, it will apply it at the transform origin point.

Practice

The forces are calculated and applied every FixedUpdate tick. If the user specified to apply the forces to the gravity centre, then it will cache the world space position of the centre of gravity of the RigidBody, otherwise it will cache the position of the game object that the component is attached to. Next the local velocity is determined. This is done by getting the velocity—direction and magnitude—at the position of the components game object and transforming that to local space. The spanwise component of the velocity is set to 0, as mainly the components perpendicular and parallel to the chord line of the airfoil generate the lift.

Now the angle of attack is determined by comparing the default forward vector—0,0,1— to the local velocity. Using that AoA the lift and drag coefficient are retrieved from the animation curves containing them.

Using those coefficients and the other data gathered so far, the following formulas are used to calculate the lift and drag forces:

$$\boldsymbol{F_L = localVelocity^2 * C_L * WingArea * Multiplier_L}$$
$$\boldsymbol{F_D = localVelocity^2 * C_D * WingArea * Multiplier_D}$$

where

- $\boldsymbol{F_x}$ is the resulting force, lift or drag, and
- $\boldsymbol{localVelocity^2}$ is the local velocity squared, and
- $\boldsymbol{C_x}$ is the drag of lift coefficient, and
- $\boldsymbol{WingArea}$ is the width * length of the wing, and
- $\boldsymbol{Multiplier_x}$ is the multiplier for lift or drag.

Due to Vector3.Angle always returning a positive value/angle, we add the sign (+ or -) back using the following line:

$$\boldsymbol{F_L *= -Mathf.Sign(localVelocity.y)}$$

Due to lift always being perpendicular to airflow, we get the cross product between the linear velocity and the local x-axis. This gives us the up direction relative to the wind/velocity direction. Drag is easier, due to it being the opposite direction of the velocity of the plane.

These forces are all applied to the RigidBody, where depending on the setting the user specified in the inspector (centre of gravity or position of components game object), as a continuous force. This mode of force application to the RigidBody accounts for the mass of the object.

Code annotations

Overall documentation wise, there is not a lot of actual documentation. This however makes sense if you account for the simplicity of the project and the physics it emulates. The XML documentation present is only in the WingCurve class, I believe due to it being the only core component where there are various custom methods, instead of making use of the Unity message functions (for example Start, Update, FixedUpdate). For the physics related things there is some inline documentation, but that's mainly for what is being done or why something is being done in very simple terms. There are however no further explanation or depth into why something is done in a specific way, or where formulas come from.

Ease of usage

The amount of customization a user has with this package is actually quite high compared to the amount of physics it emulates. A user can change the lift and drag behaviour quite easily to their desire and needs, and if the user has some basic knowledge, they can use the auto-curve generator to quite quickly remake certain NACA airfoils.

Besides that, the user can easily see the forces visualized if they turn on the gizmos (see), as well as the ability to see the values in the inspector of the SimpleWings component, thanks to the custom inspector.

Take-aways

- With small amount of physics implementations, you can already get quite far.
- Compartmentalizing the logic to separate functions can help make things clearer, as you can add XML documentation explaining in more detail how that step is being done, why it is being done and any improvements that can still be made.
- The way the plane/prefab/model is setup decides in part how the forces can be applied.

Microsoft Flight Simulator (SDK)

Overview

The Microsoft Flight Simulator SDK (Software Development Kit) equips developers with the tools, documentation and APIs needed to inspect and extend the simulator's flight-model internals. When you enable Developer Mode in the sim, a dedicated toolbar appears that lets you browse aircraft file layouts, monitor live simulation data, and access the complete SDK reference in HTML format. This documentation explains everything from the structure of key configuration files (such as `aircraft.cfg` and panel definitions) to the full list of SimVars and the interfaces for writing WebAssembly modules[1].

At the heart of the SDK are Simulation Variables (SimVars). These variables represent every piece of flight-physics data—true airspeed, angles of attack, angular rates, aerodynamic coefficients, atmospheric conditions, engine parameters and more. You read and write SimVars either through the SimConnect API or directly from within a WASM[1] module that plugs into the sim's runtime. SimConnect calls return single floats or arrays at each physics tick, allowing you to sample or tweak the simulation state in real time.

Under the hood, MSFS's flight-model logic is implemented as WebAssembly modules. The SDK provides C/C++ sample source for the "vanilla" flight-model, showing how lift, drag, thrust and control-surface forces are computed each frame. By building your own WASM plugin, you can inspect those same data structures, replace or augment control laws, and integrate custom aerodynamic routines, all while running inside the sim.

Aircraft-specific aerodynamic data—lift/drag polars, stability derivatives, thrust tables and prop-map curves—are stored in plaintext configuration files within each aircraft's `FlightModel` folder. These tables, typically formatted as CSV or simple name-value pairs, define coefficient values at discrete angles, Mach numbers or throttle settings. The SDK documents the exact file format, so you know where to plug in revised polars or altitude-density models.

Finally, the SDK's build and deployment workflow ties it all together. Using provided command-line tools or Visual Studio project templates, you compile your WASM module, bundle it into the appropriate `aircraft` or `community` folder, and launch the sim. In Developer Mode, the sim detects changes and hot-reloads your module, enabling rapid iteration on physics tweaks without restarting the entire application.

[1] Web Assembly Module (WASM) is a scripting language that allows for users to ship code that can be ran securely inside a larger application (such as a game).

The Implementation of Physics

Microsoft Flight Simulator employs a full six-degrees-of-freedom rigid-body model driven by ordinary differential equations, but it sources virtually all aerodynamic and propulsion data from precomputed tables rather than solving fluid dynamics in real time. In practice, every aircraft ships with a `flight_model.cfg` file (and related `.cfg` tables) that defines:

- **Lift and drag polars:** Coefficients $C_L\ (\alpha, Mach)$ and $C_D\ (\alpha, Mach)$ are stored as discrete lookup tables for a set of operating points (typically around 20 combinations of angle-of-attack, Mach number and control deflections). During flight, the sim simply interpolates these tables rather than computing aerodynamics on the fly.
- **Stability derivatives:** Linearised coefficients (e.g: C_{m_α}, C_{l_β}) live in similar tables. These capture pitching, rolling and yawing responses without modelling tip vortices or boundary-layer behaviour directly.
- **Propulsion and mass properties:** Thrust curves, propeller maps and engine performance are likewise table-driven, with mass and inertia defined in pounds and feet in the same config file.
- **Atmosphere model:** Standard-atmosphere density, pressure and temperature vs altitude are provided by the SDK; the sim references these values when computing dynamic pressure $q = \frac{1}{2}\rho V^2$.

By using lookup tables for every coefficient, MSFS achieves high fidelity to real-world published data while keeping runtime performance high. The trade-off is that transient effects (flow separation, dynamic stall, blade-element interactions) are baked into the tables—or omitted entirely—rather than simulated continuously. This discrete, data-driven approach ensures predictable, author-tuneable behaviour but relies on careful tuning of those underlying tables for each aircraft.

Take-aways

MFS implements all the equations/physics that I have talked about in the Research - Fundamental Aerodynamics section. This means that at first view, it does not seem that far off from SimpleWings implementations of physics, though I don't doubt that less hacky corner cutting methods have been employed, meaning what they both have in common physics wise, MFS probably does it in a way that more closely represents real life behaviour.

The main takeaway is that the possibility to get the same degree of realism in flight physics is quite high, just not the degree of realistic flight models. There is a lot of data, as well as testing that goes into that, which is something I most certainly will not have the time (n)or resources for.

Industry standard

After looking into a few other games and how they implement their flight physics, I have found out that most big simulators make use of the same solver—being the 6-DOF-rigid body solver that makes use of look-up tables. This is due to it currently being the most performant way of getting high fidelity flight simulation on a consumer system. The differences between the different simulators are slight variations in how they implement the physics specifically—for example how they apply the forces, as well as the amount of data used.

DCS for example makes use of extra data to extend the basic 6-DOF-rigid body system. They have additional tables that have dynamic surface-flow effects in them, they support multiple polars per flight regime (high-G turns, supersonic flight, etc.) and for rotorcraft (helicopters) they make use of a blade-element model, so instead of calculating it for the propeller, they calculate it per blade and add up the lift, drag and torque for the rotor disk.

WarThunder makes use of a multi-panel setup, meaning that a lifting surface is split into multiple panels. The calculations are done for every segment separately, and then summed together. This system is useful for the game as it allows them to quite easily implement damage models into the flight performance.

X-Plane's "Blade element theory" core computes forces on a multitude of spanwise strip elements, using thin-airfoil theory[1] equations on each strip. However, these equations are still closed form algebraic expressions, not CFD (on page 12). This system merely replaces a single look-up table pass with many small ones.

[1] Thin-Airfoil theory is a aerodynamic approach for slender wings and blades that are both very thin and have at most a small camber. The core result is that the sectional lift coefficient varies linearly with the AoA (in radians).

Design Document

As mentioned in the *What this document is* section, I discuss in this section of the document which features/physics I want to implement, why I want to implement them, and possibly how I plan on implementing them. I am making this document with the idea that I will make it in Unity 6. Which exact version depends on the time that I start the development process.

Features/Physics

The physics I plan to implement in the prototype are at minimum the physics discussed in the technical document. This means that I want to implement:

- Forces of flight
    - Lift
    - Drag
    - Thrust
    - Gravity
- Inherent Stability

Three extra forces related to flight and around it that I want to implement as well, if possible, are the following:

- Dynamic density
- Torque of the engine
- Friction

Gravity should already be covered by the built in physics engine of Unity, and depending on how I implement the wheels/gears, so should friction.

The reasoning behind this list is because of the test flying I did using the SimpleWings package. I noticed that it already feels quite realistic, however, there are a few aspects I am familiar with—realism wise, that I did not notice while flying around. This was side slip. This meant that if you were rolled, the plane would still fly straight, instead of slowly dipping to the side, or rotating around the yaw axis. I feel that that is a important aspect of realistic flight, at least for a propeller plane.

Outside of physics I want to simulate a few other things, however, these are more of a *want* then a *need*:

- Engine performance depends on environmental variables (height, temperature, speed, air density).
- Braking with the gear when on the ground.
    - Dynamic weight through fuel.
- Implementations of Trim/Flaps.
- Editor tool that allows for easier creation of flight models (data).

The reasoning behind these feature requests is mainly out of interest in the implementation, as well as a more game design aspect of the simulator.

Allowing the player to use trims and flaps gives them more degrees of freedom, besides being a challenge in how I approach implementing them. Braking with the wheels/gears allows for ‘gameplay’ on the ground, otherwise forcing the player to always

stay in the air. Besides that, it will allow me to easily—feature wise, implementation wise might be a different story—create incentive for the player to land and gives them a bit more of a challenge.

The engine feature is the lowest priority of them all, mainly because it is also the *want* feature that will—most likely—be the hardest to implement.

The Editor tool would allow me to iterate quicker through different data sets, making it easier to test the behaviour with different coefficient curves. Besides that, it will make it more friendly for people not familiar with or coding, or flight physics or both, to make flight models. Making it easier for other people to use the simulator, and create things for it, will allow me to get data easier, and find kinks and bugs quicker.

Data

I plan to use the same method as SimpleWings for the storing and evaluating of data, being the usage of the unity animation curve class. I will, however, look into a way to import known airfoil data—be that in a CSV[1], JSON[2] or plain text file—to make it easier to generate higher fidelity curves, that more accurately represent the actual data they are based off of.

Data of the airplane will be held by either a scriptable object that is referenced by the plane controller or held by the plane controller script itself. This information could for example be the mass of the plane, which is then put into the RigidBody component.

The same can be said for the engine. It will hold its own curve (Thrust, air density) in the class itself, this would allow me to make prefabs of engines, with their own data sets, which can then be interchanged. Compared to putting the data in a scriptable object and holding a reference to that instead, this allows you to switch out the model of the engine as well. However, that needs to be deliberated on more, as it is also not a feature I really need.

For the information of the environment, I plan to have another scriptable object, which is referred to by a manager—Game or Environment. This would then be combined with something like a singleton pattern, to allow the planes to get the data directly without needing a specific reference.

Design Patterns

As said above, for my data I am planning to use a ScriptableObject-drive pattern. This means that I create scriptable objects for certain data sets, for example, WingData could hold the lift and drag curves and EngineData would hold the thrust force data relative to altitude or other variables.

For the actual calculation of forces I plan to use a Component Pattern. This means that every aspect of the flight will be broken into sections, for example, LiftingSurface would hold the logic for lift, drag and potentially stability derivates. Engine would hold the logic behind thrust, and gears would hold the logic for lowering and raising gears. Like this

---

[1] Comma Seperated Values, commonly refered to as CSV, is a text data format that uses commas to separate values, and newlines to separate records.

[2] JavaScript Object Notation, commonly refered to as JSON, is a standard file and data interchange format that uses human-readable text to store and transmit data objects consisting of name-value pairs and arrays.

links are only made when needed, and it's easier to know what class/component does what.

For global services such as game or environment information, I will make use of the Singleton pattern. This means that a class has a static **Instance** variable that will be set at the start of the scene, ensuring that only one instance of that component/class is present in the scene. If there are two present, then the second one to initialize will destroy itself, to avoid having multiple instances. This will also allow scripts to get data from that class without a direct reference to the component instance.