Flight Dynamics Simulation

Aleksander Folfas
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
aleksander23@ru.is

Keith Kwok
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
keith23@ru.is

Toh Hong Jing
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
hong23@ru.is

Figure 1: *Airplane model in flight*

Abstract

In this project, we dive into the process of bringing flight into Unreal Engine 5 using a mix of C++ programming and blueprint scripting. The paper outlines the physics equations used to replicate real-world flight dynamics within the virtual environment. Additionally, it discusses the outcomes of the simulation and provides insights into the team's experience with utilizing Unreal Engine 5 for this purpose.

Keywords: Flight Simulation, Unreal Engine 5, aerodynamics, Game Developement

Concepts: •**Computing methodologies** → **Interactive simulation;**

1 Introduction

In the realm of flight simulation, there's a delicate balance to strike between the captivating simplicity of video game physics and the intricate realism of real-world flight dynamics. We set out to tackle this challenge head-on by crafting a custom physics engine plugin, which game developers can easily use in their projects. Our motivation is to create a simple solution to simulate flight movement in a way that captures authenticity while remaining engaging for both aviation enthusiasts and gamers alike. Through this endeavor, we aim to bridge the gap between simplified game physics and the complexities of actual flight.

2 Related Work

The gaming industry, with its emphasis on immersive experiences, has greatly influenced our work. GTA V stands out as a primary source of inspiration, especially its ability to engage players with compelling flight mechanics within a vast open-world environment. While GTA V's flight model is more simplified than a dedicated flight simulator, its integration into a dynamic game world sparked our interest in exploring how advanced flight dynamics could enhance the gameplay experience in similar titles.

Open-source initiatives have further enhanced our perspective. For example, the FlightGear flight simulator project offers an open-source, collaborative approach to flight simulation.

In the realm of commercial flight simulators, platforms like Microsoft Flight Simulator have set high standards for realism and accuracy. These simulators employ sophisticated physics engines that model the flight dynamics of various aircraft with remarkable details.

3 Flight Mechanics

To simulate the physics behind flight, one has to understand the underlying equations behind aerodynamics relevant to the flight. Firstly, the definitions for lift, drag, and weight are given below for an aerodynamic body. The below equations can be derived from NASA's open-source web page as well as Fundamentals of Aerodynamics by Anderson, J. D.

3.1 Essential Equations

$$Lift(L) = \frac{1}{2}\rho V^2 S C_L \tag{1}$$

where:

ρ = Density of Fluid (kg/m^3)
V = Velocity of Body (m/s)
S = Wing Area (m^2)
C_L = Coefficient of Lift

$$Drag(D) = \frac{1}{2}\rho V^2 A C_D \tag{2}$$

where:

ρ = Density of Fluid (kg/m^3)
V = Velocity of Body (m/s)
A = Frontal Area (m^2)
C_D = Coefficient of Drag

$$Weight(W) = mg \tag{3}$$

where:

m = Mass of Body (kg)
g = Acceleration due to Gravity (m/s^2)

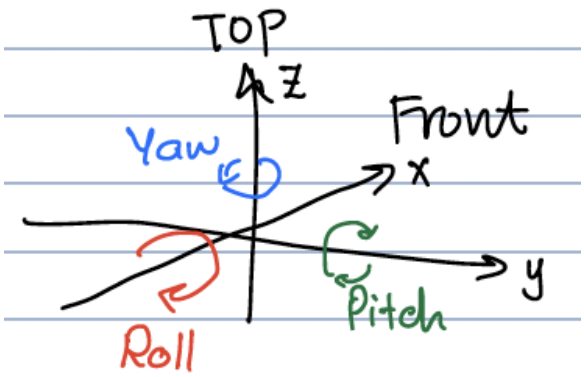

Figure 2: *Axis of rotations*

3.2 Climbing

In this section, the dynamics of body climbing will be analyzed.

Refer to Figure 2, the rotation axis of an object is illustrated. Yaw happens about the z-axis, while roll and pitch about the x- and y-axis, respectively.

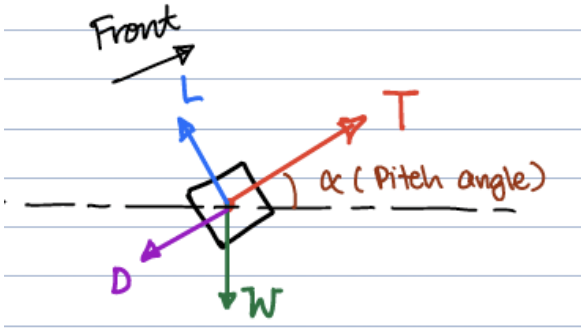

Figure 3: *Free body diagram of a climbing body*

Figure 3 illustrates the free body diagram for a body climbing without turning. The free-body diagram illustrates the forces acting on the body during a climb. Two new terms have been introduced in this free body diagram, T which represents the thrust or forward force produced by the body, α the pitch angle with reference to the relative wind direction.

With reference to the free body diagram, the following resultant forces can be calculated.

$$F_{vertical} = L\cos\alpha + T\sin\alpha - D\sin\alpha - W \tag{4}$$

where:
L = Lift of Body (N)
T = Thrust of Body (N)
D = Drag of Body (N)
W = Weight of Body (N)
α = Pitch Angle ($^\circ$)

$$F_{horizontal} = T\cos\alpha - L\sin\alpha - D\cos\alpha \tag{5}$$

where:
L = Lift of Body (N)
T = Thrust of Body (N)
D = Drag of Body (N)
α = Pitch Angle ($^\circ$)

It is to be noted that these forces could also be written as $F = ma$. Hence, one can easily find the vertical and horizontal acceleration by equating

$$F_{vertical} = ma_{vertical} \tag{6}$$

and

$$F_{forward} = ma_{horizontal}. \tag{7}$$

3.3 Turning

In this section, the dynamics of a body turning will be analyzed.

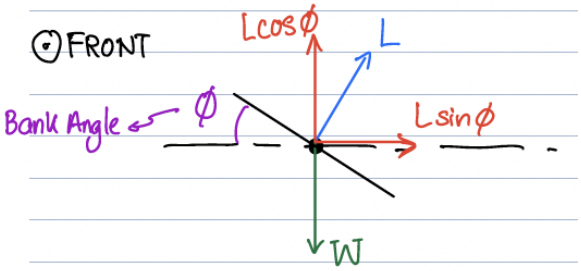

Figure 4: *Free body diagram of a turning body*

Figure 4 illustrates the free body diagram for a body turning without climbing, showing the forces applied to that body. A new term ϕ is introduced that illustrates the bank angle of the object with respect to the horizontal.

Resolving the forces of Lift, the following equations for a turning body are obtained.

$$F_{vertical} = L\cos\phi - W \tag{8}$$

where:
L = Lift of Body (N)
W = Weight of Body (N)
ϕ = Bank Angle ($^\circ$)

$$F_{side} = L\sin\phi \tag{9}$$

where:
L = Lift of Body (N)
ϕ = Bank Angle ($^\circ$)

3.4 Turning and Climbing

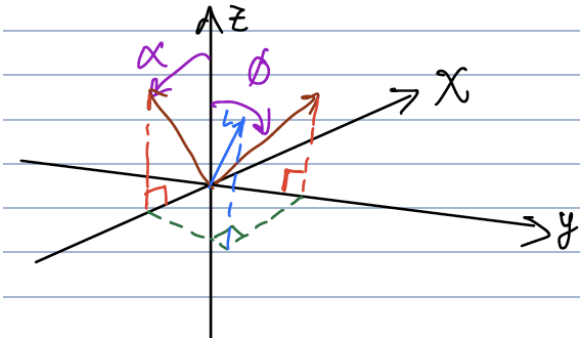

Figure 5: *Resolution of lift force based on the pitch and bank angle*

In 3-dimensions, a lift vector would be decomposed by the pitch and roll components of the lift vector. Hence, a lift vector is first

decomposed to its effective lift with respect to its pitch and finally the effective lift with respect to the roll.

$$L_{pitch} = L\cos\alpha \tag{10}$$

$$L_{roll} = L\cos\phi \tag{11}$$

Combining both roll and pitch, the effective vertical component of lift is,

$$L_{vertical} = L\cos\alpha\cos\phi \tag{12}$$

Similarly, the effective sideway component of lift is,

$$L_{side} = L\cos\alpha\sin\phi \tag{13}$$

The forward force due to the pitch of the body can also be derived from lift.

$$F_{forward\,due\,to\,pitch} = -L\sin\phi \tag{14}$$

With that, the following resultant forces on a body in a 3-dimensional environment can be derived.

$$F_{vertical} = L\cos\alpha\cdot\cos\phi + T\sin\alpha - D\sin\alpha - W \tag{15}$$

$$F_{forward} = T\cos\alpha - L\sin\alpha - D\cos\alpha \tag{16}$$

$$F_{side} = L\cos\alpha\cdot\sin\phi \tag{17}$$

4 Simulation Model

4.1 Model Specification

In this physics simulation, the aerodynamic forces applied to a body are simplified. In this model, the yaw, pitch, and roll forces are assumed to be negligible. The assumptions of the model are listed as follows:

- C_D and C_L are fixed
- Components of drag (Parasitic and lift-induced) can be modelled well with C_D
- Density of fluid remains constant with altitude
- Yaw, pitch, and roll forces are negligible
- Airflow is always from the front of the body and parallel to the horizontal
- Centre of gravity does not affect the moments of the body
- All forces act upon the centre of mass of the body
- Yawing is not possible
- Wing area is kept constant relative to the relative wind

The simulation would take in a list of inputs as follows:
World Inputs:

1. Density, ρ (kg/m^3)
2. Gravitational Acceleration, g (m/s^2)

Flyable Object Inputs:

1. Mass, m (kg)
2. Wing Area, S (m^2)
3. Coefficient of Lift, C_L (Dimensionless)
4. Coefficient of Drag, C_D (Dimensionless)
5. Frontal Area, A (m^2)

Player Inputs:

1. Thrust, T (N)
2. Roll Input (Assumed constant rate of angular change)
3. Pitch Input (Assumed constant rate of angular change)

The angles, α, and ϕ can be obtained by the relative rotation about the y-axis and x-axis of the body, respectively.

The forces to be applied to the body can be obtained from the 3-dimensional resultant force derived in equations 16, 15 and 17.

Using the resultant force, one can also find the acceleration of the body based on its mass if required.

$$a_{forward} = \frac{F_{forward}}{Mass}$$

$$a_{side} = \frac{F_{side}}{Mass}$$

$$a_{vertical} = \frac{F_{vertical}}{Mass}$$

4.2 Relative forces

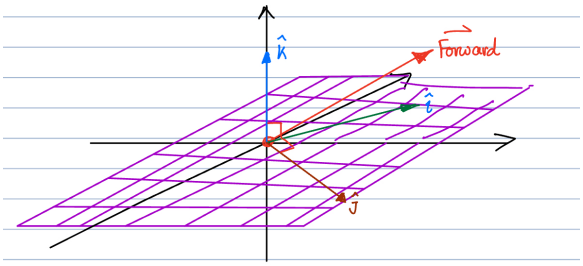

Figure 6: *Basis vector to apply forces*

Now that the vertical, forward, and side forces can be obtained by equations 15, 16 and 17. These forces need to be applied relative to the orientation of the body. As such, there is a need to obtain the basis vectors for these orientations. The direction to apply the forward force is the directional vector for the facing of the body, this can be easily obtained from the forward vector of the body, $\vec{Forward}$ in Unreal. Next, with the assumption of the model, the lift force would always be applied in the direction of the vertical axis, $\hat{k}$, hence the vector normal to the xy-plane. To find the direction of side force, $\vec{Forward}$ is projected onto the xy-plane. This can be calculated by normalizing the x and y components of $\vec{Forward}$, giving us $\hat{i}$. Finally, the cross product of $\hat{i}$ and $\hat{k}$ would yield the direction to apply the side force, $\hat{j}$ (refer to Figure 6).

5 Results

5.1 What worked

We've accomplished the development of an Unreal Engine plugin that encapsulates our physics flight engine. To access the engine's main functionality within blueprints, we are using a custom function named *Flight Simulation*, implemented in C++. The function is depicted in Figure 7 as a blueprint node. We recommend calling this node every tick using the *Event Tick* node. Users are required to provide a reference to the actor and static mesh, while the remaining parameters should align with the chosen model to enhance immersion.

A key feature of our engine is its interactive control system, which allows users to directly manipulate the thrust of an aircraft via simple keyboard inputs. By binding the increase and decrease of thrust

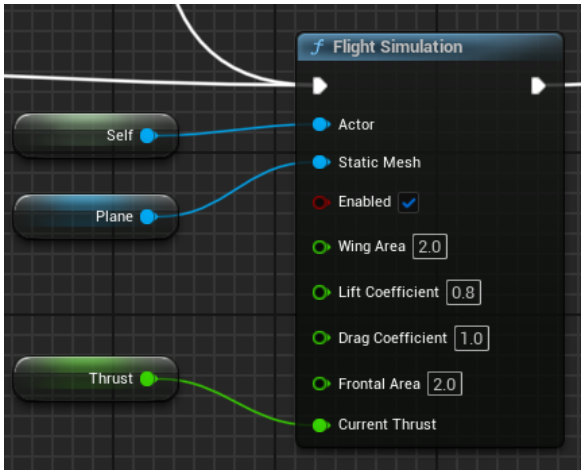

Figure 7: *Blueprint node used to access flight engine functionality, the parameters can be provided from other nodes or manually inserted.*

to the "W" and "S" keys, respectively, we provided a user-friendly interface that enables both novice and experienced users to intuitively control the aircraft. Furthermore, by binding the adjustments of the roll to the "A" and "D" keys, respectively, we were able to further enhance the realism and responsiveness of the flight simulation.

5.2 What didn't work

Integrating C++ with Unreal Engine's visual blueprints proved to be more challenging than we thought. Moreover, our development process was significantly slowed by the repeated crashes within the Unreal Game Engine.

Reflecting on these challenges, future efforts could benefit from starting with smaller, incremental integration and gradually expanding the complexity as stability and compatibility are ensured. Additionally, investing time in more robust debugging and testing frameworks within Unreal Engine could mitigate the impact of crashes and improve the overall development experience.

6 Future Work

- Create a realistic-looking environment for the simulation
- Implement dynamic weather systems such as wind, turbulence, and rain that would affect flight conditions
- Allow the body to be influenced by yaw and pitch forces, by implementing moments and applying forces in the fore and aft of the center of gravity of the body.

7 Conclusion

In conclusion, our endeavor to bring flight simulation into Unreal Engine 5 has been both challenging and rewarding. Through the development of a custom physics engine plugin and integration with Unreal Engine's visual blueprints, we have made significant progress toward creating an immersive and authentic flight experience.

Despite encountering obstacles such as crashes and difficulties in integrating C++ with visual blueprints, we have persevered and gained valuable insights that will inform future development efforts. The process has underscored the importance of incremental integration, robust testing, and investment in debugging frameworks to ensure stability and compatibility.

Looking ahead, there are exciting opportunities for further refinement and expansion of our flight simulation project. Enhancements such as realistic environment creation, dynamic weather systems, and improved flight dynamics will continue to elevate the realism and engagement of the simulation.

Ultimately, our goal remains unchanged: to bridge the gap between simplified game physics and real-world flight dynamics, providing users with an immersive and captivating flight experience. With continued dedication and innovation, we are confident that our flight simulation project will continue to evolve and delight aviation enthusiasts and gamers alike.

Acknowledgements

To Professor Hannes Högni Vilhjálmsson for his guidance.

To Tharidu for providing the airplane model for free.

References

ANDERSON, J. D. 2007. *Fundamentals of Aerodynamics (4th ed.)*. McGraw-Hill.

EPIC GAMES, 2024. Unreal engine documentation. https://dev.epicgames.com/documentation/en-us/unreal-engine. Accessed: 7 April 2024.

FLIGHTGEAR PROJECT DEVELOPERS, 2020. Flightgear flight simulator. https://www.flightgear.org/. Accessed: 7 April 2024.

MICROSOFT CORPORATION, 1982. Microsoft flight simulator. https://www.flightsimulator.com/. Accessed: 7 April 2024.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA), 2024. Beginner's guide to aeronautics: Learn about aerodynamics. https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/learn-about-aerodynamics/. Accessed: 27 March 2024.

Appendix

```
#pragma once

#include "Kismet/BlueprintFunctionLibrary.h"
#include "CoreMinimal.h"
#include "FlightSimPluginBPLibrary.generated.h"

DECLARE_LOG_CATEGORY_EXTERN(LogFlightSimPlugin, Log, All)
UCLASS(Blueprintable)
class UFlightSimPluginBPLibrary : public
    UBlueprintFunctionLibrary
{
        GENERATED_UCLASS_BODY()

        UFUNCTION(BlueprintCallable)
        static void FlightSimulation(AActor* actor,
            UStaticMeshComponent* StaticMesh, bool
            enabled, float wingArea, float
            liftCoefficient, float dragCoefficient,
            float frontalArea, float CurrentThrust);
};
```

Listing 1: *C++ code header for UFlightSimPluginBPLibrary*

```
#include "FlightSimPluginBPLibrary.h"
#include "FlightSimPlugin.h"
#include "Components/StaticMeshComponent.h"

using namespace std;

const float GRAVITY = 9.81f;
const float AIR_DENSITY = 1.225f;

DEFINE_LOG_CATEGORY(LogFlightSimPlugin);
UFlightSimPluginBPLibrary::UFlightSimPluginBPLibrary(
    const FObjectInitializer& ObjectInitializer) : Super
    (ObjectInitializer)
{

}

void UFlightSimPluginBPLibrary::FlightSimulation(AActor*
    actor, UStaticMeshComponent* StaticMesh, bool
    enabled, float wingArea, float liftCoefficient,
    float dragCoefficient, float frontalArea, float
    CurrentThrust)
{
    if (!actor)
    {
        UE_LOG(LogFlightSimPlugin, Warning, TEXT("Invalid
            actor passed to FlightSimulation function")
            );
        return;
    }

    if (!StaticMesh)
    {
        UE_LOG(LogFlightSimPlugin, Warning, TEXT("Invalid
            static mesh"));
        return;
    }

    // Get mass
    float mass = StaticMesh->GetMass();

    UWorld* World = actor->GetWorld();
    if (!World)
    {
        UE_LOG(LogFlightSimPlugin, Warning, TEXT("Invalid
            world"));
        return;
    }

    // Get Delta time from world
    float DeltaTime = World->GetDeltaSeconds();

    // Get the velocity vector
    FVector VelocityVector = StaticMesh->
        GetComponentVelocity();

    // Set the Z component to 0
    VelocityVector.Z = 0.0f;
    VelocityVector.Y = 0.0f;

    // Calculate the magnitude of the modified velocity
        vector
    float VelocityMagnitude = VelocityVector.Size() /
        100;

    // Calculating lift, drag, and weight based on the
        document
    float lift = 0.5f * AIR_DENSITY * VelocityMagnitude *
         VelocityMagnitude * wingArea * liftCoefficient;
    float drag = 0.5f * AIR_DENSITY * VelocityMagnitude *
         VelocityMagnitude * frontalArea *
        dragCoefficient;
    float weight = mass * GRAVITY;

    // Convert angle for pitch and roll from degree into
        radians
    FRotator ActorRotation = StaticMesh->
        GetComponentRotation();
    float AlphaRad = FMath::DegreesToRadians(
        ActorRotation.Roll);
    float PhiRad = FMath::DegreesToRadians(ActorRotation.
        Pitch);

    // Calulcate forces based on the physics equations
        highlighted in yellow
    float Fvert = lift * FMath::Cos(AlphaRad) * FMath::
        Cos(PhiRad) + CurrentThrust * FMath::Sin(
        AlphaRad) - drag * FMath::Sin(AlphaRad) - weight
        ;
    float Fforward = CurrentThrust * FMath::Cos(AlphaRad)
         - lift * FMath::Sin(AlphaRad) - drag * FMath::
        Cos(AlphaRad);
    float Fsideways = lift * FMath::Cos(AlphaRad) * FMath
        ::Sin(PhiRad);

    // Logs
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("Fvert is %f
        "), Fvert);
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("Fforward is
         %f"), Fforward);
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("Fsideways
        is %f"), Fsideways);
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("Lift is %f"
        ), lift);
    //UE_LOG(LogFlightSimPlugin, Warning, TEXT("Velocity
        Vector is %f, %f, %f"), VelocityVector.X,
        VelocityVector.Y, VelocityVector.Z);
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("Velocity
        magnitude is %f"), VelocityMagnitude);

    // Apply the calculated forces to the actor
    FVector ForceToAdd = FVector( Fforward, Fsideways,
        Fvert);
    FVector ForwardVector = -StaticMesh->GetForwardVector
        ();
    UE_LOG(LogFlightSimPlugin, Warning, TEXT("
        ForwardVector is %f, %f, %f"), ForwardVector.X,
        ForwardVector.Y, ForwardVector.Z);
    FVector ForwardVectorReference = ForwardVector;
    ForwardVectorReference.Z = 0;
    ForwardVectorReference = ForwardVectorReference /
        ForwardVectorReference.Size();
    FVector SidewaysVector = ForwardVectorReference.Cross
        (FVector(0, 0, 1));
```

```
84      FVector VerticalVector = (ForwardVector.Cross(
            SidewaysVector));
85      VerticalVector.Z = -VerticalVector.Z;
86
87      StaticMesh->AddForce(Fforward * SidewaysVector,
            NAME_None, true);
88      StaticMesh->AddForce(Fsideways * ForwardVector,
            NAME_None, true);
89      StaticMesh->AddForce(Fvert * VerticalVector ,
            NAME_None, true);
90  }
```

Listing 2: *C++ code for UFlightSimPluginBPLibrary*