Andreas Larsen

Methods of Real-Time Flight Simulation

December 16th, 2009

A study of real-time flight simulation by state-of-the-art methods in physics simulation and game technology.

Supervisor:	Torbjørn Hallgren
Co-Supervisors:	Jo Skjermo, Helge Langseth

Norwegian University of Science and Technology (NTNU)
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

TDT4570
Game Technology,
Specialization Project

NTNU
Norwegian University of Science and Technology

Abstract

This paper identifies novel and best-practice methods for real-time simulation of aircraft behavior in a virtual environment. The work is preliminary to a master's thesis on autonomous aircrafts and forms the theoretical background for implementing a flight simulator game to develop and test autopilot software on. Basic principles of aerodynamics are briefly introduced to help understand how aircrafts fly and we investigate methods to simulate the flight behavior. The outdoor environment is particularly difficult to visualize in real-time so this study covers recent methods to render natural scenes by tricks commonly used in games. Finally the study covers two flight simulator games that are freely available and discusses whether some of the work may be reused for this project.

The conclusion of this report is a set of methods and approaches to best realize the simulator software in light of performance, realism, visual appeal and the scope of the project.

Contents

List of Figures

Nomenclature

airfoil A structure placed in an airflow for the purpose of generating lift

BTF Bidirectional Texture Function

CFD Computational Fluid Dynamics

chord Imaginary straight line from the leading to the trailing edge of an airfoil

collective pitch The base angle of attack for all blades of a helicopter

culling Omit geometry to speed up the rendering process

cyclic control Tilting the helicopter by adjusting the angle of attack for each blade in the rotation cycle

equal transit time A condition that states that an airflow diverted around an airfoil must rejoin at the trailing edge in the same amount of time

FDM Flight Dynamics Model

fineness ratio The length divided by the maximum diameter of a body

ideal gas A theoretical gas of randomly-moving point particles that interact only through elastic collisions

inertial force A pseudo-force equal but opposite in direction of the accelerating force

laminar flow When a fluid flows in parallel layers with little or no disruption between the layers

LOD Level of Detail

MAC Mean Aerodynamic Chord

N-S Navier-Stokes, rerfers here to the equations governing mass, momentum and energy in a fluid

pitch angle For a rotor blade it is the angle of attack and for an aircraft it is the nose angle around the wing axis

RAF Relative Airflow

roll angle The tilt angle around the nose axis

structured grid A grid of rows and columns

supersonic flight Travelling faster than the speed of sound

thermal Parcels of warm air that rise upwards due to heating by sun radiation on the ground

thermal equilibrium Systems at constant temperature and volume

turbulent flow A flow of fluid characterized by chaotic, stochastic movement

unstructured grid A mesh without a direct mapping to rows and columns

viscosity A fluid's resistance to flow due to internal friction

VTM Vorticity Transport Model

yaw The horizontal angle of the aircraft's nose axis

1 Introduction

1.1 Purpose

This report presents a survey of state-of-the-art methods suitable for real-time flight simulation. The findings here is the preliminary work of a master's thesis on autonomous flight and form the theoretical foundation required to implement the simulator part of the software. This paper focuses solely on the methods related to flight simulation and will only briefly mention the topic of autopilots here.

1.2 Motivation

The field of robotics has gained significant interest over the last two decades and its applications are ever increasing. Factories use robots to automate and speed up processes and commercial airliners have fully autonomous autopilots that almost render pilots obsolete.

This project was inspired from the idea that swarms of cheap off-the-shelf model aircrafts could solve tasks safely and efficiently. The costs of issuing a search and rescue helicopter are tremendous and it takes a lot of time to cover a large area. With swarms one could ideally launch hundreds or even thousands of drones capable of navigating by themselves and sending live pictures to a ground station. This way a large area could be covered more quickly and potentially a lot cheaper. For hazardous tasks, such as in irradiated areas or in extreme weather conditions, the drones could be sent instead to avoid putting humans at risk.

In order to develop autopilots safely and with minimal risk of failure it is necessary to have a realistic and flexible flight simulator software to test the autopilots on. The development of such a simulator is the motivation for this study.

1.3 Context

Flight simulators often use game technology to achieve realistic and appealing graphics at interactive frame rates. Ever since the emergence of computer games there has been a synergy between game development and academic work, which have led to the rise of *serious* flight simulators that are both recreational and educational. Serious games aim to help understand a problem or develop real-world skills and flight simulators have become particularly important in areas such as training pilots and evaluating new aircraft designs.

A lot of prior work exist in the field of flight simulation and this study discusses some of the methods that look promising. The goal is to apply such methods in the future implementation of a physically oriented game that simulates flight behavior in a realistic manner. In particular the study focuses on how the simulator can be realized as a serious game that is intuitive to test the autopilot's performance in and that is fun to experiment with.

It is the intention that this report will provide the reader with a primary source of reference when reading the master's thesis and give the necessary background to understand the methods and choices made in the future implementation.

1.4 Intended Audience

This study is relevant for any reader interested in an entry level introduction to aerodynamics and real-time simulation of flight. The intention is not to give intricate details on methods and implementations but rather give an overview of a few promising methods and how they may be applied in this setting. It is assumed that the reader has a good understanding of graduate level calculus and physics to follow some of the equations listed, but beyond that the text should be easy to follow for the uninitiated.

1.5 Overview

The structure of the report is as follows.

Chapter 2 Introduction to theory on aircraft simulation. It introduces aerodynamics and describes methods for simulating and visualizing flight behavior.

Chapter 3 Overview of related work for comparison and inspiration.

Chapter 4 Discussion of methods and how well they fit the implementation.

Chapter 5 A summary that proposes a starting point for the real-time flight simulator implementation.

2 Theoretical Background

2.1 Basic Principles of Aerodynamics

In order to simulate aircraft physics we must first have a basic understanding of the dynamics of air. This section touches briefly on the most important topics such as lift and drag and explains how heavier-than-air flight is possible. Most of the theory is explained for fixed-wing airplanes so a separate section discusses how the same concepts apply for helicopters.

Aerodynamics is a specialization of *fluid dynamics* as air and gases have the properties of fluids. When designing an aircraft it is important to consider how it interacts with air and there are two main types of forces in flight; namely *lift* and *drag*. Lift is the upwards force created by diverting an airflow over the wing that counteract gravity and enables the aircraft to maneuver. Drag is the air equivalent to ground friction when pulling a sled. They both resist motion but drag also increases its resistance as the airspeed increases whereas friction is nearly independent of velocity.

Beyond lift and drag there are also more advanced properties of aerodynamics such as turbulence, supersonic flight and different aircraft designs that are briefly discussed in this chapter.

2.1.1 Lift

The principle of generating lift is a complex physical phenomenon and historically there have been several explanations and theories. A currently popular explanation to lift is that a difference in pressure causes the wing to pull up. This idea originates from Bernoulli's Principle, which states that for an inviscid flow the pressure decreases with increasing speed[18]. The *equal transit time* condition proposes that diverted air must rejoin at the trailing edge of the wing in the same amount of time. The difference in pressure is then a result of air traveling faster on the top side of the wing because the top side is more curved and the air must travel farther. The condition has later been proven wrong in wind tunnels, but the popular understanding of lift still prevails because it "makes sense". Although the air actually do move faster along the top side and there is a pressure difference, this alone does not explain how lift is created and there are several sources that debunk the common understanding of lift [19, 4, 20, 21, 22].

A simplified but more accurate explanation to lift [1, pp. 62-65] is that masses of air are accelerated down in order to lift the body up by Newton's second and third laws. This explanation is also crude because it neglects the effects of viscosity and the fact that air particles behave more like a continuum than tiny bullets deflecting off the wing. A

thorough explanation to the lift of wings is found in [23, ch. 3] and is too comprehensive to cover here, so we will stick with the Newtonian explanation in this section.

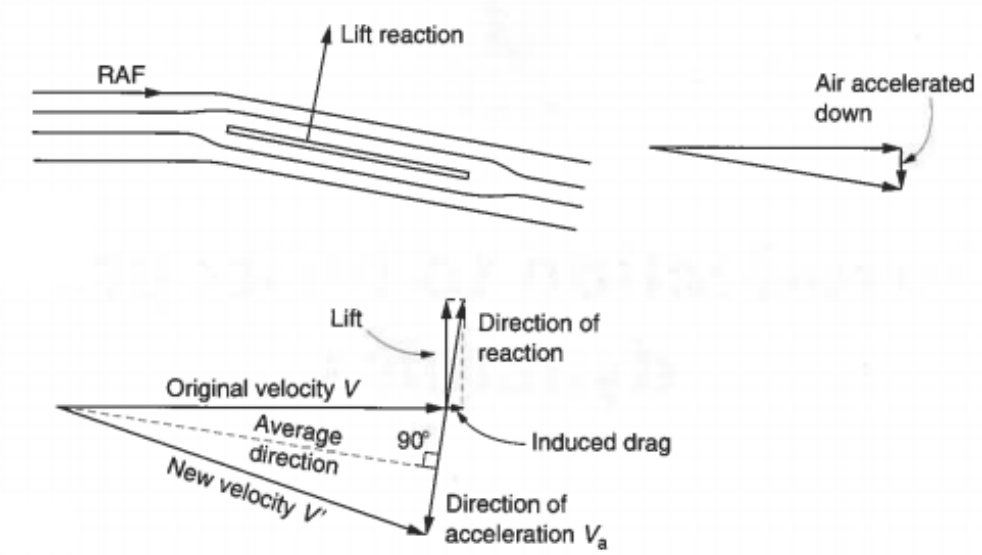


Figure 2.1: Relative airflow passing over an airfoil and generating lift. Source: [1, p. 62].

A structure placed in an airflow for the purpose of generating lift is called an *airfoil* and for airplanes and helicopters the airfoil is the wings and rotor blades respectively. The magnitude of the lift depends on the *relative airflow* (RAF). RAF has a direction and a velocity represented in figure 2.1 by vector V. When air passes around an airfoil the air is pushed down due to the *angle of attack* of the airfoil structure. We assume only the direction of the RAF is changed and not the velocity, so a new velocity vector V' is derived from V.

In order to accelerate the air in this manner the air is not accelerated straight down but at an angle V_a, as illustrated. This produces a reaction force that comprises the wanted upwards lift force and an unwanted induced drag force explained later. Intuitively the lift increases as the relative airspeed or the angle of attack increases and when the lifting force exceeds the weight of the body it makes heavier-than-air flight possible. For an airfoil where the lift coefficient for a specific angle of attack is known the lifting force can be determined[1] by equation 2.1 [7, p. 128].

[1]The lift equation only holds for specific flow conditions

$$F_L = \frac{1}{2}\rho u^2 A C_L \tag{2.1}$$

$$F_D = \frac{1}{2}\rho u^2 A C_D \tag{2.2}$$

ρ	mass density of the fluid
u	relative airflow velocity
A	reference area (typically the square of the mean chord length for a wing)
C_D	drag coefficient
C_L	lift coefficient

2.1.2 Drag

When an object moves in a fluid it is subjected to resistive drag forces. In aircraft designs this is something we want to minimize, and to better understand how we need to distinguish between different types of drag. According to [7, pp. 236-256] there are three major categories:

parasite drag from the pressure when an airflow is split around a structure

induced drag because the wing requires an angle of attack to generate lift

wave drag due to the formation of shock waves around the aircraft

2.1.2.1 Parasite Drag

Parasite drag covers all types of drag that are not related to the generation of lift. It occurs when a solid body is placed in an airflow forcing the flow to separate.

There are three main types of parasite drag and they all follow the drag equation 2.2 and rise by the square of the RAF velocity. Dependent on the form the body induces drag by the bluntness of the body and by air passing along its surface named *form drag* and *skin friction drag* respectively. Figure 2.2 and table 2.1 shows how the distribution generally varies for different forms. Intuitively it can be seen that form drag is the major contributor and that streamlined bodies with gradually changing cross-sections are preferred for flying structures.

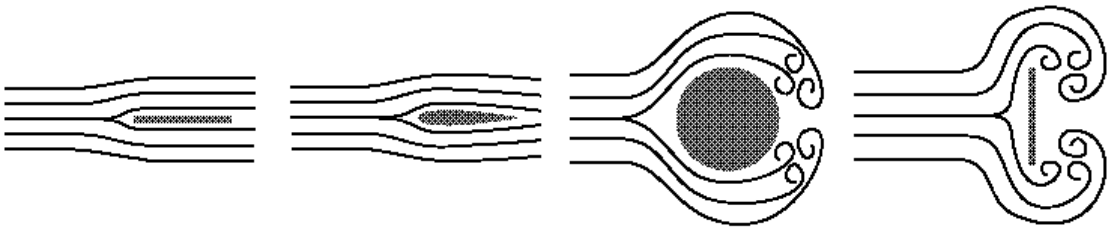


Figure 2.2: Form drag and skin friction for different types of forms. Source: [2]

Form drag	0%	10%	90%	100%
Skin friction	100%	90%	10%	0%

Table 2.1: Distribution of form and skin friction drag. Source: [2]

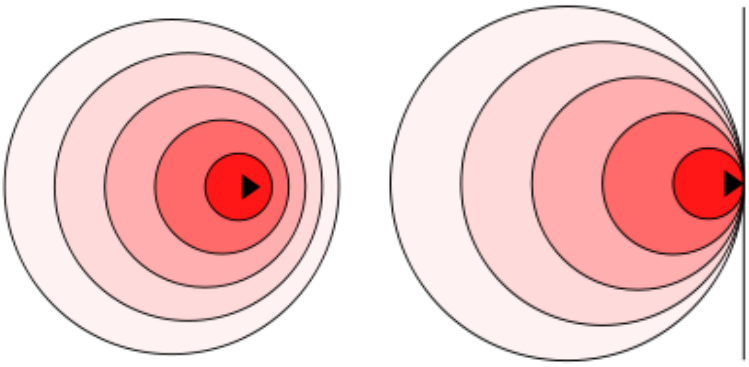
Finally there is *interference drag* induced by airflow vortices. Vortices arise when two surfaces meet at a sharp angle and draw energy from the aircraft in the form of drag. To minimize interference drag the designers try to smooth out all sharp angles by adding *fairings;* structures that gradually join each surface. Vortices also appear where separate bodies are close to each other such as between the wing and the engines.

2.1.2.2 Induced Drag

The effect of accelerating air downwards has the undesirable consequence of inducing drag due to the wing's angle of attack. Figure 2.1 shows how changing the RAF vector gives a reaction vector that is at an angle to the desired vertical lift direction. This angle comprises a component that is backwards and will try to slow the wing down proportional to the lift created. Generally the lift-induced drag increases when the RAF velocity or the angle of attack is increased.

2.1.2.3 Wave Drag

Aircrafts that are able to travel faster than the speed of sound perform what is widely known as *supersonic flight* and breaking the sound barrier. The *Mach number* [4, pp. 149-167] is an aircraft's relative airspeed in units of the speed of sound and at Mach 1 the aircraft is moving through air at the speed of sound. As the aircraft diverts air around its body it emits waves of sound (fig 2.3(a)) that travels for several miles in the air just as a big boat creates waves in the water. These waves draw kinetic energy from the aircraft in the form of drag and the faster the aircraft moves the more powerful the waves become.



(a) Subsonic flight emits ripples of sound waves

(b) The wake of Mach 1 accumulates into shock waves

Figure 2.3: Wakes of sound waves during flight. Source: [3]

When nearing Mach 1, known as *transonic flight*, there is a rapid increase in wave drag as shown by figure 2.4. Even if the aircraft is traveling below Mach 1 the air has to move around thick structures and can accelerate into supersonic speeds to keep up with the outer airflow. This results in small shock waves near excrescences in subsonic flight and is shown by the sudden increase in drag for Mach numbers between 0.85 and 1.

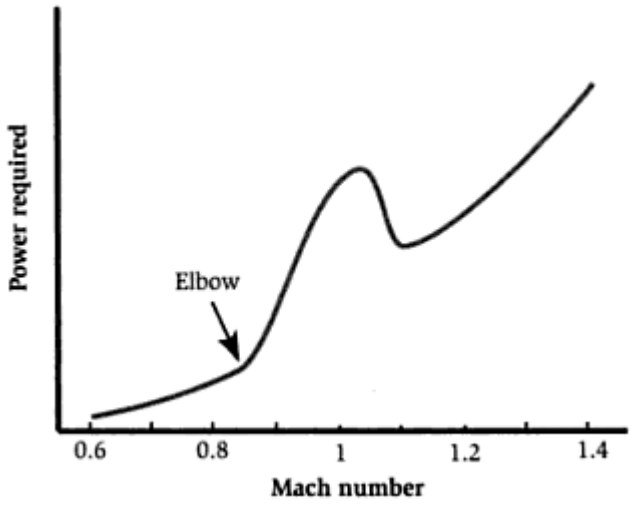


Figure 2.4: Required power increases dramatically as the airplane approaches Mach 1. Source: [4, p. 161]

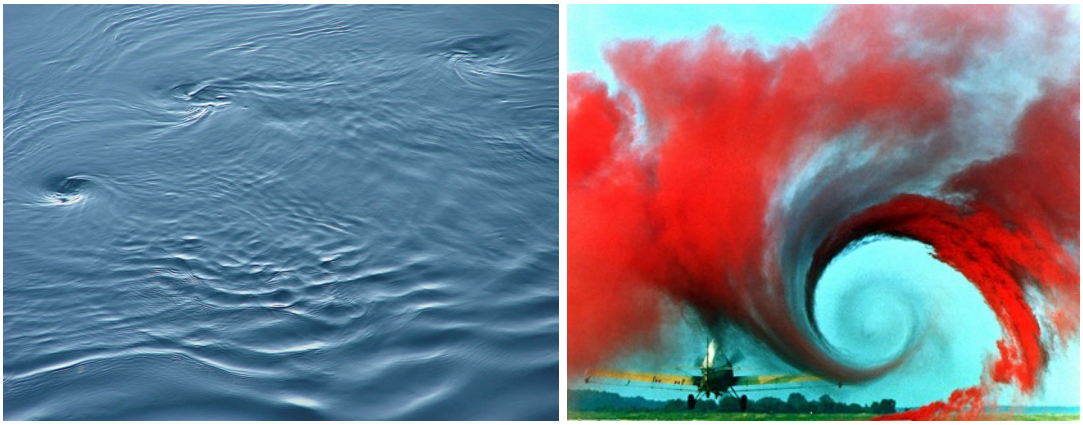
The waves in front of the aircraft travel by the speed of sound and if the aircraft is moving at Mach 1 they cannot get out of the way. The sound waves then merge into extremely powerful shock waves that sound very much like an explosion on the ground.

These *sonic booms* are so powerful that supersonic flight is often banned over populated areas.

2.1.3 Turbulence

One of the truly complex behaviors of fluid dynamics is turbulence, and this stochastic chaos is an area where theory lags behind experiment. Although turbulence is widely studied in many fields, such as in the natural mixture of fluids and in the flow of water pipes, we will only discuss here what is relevant for aircrafts.

In flight this phenomenon is commonly known as *air pockets* and rough rides through windy conditions, but there is more to turbulence than most people think. For instance it plays a crucial role in creating lift that is not too intuitive. According to [24, pp. 138-141] there are two types of fluid flows; laminar and turbulent, where the first is relatively smooth and parallel and the other becomes sinuous and eddying.



(a) Eddies in the water (b) Eddies in the air

Figure 2.5: Turbulent flow can result in eddies. Sources: [5], [6]

Consider moving the oar of a row boat slowly through the water. The water is not disturbed and flows smoothly around it in what is called a *laminar flow*. Then force the oar quickly through the water. The water separates so fast that nearby water particles are not able to immediately fill the void resulting in a sinuous motion of water and eddies in the trail known as *turbulent flow.*

When flying a commercial airliner the disturbance we may experience is due to *atmospheric turbulence.* This generally arises from two conditions:

1. Shear wind forces when the wind velocity varies at different positions
2. *Thermal convection* when warm air rises and cooler air descends

Both sources are generated in the lower 1-2 km of the atmosphere known as the *boundary layer.* Shear forces occur when air closer to the ground is retarded by friction from

structures such as trees, buildings and mountains. The latter is a natural consequence of the ground being heated by sun radiation forming vertical structures of rising air known as *thermals.*

2.1.3.1 Reynolds Number

An important parameter of fluid dynamics is the Reynolds Number. It is simply the ratio of *inertial force*s (resistance to change in motion) to the *viscous forces* (stickiness) of a fluid. The number is dimensionless and indicates how much the *viscosity* affects the flow of the fluid at a given mean flow velocity V and a reference dimension L.

There is no theorem relating the Reynolds Number magnitude to turbulent flow, but [25] describes how high numbers in the order of tens of millions for the wing of an airplane indicate a nearly inviscid behavior and can result in a turbulent flow of air around the wing. For Reynolds numbers below a few thousands the viscous forces are more significant and exhibit laminar flows, such as for the flow of blood particles in the veins. However, the significance of the viscid forces must be considered per application as the flow will have varying degrees of turbulence depending on the properties of the fluid and any obstructions in the flow. The water flow in a straight pipe is very different than for the flow of an angled pipe or for the currents of the Gulf Stream.

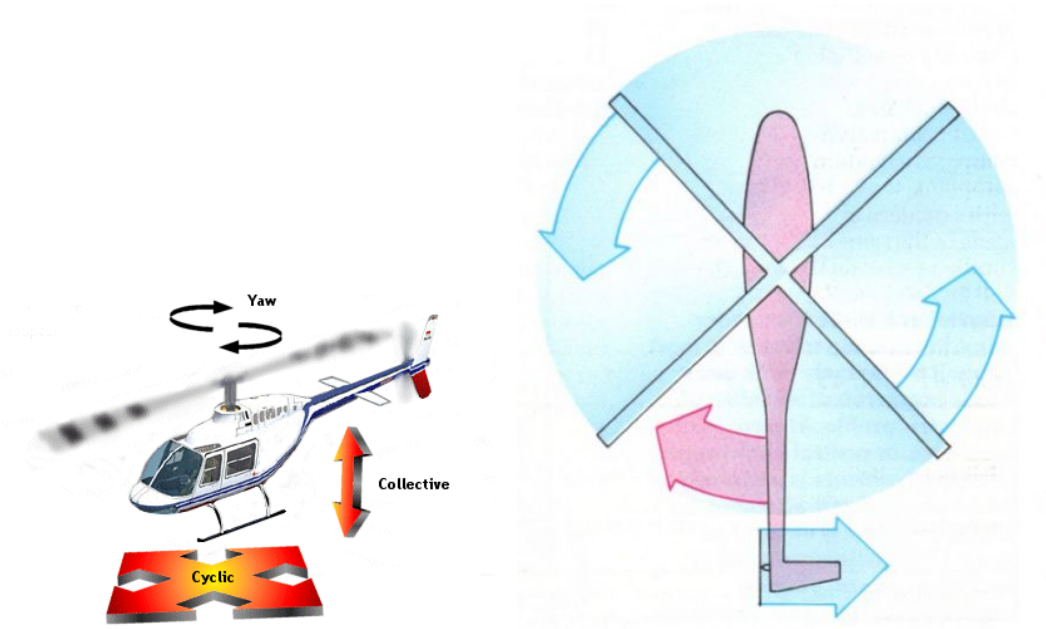
In aircraft design analysis the Reynolds Number is used to calculate the drag and lift coefficients, and when testing the design in a wind tunnel it is common to use a model-scale prototype. Air performs differently on smaller objects and so it is necessary to scale the flow as well. Accurate wind tunnel testing can be achieved by matching the Reynolds Number for the flow at the full and model scale objects. The Reynolds Number is given by equation 2.3 and is a function of fluid density, viscosity, relative flow velocity and a reference dimension. For airplanes the wing chord is often chosen as the reference length L so in order to match the two flow regimes the wind tunnel airspeed *or* the air density is increased for tests on the model-scale aircraft.

$$Re = \frac{\rho V L}{\mu} \tag{2.3}$$

ρ	fluid density
V	mean relative flow velocity
L	reference length (typically the chord length for wings)
μ	dynamic viscosity

2.1.4 Rotary-Wing Aerodynamics

Most of the aerodynamics so far have been focused on airplane analogies where fixed-wing structures are subjected to lift and drag. Airplanes are relatively simple in nature and make complex aerodynamic behavior easier to understand. Another important type of flight is by rotary-wing designs such as helicopters and this section will briefly discuss the key properties. A complete description of their workings is found in [8].



(a) The controls of a helicopter (b) Tail rotor compensates for main rotor torque

Figure 2.6: Basic helicopter dynamics

While airplanes move at high speeds to generate sufficient airflow and lift the rotor blades are instead spun at high speeds around an axis of the helicopter. The rotor blades are formed like small wings and typically one or two pairs of blades are used to generate lift. The dynamics of helicopters are far more complex than for airplanes so we will break it down into basic maneuvers.

2.1.4.1 Hovering

The main advantage of helicopters is the ability to move in all three axes. This allows for vertical take-off and landing and hovering at a fixed position in the air that allows the helicopter to perform many tasks that airplanes are not able to. The challenge in hovering is that wind, turbulent air and mechanical imperfections will eventually tilt the helicopter and accelerate it away from the position. If the drift is not immediately countered by the pilot the helicopter will continue to tilt, accelerate even more and speed up dramatically. Hovering may be compared to balancing on a ball and a stable hover takes a lot of flight experience and is difficult even in calm conditions.

2.1.4.2 Collective Pitch

There are two ways to control the induced lift shown in figure 2.6(a). First the thrust can be increased to spin the rotor blades faster and second the angle of attack of the blades can be increased.

Thrust-controlled lift is considered simpler and cheaper, but suffers from slow and disproportionate lift response because of varying resistive forces such as drag and mechanical friction and because the inertia of the engine resists change in motion.

Collective pitch instead changes the angle of attack of the blades to control lift. The rotor blade inertia helps improve the proportionality and the response time of the control output and is one of the reasons why most modern helicopters use collective pitch. In addition it enables the pilot to perform emergency landings if the engine should stop working. *Autorotation* is a technique that exploits the inertia of the rotor blades and the potential energy at an altitude to generate enough lift to land safely.

2.1.4.3 Cyclic Control

One part of flying a helicopter is using lift to control the altitude and the other is using lift to move around. Since the main rotor is the only source of lift, moving from A to B is simply a manner of tilting the helicopter in a direction so that part of the lift is used to counter gravity and the other part is used to accelerate towards B. Tilting is achieved by *cyclic control* and enables the helicopter to move horizontally as in figure 2.6(a). A complex mechanical structure allows the angle of attack to be controlled separately for each rotor blade. By producing a greater angle of attack on the left side looking in the nose direction each blade will produce more lift on the left side and cause the helicopter to tilt to the right. Tilting to the left or right and tilting the nose up or down is called changing the *roll* and *pitch* angles respectively. The helicopter may tilt in any combination of pitch and roll and enables the helicopter to perform very advanced maneuvers.

2.1.4.4 Tail Rotor

In flight the engine must constantly apply torque to the main rotor shaft to overcome the resistive forces. The consequence is that without a fixed point to hold on to the helicopter will spin in the opposite direction [1, pp. 166-177].

To counter this effect the conventional helicopter uses a smaller tail rotor that produce thrust in the opposite rotational direction of the main rotor, illustrated by figure 2.6(b). The tail rotor is positioned at a distance $\mathbf{d}$ from the main rotor axis so that its thrust $\mathbf{F}$ can be relatively small to compensate for the main rotor torque, as given by the torque equation $\tau = \mathbf{F} \times \mathbf{d}$.

In addition to counter torque the tail rotor enables the pilot to control the *yaw* angle and point the nose in an arbitrary direction.

2.2 Methods of Flight Simulation

Predicting and simulating aircraft behavior are two strongly related fields of theory so this chapter covers methods for predicting the dynamics of flight. As already noted, the theory in this field is still young and often combined with empirical approaches. This chapter will not go into all the details but rather give an overview over methods that are widely used when predicting the aerodynamic properties of an aircraft structure.

2.2.1 Parametric Equations

We can approximate drag and lift for most parts of the aircraft by transforming their geometrical structures to similar structures that have empirically derived tables. In this section we discuss two methods to estimate the parasite drag for wing-like and body-like parts of the aircraft. Equivalent methods also exist for lift and a number of other drag components, but since they are similar in concept we omit them here and refer to [7] for further details. It should be noted that the estimations are crude and are typically corrected by empirical data from previous wind tunnel test data of similar designs.

One approach [7, pp. 229-256] is to calculate the parasite drag coefficient separately for the parts of an aircraft and sum the total drag coefficient. The following equation 2.4 divides the aircraft in N parts and each part has a form factor K, skin-friction coefficient $\overline{C}_f$ and a wetted surface S_{wet} susceptible to airflow. The reference area S_{ref} can be any surface but is typically chosen as the top surface of the wing.

$$C_D \equiv \frac{\sum_{i=1}^{N} K_i \overline{C}_{f_i} S_{wet_i}}{S_{ref}} \tag{2.4}$$

The parasite drag forces in subsonic speeds are mainly due to form and skin friction, but since parts of the aircraft are very different we distinguish between wing-like structures and body-like structures. This section will cover two methods for determining K_i, $\overline{C}_{f_i}$ and $S_{wet_i}$for the two types of structures so that a total parasite drag coefficient can be estimated.

2.2.1.1 Wing method

For thin wing-like structures skin friction is the major contributor to drag. Equation 2.5 was derived from theory and experiment by Prandtl-Schlichting [7, pp. 196-200] for the skin friction of turbulent flow over flat plates.

$$\overline{C}_f \equiv \frac{0.455}{(log_{10} Re_L)^{2.58}} \tag{2.5}$$

To apply this equation we must first fit the surface of a wing to a flat plate equivalent. A wing is not flat and is typically trapezoidal with varying *chord* lengths as shown in figure 2.7. One solution is to find the mean aerodynamic chord (MAC) length by geometry and define the wing as a rectangle of area $MAC \times wing\,span$.

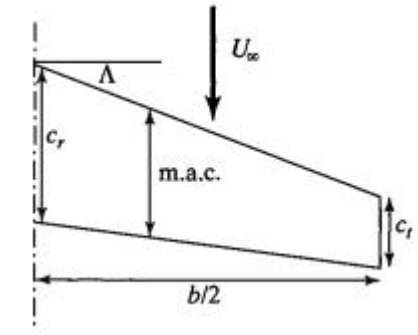


Figure 2.7: The mean aerodynamic chord (MAC) of a wing. Source: [7]

The wing surface S_{top} has a curvature due to its thickness and is flattened by a thickness factor $(1+0.2\frac{t}{c})$. Since the total wing surface comprises a top and bottom side the top surface is doubled. S_{wet} is then the total wetted surface of the wing transformed into a flat plate equivalent.

$$S_{wet} \approx 2S_{top}(1 + 0.2\frac{t}{c}) \tag{2.6}$$

S_{wet} flate plate equivalent of surface exposed to airflow

S_{top} orthogonal top surface area of wing

$\frac{t}{c}$ ratio of thickness to the chord length of the wing

At last the form factor of the wing must be determined. As previously mentioned the form factor affects the distribution of pressure and skin friction. The form factor K can be determined from tables of empirical data and is a function of thickness to length ratio and the angle at which the wings are swept back. With all the parameters known the parasite drag of a wing-like component can now be calculated by equation 2.7.

$$C_{D_{wing}} = K\overline{C}_f\frac{S_{wet}}{S_{ref}} \tag{2.7}$$

$C_{D_{wing}}$ parasite drag coefficient for a wing-like body

$\overline{C}_f$ skin friction for a flat plate equivalent

K form factor of wing

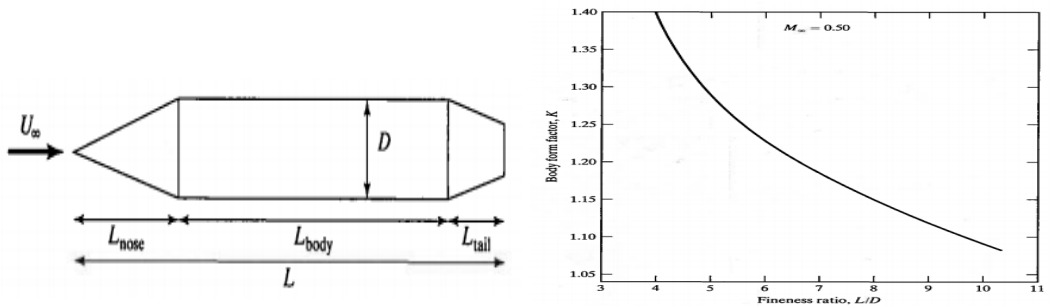
Re_L mean Reynolds Number for a wing of MAC length L (by equation 2.3)

S_{ref} reference surface area (typically S_{top})

2.2.1.2 Fuselage method

The fuselage of an aircraft has a constant chord length and is much simpler to calculate than wings. The drag coefficient is found by the same equation 2.7 but the parameters are slightly different. The wetted surface S_{wet} can now be approximated by cones, cylinders and other well known geometries as shown in figure 2.8(a). The form factor K is determined by the *fineness ratio* of the body, defined as the length divided by the maximum diameter (L/D). Figure 2.8(b) is derived from experiment and shows how the form factor decreases as the fuselage becomes longer and more slender.

Finally, the flat plate skin friction $\overline{C}_f$ is found by equation 2.5. Here the Reynolds Number at the end of the fuselage is used, given by equation 2.3 and the fuselage length L from figure 2.8(a) as the reference length.



(a) Approximating the surface by geometry (b) Form factor as a function of fuselage fineness

Figure 2.8: Drag coefficient parameters using the fuselage method. Source: [7, p. 251]

2.2.2 Computational Fluid Dynamics

The problem with parametric equations is that some behavior is hard to express by parameters and the methods generally suffer from crude approximations mapped to empirical data. A different approach is to simulate the behavior of air and calculate the forces that air exert on the aircraft. The computation of fluid dynamics is a twofold process [26, ch. 1]. First a physical model must be chosen to approximate real air, defined by governing flow equations. The models have trade-offs in performance and correctness and the choice should reflect the requirements of the application. Second a numerical modeling method transforms the physical model to a discrete domain that can be solved on a computer. Numerical methods also have trade-offs and varies in ease of implementation, performance and accuracy, but all methods suffer from numerical dissipation.

As computational fluid dynamics (CFD) gained interest in the 80s it has been touted as the Holy Grail for aerodynamics analysts. A dramatic increase in computational power and memory capacity made this approach economically feasible and the accuracy is ever increasing. However, [26, ch. 1] notes that the simulation is no substitute for experiment as the results are yet impossible to validate without careful testing in equal

conditions.

The current purpose of CFD is to give insight into complex behavior and is often used to speed up design processes by simulating the effects of changes before a prototype is built and tested. In the perspective of a modern flight simulator this technology can be used to simulate very complex behavior such as winds, turbulence, interfering airwakes and the effects of modifying an aircraft design.

2.2.2.1 Governing Flow Equations

The governing equations of fluid flow are mathematical expressions of physical conservation laws. A conservation law states that a particular measurable property in an isolated physical system does not change over time. When modeling fluid motion there are three laws discussed in [26, ch. 2]:

1. The mass of a fluid is conserved
2. The rate of change of momentum for a fluid particle equals the sum of its forces by Newton's second law
3. The rate of change of energy equals the sum of the rate of heat addition to and the rate of work done on a fluid particle by Newton's law of thermodynamics

For practical reasons the fluid may be treated as a continuum with macroscopic properties such as velocity, pressure, density and temperature. The molecular structures and motions only complicate matters. The Navier-Stokes equations are currently considered one of the most physical correct descriptions of fluid dynamics but they have proven very hard to solve for. A hierarchy of governing equations has later been proposed that simplify the model but still preserve important fluid behavior while being easier to compute. This section briefly covers three promising flow equations described in detail in [8, ch. 14].

2.2.2.2 Navier-Stokes Equations

The most fundamental equations governing the fluid dynamics were found independently by Navier and Stokes and originate from the conservation of mass and the interchange of momentum and energy within a fluid. In [8, ch. 14] this is written in conservation form as equation 2.8, where Q is the conserved variable vector and E, F and G are *flux vectors* that gives the rate at which mass, momentum and energy are being transported at any point in the fluid. The right hand terms E_v, F_v and G_v express the fluxes resulting from the viscosity of the flow.

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = \frac{\partial \mathbf{E_v}}{\partial x} + \frac{\partial \mathbf{F_v}}{\partial y} + \frac{\partial \mathbf{G_v}}{\partial z} \tag{2.8}$$

The Navier-Stokes (N-S) equations are strongly coupled, non-linear partial differential equations and are extremely difficult and expensive to solve without major simplifications, such as assuming *thermal equilibrium*, *ideal gasses* and incompressible fluids.

A key issue with any CFD calculation is the generation of grids on which to solve the governing equations. Turbulence and viscous shearing effects require an extremely fine grid resolution and large amounts of memory and processing power to compute.

Another problem is that theory lacks closure on the governing equations, meaning there are too few equations and too many unknowns. In particular the *turbulence closure problem* has proven difficult. Empirical closure models were introduced to overcome this and had success for simple cases like turbulent flow in a straight pipe, but history has proven it difficult to find a general model that fits different and more complicated cases.

2.2.2.3 Euler Equations

If it can be assumed that viscous forces do not need to be resolved and the fluid is treated as inviscid, then according to [8, ch. 14] the N-S equations reduce to the Euler equations shown by dropping the viscous terms of equation 2.8 to yield equation 2.9.

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = 0 \tag{2.9}$$

In theory the Euler equations should not be able to model turbulence since fluids would never circulate were it not for viscous forces. Although it should be noted that too much viscosity would also prevent the formation of vortices since they would be damped by internal resistance in the same way as syrup does not form eddies. However, in practice the Euler equations yield a very good approximation to the airflows around an aircraft since the flows are nearly inviscid. Numerical methods suffer from numerical dissipation and this artificial dissipation will to some extent make up for the missing viscid terms and allow vortices to form. Since the Euler equations are far simpler to solve for than N-S this is a very desirable property.

2.2.2.4 Vorticity Transport Equations

No matter what governing equations are used there is a separate issue when modeling turbulence. For practical use it is often necessary to consider how turbulent flows indirectly affect the aircraft. For instance a helicopter that hovers near the ground will disrupt the air around it by the wake of turbulent flows from the rotor itself.

In order to model how this turbulence is recycled back into the in-flow it is necessary to preserve the detailed vortices over time and over a large amount of space around the helicopter. It would be infeasible to have sufficiently detailed grids for the entire space and so there is need for a separate set of governing equations that model the vortices directly. One well known technique is the *Vorticity Transport Model* (VTM).

Figure 2.9: Vorticity in the rotor wake of a helicopter modeled by VTM. Note how the grid resolution is detailed near the rotors and gradually less detailed further away. Source: [8, p. 779]

If the flow can be assumed to be incompressible, which is reasonable for the case of a hovering helicopter where RAF velocities are relatively small, then [8, ch. 14] describes how VTM can be used to minimize the unwanted numerical diffusion of vorticity over space and time. This is done by solving the governing equations for the flow directly in terms of vorticity ω and velocity V as shown in equation 2.10.

The advection term refers to the transport of vorticity due to the fluid flow, strain is the deformation of the fluid's particle structure and diffusion is how the vorticity is dampened by viscosity v.

$$\underbrace{\frac{\partial w}{\partial t}}_{temporal} + \underbrace{(\mathbf{V}\cdot\nabla)\omega}_{advection} = \underbrace{(\omega\cdot\nabla)\mathbf{V}}_{strain} + \underbrace{v\triangle\cdot\omega}_{diffusion} \tag{2.10}$$

If the vorticity field is known or initially assumed then the velocity field can be calculated by the Biot-Savart relationship 2.11 and applied to a traditional CFD grid approach. For practical use the field may be easier to calculate in integral form as shown in 2.12.

$$\nabla^2\mathbf{V} = -\nabla\times\omega \tag{2.11}$$

$$\mathbf{V}(x) = -\frac{1}{4\pi}\int\frac{(x-y)}{|x-y|^3}\times\omega(y)dy \tag{2.12}$$

VTM is an intermediate between traditional CFD techniques and pure vortex methods and provides a very complete model for the evolution of the airflow structure in the wake, illustrated by figure 2.9. However, as with other CFD approaches it suffers from being computationally expensive and is still impractical for routine use, but provides excellent insight into the complex turbulent flows and how they affect aircraft flight.

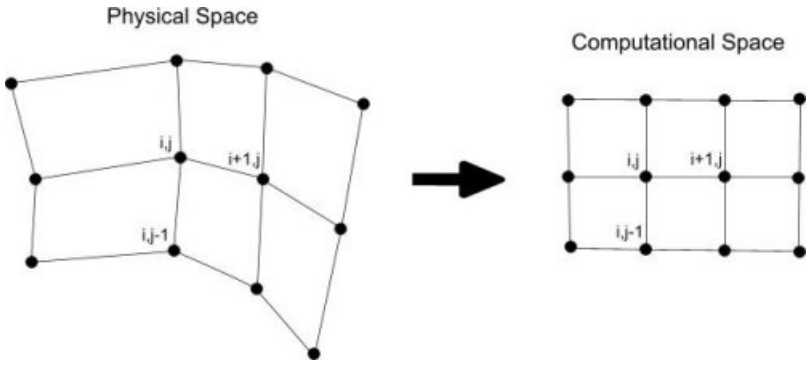


Figure 2.10: Structured grid. Source: [9]

2.2.3 Numerical Modeling

In practical use there is a need for numerical methods to calculate the flow state using any of the governing equations above. The equations only describe relations of physical properties in the flow, but to compute a result we must discretize time, space and their derivatives.

According to [26, ch. 4] there are three widely used methods that solve an approximation of the problem; *finite element, finite difference* and *finite volume*. The three methods take different mathematical approaches to approximating and discretizing flow variables, but rely on the same basic steps. First a grid must be generated for the area of interest, then time, space and their derivatives must be discretized and solved for the governing equations by the values in the grid.

Grid generation has already been mentioned and is an essential part of CFD techniques. Except for simple cases it is extremely difficult to analyze fluid flows due to complex and interchanging partial differential equations. To overcome this the flow domains are subdivided into smaller domains and solved independently. The grid should encompass all areas where we are interested in calculating the flow and a straight-forward solution is to use *structured grids*. Structured does not refer to the geometrical form but rather how grid information is accessed by the computer, as shown in figure 2.10 by a mesh with rows and columns. The neighbors are then easily found and simplify the CFD code.

The problem with structured grids is that we can't refine the resolution at certain areas in the grid without increasing the entire grid resolution, and that is likely to become a bottleneck in applications with complex geometries in the flow. An alternative is to use *unstructured grids* that may consist of tetrahedrons, hexahedrons or prism cell shapes as shown in figure 2.11. These geometries allow local grid refinement but are also harder to implement solvers for. In theory any type of complex geometry can be properly wrapped inside a flow grid but the process of creating the grid is so complicated that it often requires human interaction to achieve a good balance between efficiency and accuracy.

Two techniques that are used to automatically improve the CFD grid are described

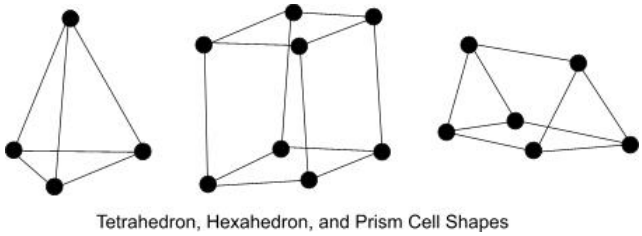


Figure 2.11: 3D cell shapes in unstructured grids. Source: [9]

in [26, ch. 1] as *convergence* and *grid independence*. Convergence denotes how quickly a solution converges to within an accepted error margin in a solution algorithm that is iterative in its nature. The convergence can be adjusted using relaxation and acceleration methods in much the same way the stiffness of a spring can be adjusted. Soft springs produce inaccurate results while stiff springs may take excessive computational time to avoid numerical instability.

Unstructured grids are particularly difficult to balance between good results and cost in areas with complex flow patterns. Grid independence refers to when a grid has a resolution that produces sufficiently accurate results and that enhancing the grid resolution further will not improve the results significantly.

Evolutionary algorithms can be useful to refine the grids more quickly. Convergence and grid independence can be used as heuristics to help grid generators produce both efficient and accurate grids given user-defined constraints.

2.3 Methods of Visualization

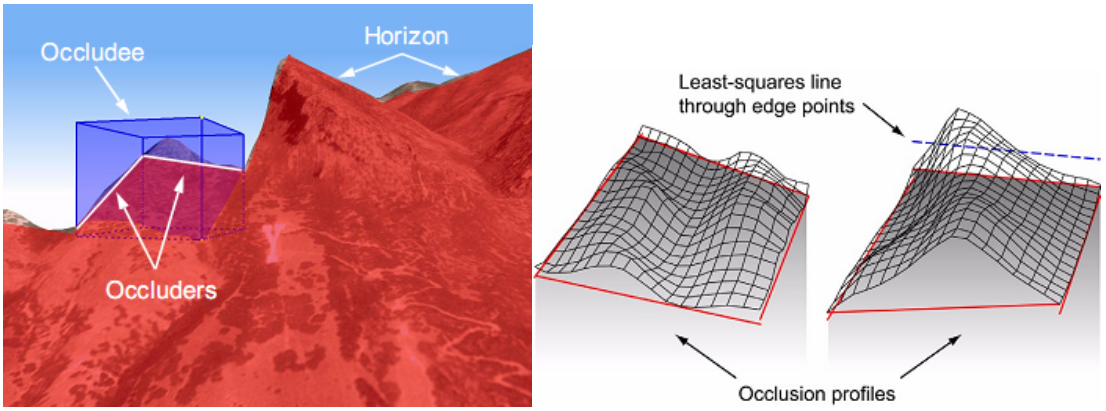
An important part of the flight simulator is to give a realistic and appealing representation of the virtual world. Outdoor environments have proven particularly difficult to render quickly and game development has been a key driver here. Games are renowned for their endless struggle to squeeze a few extra frames per second out of the hardware, and this has led to many creative solutions using approximations and a sleeve full of dirty tricks. This section discusses some essential issues of rendering natural scenes and describes fast methods that cleverly approximates their appearance.

2.3.1 Terrain

The virtual world of a flight simulator consists in large part of ground that often stretches over long distances. It would be desirable to give the user an impression of being able to fly as far as the eye can see and still retain a high level of detail.

2.3.1.1 Horizon Culling

One big concern with rendering terrain in real-time is how to maintain highly detailed variations over large areas, since the graphics card memory is limited. Even if the terrain data is small enough to be rendered directly the large amount of polygons often become a rendering bottleneck and limits the frame rate. To overcome this problem it makes sense to only render the parts of the terrain that is actually visible and to reduce the level of detail without degrading the image quality.



(a) Occluders and occludees (b) Approximation of gridlet by a quad

Figure 2.12: The horizon edge is used to cull patches of terrain. Source: [10]

Horizon culling [10] is a method that quickly determines if geometry is visible from a point of view by keeping track of the highest terrain points in the screen space projection. The method relies on a rendering algorithm that renders the patches of terrain, or

gridlets, in a front-to-back manner. This way the horizon can be updated for each rendered gridlet and all subsequent renders need only test against the current horizon.

Obviously the tests against the horizon are performed frequently and so the paper proposes an optimization by *occluders* and *occludees* as figure 2.12(a) shows. When a gridlet is tested against the current horizon, it is conservatively approximated by a quad polygon that guarantees to be lower or equal to the terrain points along the edges of the gridlet, shown by figure 2.12(b). Then this quad is tested for visibility against the screen space horizon edge and if any part of it exceeds the horizon then the gridlet is rendered and the horizon edge updated.

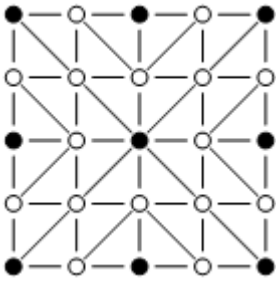
The paper also proposes that this method be used in combination with frustum culling, where gridlets that are fully outside the view space can be omitted. The horizon culling is then simply applied to further refine the culling of terrain patches that are guaranteed to be fully occluded.

In a particular scenario where the camera flew across a terrain with a variety of views the paper claimed to increase the frame rate by 2-4 times compared to only using frustum culling. This makes it an interesting method for rendering large terrains in a real-time flight simulator.

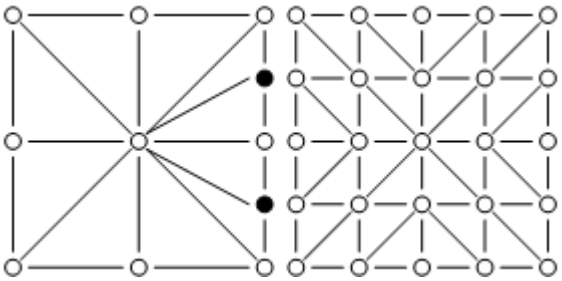
2.3.1.2 Big Textures and Geometric Level of Detail

Another concern is that the level of detail (LOD) in terrains can be extremely high and the textures just as large. Fortunately, the final image quality only require a certain LOD at a distance and a method is proposed in [11] that handles both geometric mipmapping and very large textures.

Modern graphics cards are designed for games and are currently limited to textures of about 4096×4096. For a terrain of 100km^2 a texture of that size would only hold a texel per 2-3 meters and would be insufficient by far for details up close.



(a) Level of detail is achieved by omitting the white points



(b) The edges between patches of different LOD are matched

Figure 2.13: Geometric level of detail for patches of terrain. Source: [11]

The method divides the terrain into a 2D array of patches called GeoMipMaps (GMM), which holds information about terrain height points and level of detail for that patch. The large texture is then simply split up in tiles small enough to render and the tiles mapped onto their respective GMMs.

The advantage of this structure is that each GMM can easily change its LOD by omitting n height points in both directions and conforming the shared edges of GMMs with different LOD, as shown by figures 2.13(a) and 2.13(b). A maximum geometrical error can then be pre-calculated for each LOD level, so that at run-time the minimum LOD for a GMM is trivially queried from the current view parameters and a user-defined threshold.

The concerns of geometric LOD and mapping tiles of large textures have much in common and this approach handles both. Its implementation is manageable and the author claims good performance, but the paper does not specify how well the method of geometric LOD performs for differently sized terrains compared to rendering with full LOD.

2.3.2 Vegetation

Grass and trees exist in abundance on Earth and is an important element in natural 3D scenes, but the complex geometries pose a huge challenge. It has proven difficult to get realistic looking vegetation at interactive frame rates so many tricks and approximations are often used. This section discusses a few recent methods that aim to produce believable pictures of vegetation at very high speeds on modern graphics cards.

2.3.2.1 Dynamically Lit Grass

One paper [12] proposes a method that allows dynamic lighting and a prominent parallax effect of grass in real-time. It uses a combination of geometric, volumetric and surface rendering to obtain highly detailed grass near the camera and a smooth transition to distances where grass blades can no longer be distinguished, shown by figure 2.14.

The method makes use of Bidirectional Texture Functions (BTFs) to pre-calculate per-pixel lighting of grass at a distance for different view and lighting angles and enables fast, dynamic lighting of non-geometric grass. It also supports density texture maps that enables artistic possibilities in 3D scenes as well as improves the transition between geometric and non-geometric grass. Lastly, the method supports shadowing and the grass blades can cast shadows onto the ground and each other. For non-geometric grass the shadows are baked into the BTFs and allow for truly realistic lighting and natural looking grass at all distances.

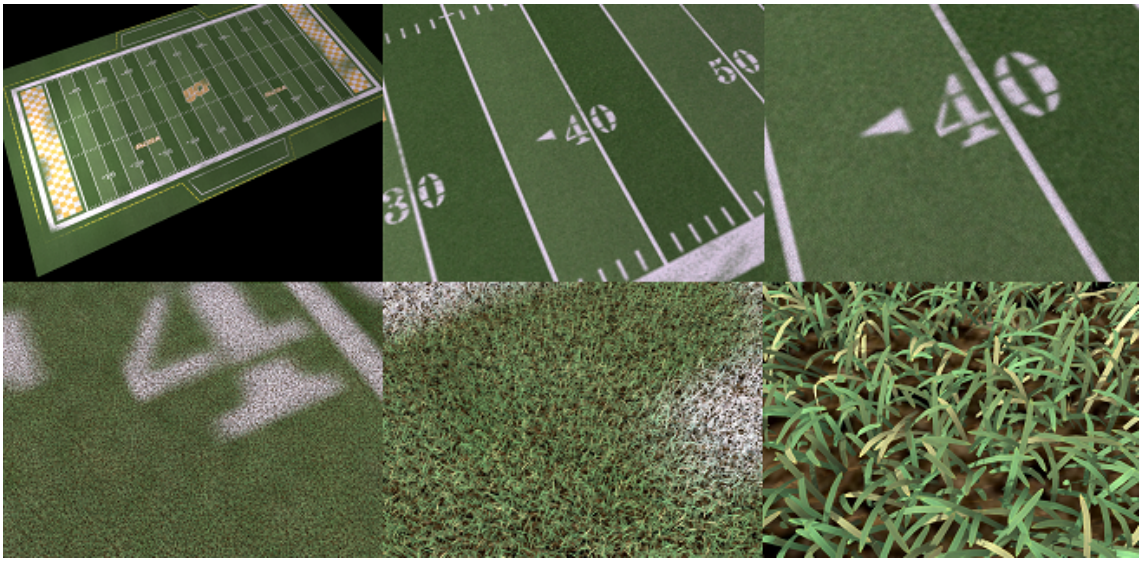
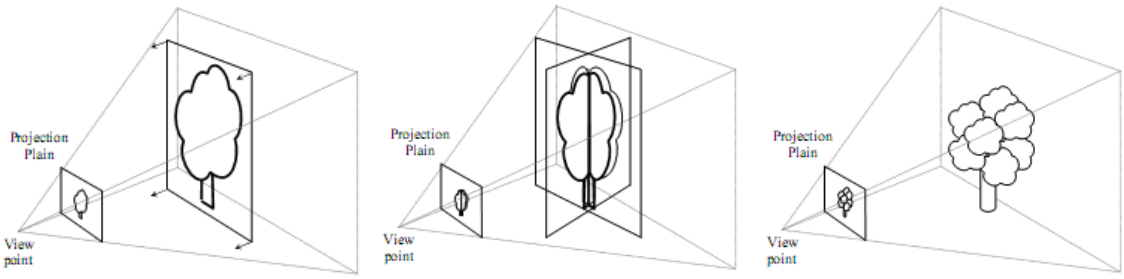


Figure 2.14: A football field rendered with dynamically lit grass at various distances, using a combination of geometric, volumetric and surface rendering. Source: [12]

2.3.2.2 Hardware Rendered Foliage

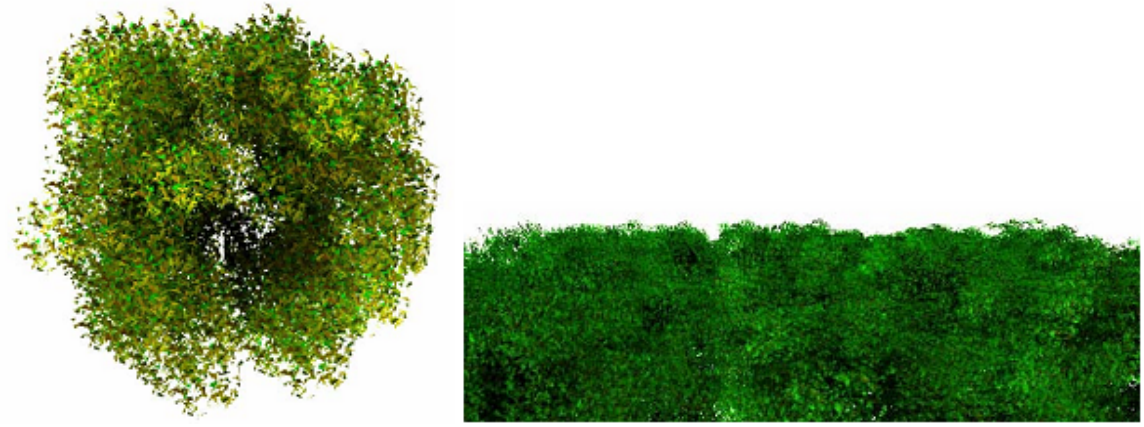
The method presented in [13] takes a novel approach to rendering trees and foliage quickly, using an extension of the well-known billboarding technique.

The conventional billboard tree is a transparent 2D texture that always faces the camera or a combination of multiple textures for different angles of the tree, as in figures 2.15(a) and 2.15(b) respectively. The main limitation with static billboards is a loss of motion parallax effect within the tree itself, so that the trees look very much like sheets of paper instead of geometric trees with depth.



(a) Always facing the camera (b) Two-faced cutout (c) 2.5 dimensional impostors

Figure 2.15: Comparison of billboarding techniques. Source: [13]



(a) Tree canopy (b) Forest of tree canopies

Figure 2.16: Final results using 2.5 dimensional impostors. Source: [13]

This algorithm introduces the concept of *dynamic 2.5 dimensional impostors* to visualize trees in a convincing quality while being fast and cheap on resources. Each frame a set of impostors are rendered at various sides of the tree. An impostor is simply a billboard texture rendered from a set of randomly placed geometric leaves, and because they are rendered each frame the impostors become view-dependent giving both depth and a natural parallax effect. The impostors are then duplicated as sprites for each their respective sides and make it look like the tree is rendered with thousands of geometric leaves instead of billboards of leaves.

The final touch of the method is to use the alpha channel of the impostor textures for depth information of each leaf within the rendered impostor volume, giving a semi-3D rendering of the leaves by standard depth-testing. This means branches and other geometries are successfully surrounded by leaves and the leaves themselves occlude each other properly.

2.3.3 Weather

Virtual worlds that require a realistic outdoor environment need to visualize the weather in one way or another. For sunny or cloudy days it is common in real-time applications to simply use a textured skydome with appropriate lighting, but if harsh weather conditions are simulated or visibility is reduced by fog it may be necessary to use additional methods.

One method presented in [14] describes how to efficiently render falling rain or snow when the camera is moving. The cost of animating and rendering such a vast number of particles is not practical in real-time, so an approximation to the same visual effect is proposed. By mapping textures of rain or snow streaks onto an invisible double cone, as shown in figures 2.17(a) and 2.17(b), simple texture transformations can animate the streaks along the cone and around the camera. The end result is that particles of rain or snow seem to fall continuously and by tilting the cone it will look like the camera is moving through the falling particles as they fall towards the camera.

(a) Cone placed around camera (b) Rain texture (c) Snow seems to be falling

Figure 2.17: An illusion of falling snow or rain by moving textures. Source: [14]

2.3.4 Perception of Depth

One of the challenges in computer graphics is to overcome the lack of depth on computer monitors and it can be hard to precisely determine the position of objects in a virtual world.

A perception of depth is particularly important when flying near the ground. The pilot needs visual references such as trees, stones, buildings and other types of objects whose relative position and general size can be ascertained. Good references help the pilot estimate his position and velocity and without a good perception of depth this task is complicated.

2.3.4.1 Stereoscopy

Stereoscopy is an effective technique that exploits our natural stereo vision. There are many variations in stereoscopy, ranging from simple stereo cards of two photographs to high-tech head mounted displays and immersive stereo-projected caves shown in figures 2.18 and 2.19. However, the principle remains the same. Stereoscopic pictures let each eye see only one of two pictures, and when the pictures are taken at angles that correspond to each eye's position the brain interprets a perception of depth as we do in the real world.

The advantage of stereoscopy is that it provides a true perception of depth so the position and scale of objects can be more accurately perceived. For aircraft simulation this would greatly increase the pilot's feel of the aircraft and is particularly helpful when navigating near the ground or near obstacles to determine distances and the scale of things.

Figure 2.18: Head mounted display with tracking. Source: [15]

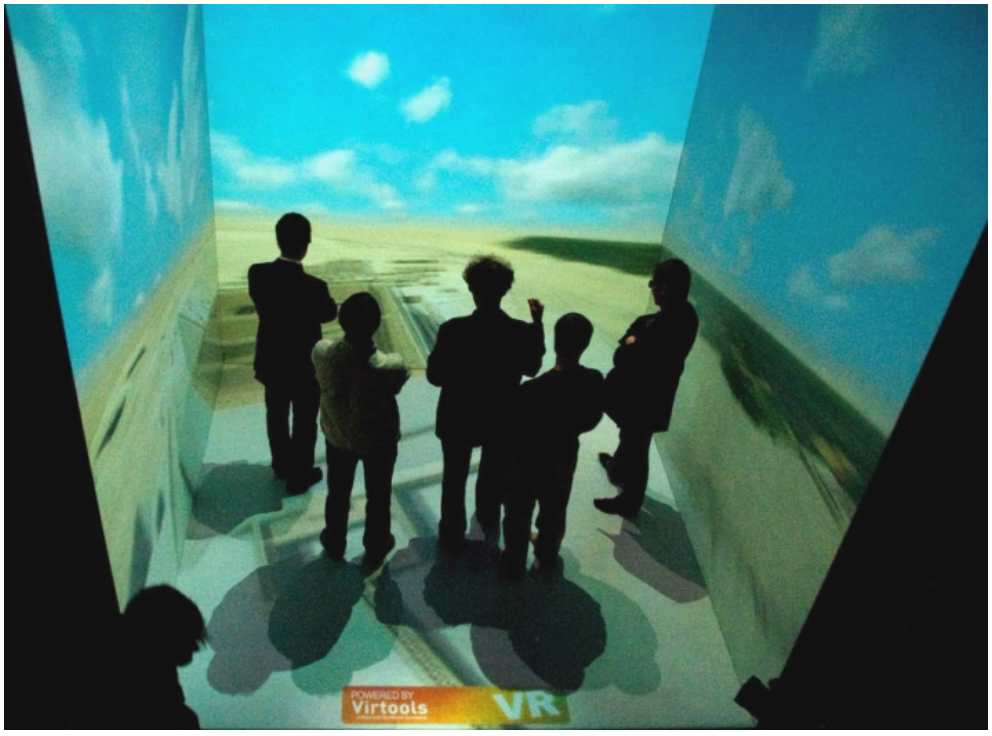


Figure 2.19: Stereo projected cave system. Source: [15]

3 Related Work

Flight simulators have been around in many flavors since the emergence of games on the computer. The advantage of using a finished product as a starting point is evident and would save a lot of time, but could also restrict the solution in terms of flexibility, modifiability and performance. This section briefly covers two popular software projects that have much in common with the aircraft simulator of this project and investigates whether the products or parts of them could be reused in this setting.

3.1 Freeware: Flying-Model-Simulator (FMS)

FMS [16] is a freeware model flight simulator originally intended for practicing on the control of small remote controlled airplanes and helicopters. The software is not open sourced and has been voluntarily maintained by two persons since year 2000, and naturally lags behind commercial or open source simulators. However, because it is free it has gained tremendous popularity in the RC (remote-control) community and is often bundled with RC products as an introduction to flying.

As a flight simulator FMS could fit the bill, but since its source is closed one cannot run autopilot software on it or extend the simulator to challenge the autopilot with windy conditions. However, if the source code could be granted by the authors, it does run a semi-realistic flight simulation that probably could be extended for autopiloting. The disadvantage is that the source code has gotten outdated by now and users report trouble getting the system to run on modern computers. Also the project is no longer actively maintained and the latest release dates back to 2005. This poses a risk if bugs should arise that is hard to trace without in-depth knowledge to the simulator engine.

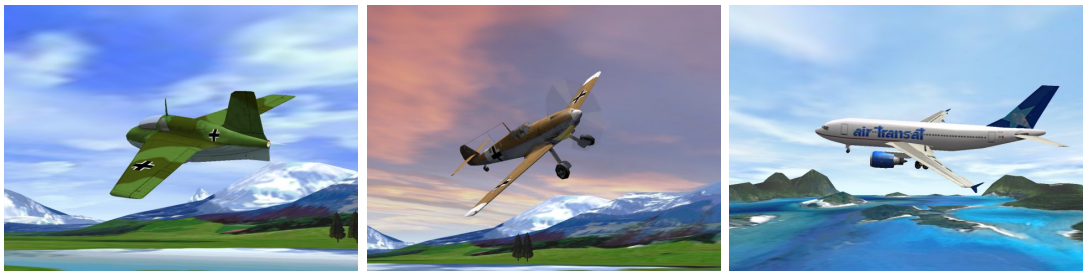


Figure 3.1: Screenshots from FMS 2.0 Beta 7. Source: [16]

Figure 3.2: Screenshots from FlightGear v1.9. Source: [17]

3.2 Open Source: FlightGear

A quick search on the Internet (pages [27, 28] among others) indicates that FlightGear[17] is the biggest, most popular and most active open source project on flight simulation today. It is daily updated by a large number of contributors and it features both modern graphics and a very sophisticated flight simulation. The code is free to download and modify by the terms of the GPL license, and it also supports high-level extensions that does not require one to modify the FlightGear engine code. The software runs on several platforms including Windows, Linux and Mac OSX.

One can choose between three different flight dynamics models (FDM) that each use different methods to determine the behavior of a specific aircraft.

JSBSim	a general open source FDM component
YASim	simulates the effects of airflow on different parts of the aircraft based on its mass and geometry
UIUC	an extension of an FDM originally written by NASA

The simulator features an extremely detailed world with real lakes, rivers, roads, cities and over 20,000 airports. The time of day modeling accurately represents the sun, moon and stars for any specific date and time and all four seasons are modeled as they naturally occur for different parts of the world.

The extreme level of simulation in FlightGear is nothing but impressive and it competes with the best of commercial simulators. Its extensibility in being open source and the plug-in support makes this software a great starting point for autopilot simulation. The only drawback is that it is designed for full-scale aircraft simulation, but it should be possible to model the dynamics of small-scale aircrafts as well. Also the level of simulation goes far beyond the needs of a small-scale drone simulator where the focus is on precise short-range flights near the ground. However, a lot of inspiration can be taken from the pluggable FDM solution it has implemented, since flight dynamics is the interesting part for testing autopilots on different aircrafts.

4 Discussion of Methods

The intention of this study is to identify novel or best-practice methods for implementing a real-time flight simulator to test autopilot software on. So far this report has covered basic principles of aerodynamics to better understand the forces at work during flight and how to control these for maneuvering. Secondly it has covered methods to calculate the effects of these forces in terms of drag and lift to simulate the flight behavior of the aircraft. Thirdly a few methods have been selected to help create a convincing visualization of outdoor flight, because the natural environment poses extra challenges in real-time applications.

This section will now discuss how these methods may be applied and their fit will be assessed by weighing their levels of realism and appeal to their costs in implementation and performance.

4.1 Flight Dynamics Model

The choice of physics simulation is a delicate compromise between performance and level of realism. In section 2.2.1 we covered parametric equations, and in particular the wing and fuselage methods were described in how to determine the parasite drag coefficient of an aircraft. Other methods also exist that derive the lift and drag coefficients from the geometrical and material structure, but generally all these methods suffer from inaccuracies and need to be corrected by empirical test results.

Computational fluid dynamics is an alternative approach covered in sections 2.2.2 and 2.2.3. In this context CFD aims to simulate the air particles and determine the interacting forces between the air and the aircraft. It is believed by many analysts that this method could simulate the entire complex behavior of air by the Navier-Stokes equations, but in practice there are no numerical methods or computing power available today that can solve this in real-time without dire approximations.

From these two categories of methods we can define four types of flight dynamics models (FDM) to use in a real-time flight simulator, as shown in table 4.2. Model 1 needs no prior knowledge about the aircraft structure and aims to simulate the effects of air around the body during flight. Models 2-4 rely on knowing the lift and drag coefficients for different angles of airflow at all significant parts of the aircraft and simply feed those to the lift and drag equations 2.1 and 2.2 respectively.

#	Method(s)	Model Description
1	CFD	Simulate the interacting forces of air and the aircraft structure by some governing equation for air
2	CFD + Parametric	Determine the lift and drag coefficients by simulating the air in a virtual wind tunnel
3	Parametric	Use published wind tunnel test data for specific aircraft models
4	Parametric	Pick lift and drag coefficients that give reasonable max velocities and control responses

Table 4.2: Different combinations of computational fluid dynamics and parametric equations to model the flight dynamics

4.1.1 Published Wind Tunnel Data

Models 2-4 differ only in how they obtain these coefficients, and if possible the best is to use published numbers from wind tunnels as proposed in model 3. Microsoft Flight Simulator X [29] is one commercial flight simulator that relies on numbers from aircraft manufacturers and achieves a high level of realism simulating real-world aircrafts. However, it seems difficult to find publications for small model airplanes and helicopters so that may rule out model 3.

4.1.2 Empirically Chosen Data

If we have no test data available a straight-forward approach is to manually pick coefficients as proposed in model 4. If one is trying to match a specific real-world model airplane or helicopter then experiments could be performed to measure the linear and angular motion as a function of control outputs over time. This could be realized by two cameras monitoring a remote controlled aircraft and the control sticks during flight. From this it would be possible to derive coefficients by trial and error that approximately match the original dynamics in a flight simulator. If live experiments cannot be performed it is still possible to construct a semi-realistic and believable flight dynamics model by picking coefficients that comply to reasonable constraints in max velocity and acceleration in linear and angular motion.

4.1.3 Virtual Wind Tunnel

Model 2 uses CFD to simulate air particles in a virtual wind tunnel and determines the lift and drag dynamics as one would in a real tunnel. Ideally this method is able to analyze an arbitrarily designed aircraft, and even a piano could be tested for its ability to fly (but it would be rather poor!). However, in practice this method is as good as its implementation and suffers from high computational costs and numerical dissipation, just as any CFD implementation. The advantage is that this analysis could be done just

once and so a lot of time could be spent to analyze the dynamics accurately and then apply the results to parametric methods for simulating the dynamics in real-time.

X-Plane [30] is a commercial flight simulator that allows you to design your own aircraft. The flight behavior is determined by breaking the aircraft down into many small elements and calculating the forces on each of them. The coefficients are obtained in run-time from the dynamic airflow and the pre-calculated structural analysis and used to calculate the forces of lift and drag.

4.1.4 Real-Time Aerodynamics Simulation

Model 1 is the most complex and computationally expensive solution of the four. It is so demanding that it is generally considered infeasible for flight simulation and most of the prior work is academic. However, the main limitations are computational power and fast, accurate numerical implementations. Many aerodynamics analysts believe [8, ch. 14] that once this limitation is overcome then CFD may just be the answer to aircraft design analysis and simulation. It is currently considered to have the best potential of truly modeling the complex behavior of air of the methods known today.

One study [31] from 2006 did try to use a CFD implementation in a real-time helicopter flight simulation to account for the complex interference of airwakes when landing on a boat. The report concluded that it should be possible to run such simulations at interactive frame rates using 500 to 1000 processor cores in parallel, so real-time flight simulation by CFD is probably some way into the future yet.

4.1.5 Best Fit for a Real-Time Flight Simulator

There is no obvious single best fit from the four flight dynamics models listed. Each has its pros and cons in terms of accuracy, performance, flexibility and implementation cost. Generally the models in table 4.2 are listed by decreasing complexity in a top-down manner, but it should be noted that the level of realism is not necessarily in the same order. The behavior of real-world aircrafts has been successfully approximated by parametric equations in simulators such as the Microsoft Flight Simulator series and used as a professional training aid for many years already.

However, the lift and drag coefficients by themselves are not sufficient. Moving parts such as flaps and gears complicate matters and atmospheric dynamics, local airflows and airwakes all affect the velocity field around the aircraft. To resolve local flows one could turn to fluid dynamics or similar variants such as the vorticity transport equations covered in section 2.2.2.4, but the backside is that without sufficiently accurate simulation even the simplest types of lift and drag would be poorly modeled. It is likely that the flight behavior would behave unnaturally if it would even fly at all.

For a master's thesis on the autopilot of model aircrafts the main concern is not so much the realism of flight dynamics as it is the autopilot's ability to adapt to different aircraft dynamics and winds. The physics need to be seemingly realistic, but a perfect model of air dynamics and local airflows is not a priority. By this assumption we can

safely conclude that a complete CFD simulation is neither necessary nor feasible for the scope of such a project.

A hybrid as described in model 2 is more interesting. To develop a virtual wind tunnel from scratch and determine the coefficients is a huge task and essentially a thesis on its own, but if open source components are available such as YASim in FlightGear then that could be one way to achieve a high level of realism for different types of model aircrafts.

If wind tunnel test data is available for such small-scale aircrafts then model 3 could be the best solution, but the flexibility in testing different aircrafts would be limited to the data available and manually tweaking existing data. If data is lacking then model 4 is a reasonable alternative. It should be possible to manually define flight dynamics that are sufficiently realistic for flying and testing the autopilot on. The approach is entirely empirical and a matter of subjective choices, but it is very quick to implement and for a project focused on developing the autopilot software that could be a good starting point in any case. The flight dynamics component should be designed to be pluggable and that should make it easier to improve or replace later.

4.2 Visualization

The goal of the visualization is not to accurately present the results of the physics simulation per se, but rather give the user a graphical presentation that is intuitive, appealing and fun to use. The simulator will function both as a flight simulator game and as a tool for testing autopilot control logic. The rendered environment is therefore only an approximation to the real world and the key focus is to give the user a good sense of scale, position and motion in the virtual world.

In section 2.3 we cover some methods to visualize a natural scene with terrain, trees, grass and weather. Natural scenes are particularly difficult to render in real-time and game development has been one of the main drivers behind such methods. Solutions exist in abundance in game development communities and in published works, so for this study we had to select a few promising methods that aim to solve some key issues of rendering outdoor scenes in real-time. The fit for this project was subjectively assessed for each method by how much time it would require to implement it, expected performance hit and its potential to create convincing and appealing graphics.

Evaluation of visualization methods differs slightly than for the methods of flight dynamics. We are still restricted to real-time performance, but in graphics the choice of methods do not necessarily decide the level of realism and appeal in the graphics. The final result often depends just as much on the artistic work as on the method used. A simple billboard tree with high quality textures can look convincingly real in certain applications in the same way as truly geometric grass can look artificial without proper lighting, geometry and textures.

This section discusses the fit of each method and whether the method or a variation of it could be used in the implementation.

4.2.1 Terrain

Rendering realistic terrain requires detailed meshes that often become a bottleneck. In section 2.3.1 we propose methods to cull parts of the terrain that do not need to be rendered and to reduce the level of detail for distant geometry with little impact on image quality.

Horizon culling is a clever culling technique that claims [10] to increase the frame rate by 2-4 times to ordinary frustum culling in detailed terrain with height variations that hide large parts of the terrain from the camera. The method seems fairly easy to implement and should provide a much needed performance boost for large detailed terrains. In particular it should prove effective for low altitude flights such as in simulated search missions.

To further improve the performance one method [11] suggests how the geometrical level of detail may be adjusted. The details of the paper indicate that the method is slightly intricate and could take some time to implement. In particular there are issues in the approach such as joining gridlets of different LOD and preserving normals across the edges of curved gridlets that complicate the implementation. It may not be desirable to implement this method unless there is an absolute need to improve the performance beyond frustum and horizontal culling. Also the simplicity of the technique is expected to produce "popping" behavior when a gridlet changes its LOD. However, the method does describe an implementation structure for dividing a large texture into tiles and mapping them to gridlets that will prove valuable in creating large, detailed non-repeating textures. For instance, it could be interesting to use large high-resolution aerial photographs of real-world terrain as a texture to create more realistic graphics.

4.2.2 Vegetation

Trees, bushes and grass are obvious elements in a natural 3D scene, but they are also important in providing a sense of scale. A small model helicopter flying in a terrain without vegetation may very well look full-scale if the terrain size cannot easily be determined. Humans excel at comparing the relative sizes of objects, but without good reference objects it can be hard to figure out the distance to or the size of an object in a 3D scene. This is particularly true if the visualization medium lacks depth, such as in computer screens.

In section 2.3.2 we suggest two methods for rendering different types of vegetation in real-time. The first method describes how to render realistic looking grass. Grass is an interesting element because its parallax effect is expected to greatly improve the sense of movement and visual appeal near the ground, and should prove particularly useful for precision navigation. The paper claims to achieve highly detailed grass up close that still looks natural and retains the parallax effect at a distance. Unfortunately the implementation seems too time consuming in that three different techniques are combined and each one would probably take some time to get right. It could be interesting to implement only the geometric technique as described in the paper, but the performance hit would likely limit the area and density of the grass. The transition to textured sur-

faces would also look very unnatural, and even though the density map technique helps transition between the two that is yet another implementation cost.

The method is generally complex and there are no obvious fallbacks if the implementation only gets halfway. Also the method only supports flat terrain and is not very well suited for the terrain of a flight simulator, although the paper notes that the technique could be extended to handle curved terrain as well. The gain of rendering grass is evident, but it seems the costs in performance and implementation may not be justified for the scope of this project. There are many other alternatives to grass that are far simpler and more efficient such as grass textures and billboards, that are commonly used in games today. It seems reasonable to start out simple and later add more advanced grass if time allows it.

The other method describes how to render the foliage of trees and bushes quickly and is particularly effective in rendering forests or large groups of bushes. The approach uses a novel improvement over billboards and introduces dynamically rendered impostors with per-leaf depth testing. This enables us to render very rich trees with thousands of seemingly geometric leaves, but there are a couple of drawbacks.

First it performs significantly slower than static billboarding techniques. Each visible tree must render a set of impostors with geometric leaves every frame and duplicate the impostors many times by sprite rendering. However, the paper claimed to render a forest canopy of approximately 200 trees with no culling at 50 frames per second on a modern PC per 2003, and should perform well in the real-time implementation slated for early 2010.

Second it is likely that the canopy looks natural only at a distance, and for flight near the ground with small-scale aircrafts it is expected that the impostors lose some of their illusion when flying near trees and bushes. This is not a major issue for a game-oriented visualization, but it should be clear that this method is far from a truly geometric representation of trees. It could be possible to use a level of detail technique to blend between impostor trees and fully geometric trees as a function of distance, but it might be difficult to achieve this without very noticable “popping” between the two representations.

If performance becomes a limiting factor the method easily allows one to adjust the number of leaves and impostors and if the implementation takes longer than expected it is relatively easy to degrade to a static billboarding technique. These factors make this method a good starting point for populating the terrain with vegetation.

An alternative would be to use off-the-shelf solutions for rendering trees at different levels of detail. SpeedTree [32] is one commercial software library that generates realistic, geometric foliage in many different flavors and is extensively licensed in recent games like Fallout 3, Batman: Arkham Asylum and Age of Conan - Hyborian.

4.2.3 Weather

In section 2.3.3 we described how the sky can easily be presented by a textured skydome. This technique is used in most 3D games today because it is very fast and yet looks convincing with proper artistic work. Clouds are typically superimposed as textures on

the skydome and animated by transformations, and the weather and time of day can easily be changed by blending between different textures of clouds and skies. Skydomes fit this simulator well because the typical scenario is flight near the ground and flying through volumetric clouds is not an issue. However, to present flying in poor weather we may need additional methods.

One method describes how to efficiently render falling rain and snow in a flight simulator as if the camera is moving through the falling particles. This is a typical example where game technology applies simple methods in a clever way to give the impression that more advanced visualization is going on. This illusion of rain and snow is computationally cheap and would be a good fit for visualizing flight in such weather conditions. For its application the main drawback is realism, since flat 2D textures are used to give an illusion of moving among millions of particles. However, the paper notes how a parallax effect can be achieved by layering several textures and scrolling them at different speeds, and with some tuning the end result can become quite believable.

An extension of this method could also to some extent visualize windy conditions. Gusts of wind could be shown by adjusting the direction that rain or snow is falling as a function of the wind vector. However, this is a crude approximation since all particles would be affected simultaneously and uniformly, ignoring the fact that it takes time for wind to propagate and that wind swirls as it goes. But even if it is not realistic it does help visualize gusts of wind and how the autopilot reacts when they occur.

The method is generally versatile and should run a lot faster and be simpler to implement than a particle system. Although the illusion is not very realistic the method complements the skydome in visualizing different kinds of weather and is a good starting point that fits the scope of this project well.

4.2.4 Perception of Depth

3D goggles for PCs have gained popularity in the recent years, much due to light-weight goggles that fit the home user budget. Polarized glasses have been replaced by LCD shutter glasses that enable vivid colors and smooth frame rates.

In 3D scenes this helps the user to accurately determine the position and the scale of things and can give the user a heightened sense of presence in the virtual world. The advantage of stereoscopy on computers is that the implementation is relatively simple and only requires the scene to be rendered from two slightly different camera positions. This does not necessarily mean we get half the frame rate, but there is a significant drop in performance for stereoscopic rendering that must be taken into account.

Generally one must explicitly render the stereoscopic effect, but NVIDIA 3D Vision [33] is a recent solution that turns any 3D application into a stereoscopic experience. The advantage is that older applications that originally did not support sterescopy can now be viewed with 3D googles. Unfortunately there is a problem that elements rendered without depth information, such as the heads up display, overlaid texts and crosshairs, suffer from ghosting artifacts and floating uncomfortably in front of where one is focusing. Generally one must explicitly render such elements at a proper depth, however 3D Vision has managed to substitute the crosshair with a 3D version in their graphics card

driver and fixes the crosshair in older first-person shooter games. It is possible that the remaining elements can be fixed in the future too, and if so it might no longer even be necessary to write stereoscopy code.

Tracking systems have also become increasingly popular because they are getting cheaper and more accurate. High-end HMDs often incorporate tracking of the head movement so that the pictures reflect the movement of the person wearing it. This opens up many interesting possibilities, such as sitting in a virtual cockpit of the aircraft and being able to move and look around as a pilot would in a real aircraft. With systems that can track the head movement over several meters it is to some extent possible to let the user walk and look around as if he is present in the virtual world. For movement over larger areas this type of tracking can complement regular game controllers in an interesting combination.

The ability to walk around makes the simulator more interactive and fun to use, but it also has practical uses. In particular it would be very helpful in evaluating the autopilot performance in precision navigation. The degree of freedom in walking around and observing the flight pattern in real-time is unmatched and would closely resemble a real flight experiment, and with no risk to life and limb!

5 Conclusion

A complete flight simulator is a huge undertaking and the intention of this study is merely to identify key theory and methods to better understand how a flight simulator can be realized. This report has given some insight into aerodynamics and discussed methods that simulate flight dynamics and visualize the outdoor environment in a realistic manner. We have covered existing flight simulators that each solve the task differently, from the relatively simple hobbyist project FMS to the huge open source project FlightGear that accurately simulates the inner workings of an aircraft down to the malfunction of gauges and mechanics.

We have learned that flight simulators generally model the flight dynamics by lift and drag coefficients for major parts of the aircraft or for the aircraft as a whole. These coefficients simply describe the tendency of an object to produce lift and drag when placed in an airflow at a specific angle. Wind tunnels measure these coefficients for different types of aircrafts and enable flight simulators to accurately model the dynamics of real-world aircrafts. We have also explored the possibility of simulating complex airflows and arbitrary aircraft designs by turning to computational fluid dynamics. Although real-time flight simulation by fluid dynamics is some way into the future yet, a lot of interesting work has been done in this field. We discovered that hybrid solutions exist such as YASim for FlightGear that do not necessarily model the airflow, but rather determine the flight dynamics by structural analysis.

The study of visualization techniques revealed some novel methods inspired by game technology for fast and convincing results. Outdoor 3D scenes are generally considered difficult because of the tremendous amount of detail in nature. This forces the methods to balance between performance and realism, and try to land on a compromise that is "good enough". We have learned how terrain rendering can be sped up by horizon and frustum culling and by adjusting the geometric level of detail, and how super-sized textures can be split up in tiles and properly mapped for highly detailed terrain. We have investigated recent methods for rendering huge amounts of living elements such as trees, bushes and grass in real-time to populate the terrain and increase the sense of scale and motion. We have discussed how to visualize clouds, rain, snow and winds to present different kinds of weather, and finally we have explored the possibilities of stereoscopy and tracking systems and how the immersive experience can be both useful and fun in a flight simulator game.

The theory and methods in this study give a brief glimpse of the challenges in creating a realistic flight simulator. We have discussed each of the methods in terms of realism and appeal compared to the costs in implementation and performance, and some of the methods stick out as more promising. The methods of horizon culling, large texture mapping and 2.5 dimensional foliage are particularly interesting since they seem rela-

tively easy to implement and should perform well by the claims in the papers. Together they enable large, detailed and textured terrains filled with hundreds of trees and bushes in real-time, and that would be a good starting point for the virtual world of a flight simulator.

In flight dynamics there is one method that looks promising. CFD simulation was discarded due to its complexity and computational cost, so that leaves simulation by parametric equations. There are generally two problems related to simulation by this method. First we need to determine the lift and drag coefficients for a particular aircraft design. We have already discussed methods to get these numbers, and options include open source components that perform structural analysis, published wind tunnel test data and choosing coefficients empirically. Second we must maintain a velocity vector field to model winds, atmospheric turbulence, local airflows and similar variations. To calculate the motion of an aircraft is then a matter of resolving force vectors of drag and lift by the vector field and adding in the thrust force vector. We also noted that an open source FDM such as JSBSim, YASim or UIUC could be used to handle this part of the simulator and reduce the scope of the implementation.

Realistic flight simulators have been around for some time now and best-practices are well established. Ever since the beginning parameters have been used to define the behavior of different types of aircrafts, and the level of realism was mainly constrained by the match of parameters and the level at which winds, weather processes and local airflows were modelled. However, in recent work [30, 31] we also see the emergence of structural analysis and simulation of fluid dynamics to further up the flexibility and realism.

This study set out to identify novel and best-practice methods for implementing a realistic real-time flight simulator game. The result is a thorough survey of methods discussed in light of solid theoretical background and related to prior work, which should provide a solid foundation to understand and realize such an implementation.

Bibliography

[1] J. Watkinson, Art of the Helicopter, Butterworth-Heinemann, 2004.

[2] Wiki, Drag (physics), http://en.wikipedia.org/wiki/Drag_(physics), visited 29.10.2009.

[3] Wiki, Supersonic, http://en.wikipedia.org/wiki/Supersonic, visited 20.11.2009.

[4] D. Anderson, S. Eberhardt, Understanding Flight (2nd Edition), McGraw-Hill Professional, 2009.

[5] The Renaissance Charter School's Public Wiki, Eddies, http://www.andybaird.com/travels/life05/photos/three-eddies.jpg, visited 16.09.2009.

[6] C. G. Forum, What's the most aerodynamic shape?, http://cr4.globalspec.com/PostImages/200806/737px_Airplane_vortex_edit_8F32D8CA-9576-EF75-3EAA339AEEAB9FFB.jpg, visited 09.09.2009.

[7] J. J. Bertin, R. M. Cummings, Aerodynamics for Engineers (5th Edition), Prentice Hall, 2008.

[8] J. G. Leishman, Principles of Helicopter Aerodynamics (Cambridge Aerospace Series), Cambridge University Press, 2006.

[9] Innovative-CFD.com, Quick overview of cfd grid terminology, http://www.innovative-cfd.com/cfd-grid.html, visited 03.10.2009.

[10] B. Lloyd, P. Egbert, Horizon occlusion culling for real-time rendering of hierarchical terrains, in: VIS '02: Proceedings of the conference on Visualization '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 403–410.

[11] A. Brodersen, Real-time visualization of large textured terrains (2005) 439–442doi:10.1145/1101389.1101477.

[12] K. Bouatouch, K. Boulanger, S. Pattanaik, Rendering Grass in Real Time with Dynamic Light Sources, Research Report RR-5960, INRIA (2006). URL http://hal.inria.fr/inria-00087776/en/

[13] G. Szijártó, J. Koloszár, Hardware accelerated rendering of foliage for real-time applications (2003) 141–148doi:10.1145/984952.984976.

[14] N. Wang, B. Wade, Rendering falling rain and snow (2004) 14doi:10.1145/1186223.1186241.

[15] S. Kuntz, A vr geek blog, http://cb.nowan.net/blog/tag/reference/, visited 11.11.2009.

[16] Roman, M. Möller, Flight-model-simulator, http://n.ethz.ch/student/mmoeller/fms/index_e.html, visited 13.11.2009.

[17] Flightgear, http://www.flightgear.org, visited 13.11.2009.

[18] Wiki, Bernoulli's principle, http://en.wikipedia.org/wiki/Bernoulli's_principle, visited 07.12.2009.

[19] S. E. D. Anderson, How airplanes fly: A physical description of lift, http://www.aviation-history.com/theory/lift.htm, visited 20.11.2009 (02 1999).

[20] Fixed wing aircraft facts and how aircraft fly, http://www.aviationexplorer.com/fixed_wing_aircraft.htm, visited 07.12.2009.

[21] G. M. Craig, Physical principles of winged flight, http://www.regenpress.com/aerodynamics.pdf, visited 07.12.2009 (2003).

[22] P. Eastwell, Bernoulli? perhaps, but what about viscosity?, http://www.scienceeducationreview.com/open_access/eastwell-bernoulli.pdf, visited 07.12.2009 (2007).

[23] J. S. Denker, See how it flies: Perceptions, procedures and principles of flight (1996).

[24] B. Atkinson, Dynamical Meteorology: An Introductory Selection (1st Edition), Routledge, 1981.
URL http://books.google.no/books?id=gikOAAAAQAAJ&lpg=PP1&pg=PP1#v=onepage&q=&f=false

[25] T. Benson, Reynolds number, http://www.grc.nasa.gov/WWW/BGH/reynolds.html, visited 28.10.2009.

[26] H. Versteeg, W. Malalasekra, An Introduction to Computational Fluid Dynamics: The Finite Volume Method (2nd Edition), Prentice Hall, 2007.

[27] J. Montgomery, Top 10 best open source games, http://tech.blorge.com/Structure:%20/2009/02/15/top-10-best-open-source-games/, visited 07.12.2009 (02 2009).

[28] Top 5 free linux games,
http://www.ixibo.com/2008/09/top-5-free-linux-games-download-now,
visited 07.12.2009 (09 2008).

[29] Microsoft flight simulator x,
http://www.microsoft.com/games/flightsimulatorX, visited 17.11.2009.

[30] X-plane, http://www.x-plane.com, visited 17.11.2009.

[31] Real-time cfd for helicopter flight simulation,
http://cmg.devel.mauveinternet.co.uk/research/projects/
real-time-cfd-for-helicopter-flight-simulation/, visited 17.11.2009.

[32] Speedtree, http://www.speedtree.com, visited 13.12.2009.

[33] J. M. H. Butler, Nvidia geforce 3dvision & how 3d works, http://www.bit-tech.
net/hardware/graphics/2009/01/09/nvidia-geforce-3dvision-review/4,
visited 18.11.2009 (01 2009).

NTNU

Norwegian University of
Science and Technology