

Politecnico di Torino

Facoltà di Ingegneria

Corso di laurea specialistica in Ingegneria Informatica



Tesi di Laurea

3D scientific visualization of wake vortices using the game
engine Unity3D

Candidate:	Matteo Franchini
Relatore:	Prof. Andrea Bottino
Supervisore:	Ing. Wito Engelke

Ringraziamenti

Alla mia famiglia, che nei momenti di bisogno mi ha aiutato ed ha reso possibile più di chiunque altro questo obiettivo.

Ai miei amici, che anche inconsapevolmente mi hanno sempre spronato ad andare avanti. Siete stati la medicina contro la mia innata pigrizia

Index

1	ABSTRACT	4
2	INTRODUCTION	5
2.1	WAKE VORTICES ISSUE	5
2.2	P2P MODEL AND VISUAL REPRESENTATION	6
2.3	TESTING UNITY3D	7
2.4	GOALS	7
3	BACKGROUND	8
3.1	AERODYNAMIC EFFECT OF THE WINGTIP VORTICES	8
3.2	P2P PROGRAM	10
3.3	GAME ENGINES AND UNITY 3D	12
3.4	UNITY 3D IN SCIENTIFIC VISUALIZATION	13
4	DEVELOPMENT	14
4.1	3D WORLD SCENARIO	14
4.2	MODELING PROCESS	16
4.3	STRUCTURE AND WORKFLOW	18
4.4	ANIMATION SCRIPTS	20
4.5	NETWORKING	27
5	USE CASE	28
5.1	APPLICATION SET UP	28
5.2	USER INTERFACE AND SCREENSHOTS	30
5.3	DATA CONGRUENCE VALIDATION	31
6	RESULTS AND CONCLUSIONS	34
6.1	RESULTS ACCORDING TO GOALS	34
6.2	CAN UNITY3D USED FOR SCIENTIFIC VISUALIZATION?	35
6.3	FUTURE DEVELOPMENTS	35

1 Abstract

As a consequence of lift, aircraft's wings generate a pair of counter rotating wake vortices that represent a serious risk for following airplanes. Nowadays separation time standards between consecutive aircrafts, significantly influence the air traffic control, often leading to delays and congestions, and consequentially to a drop of the airport capacity. Having a 3D representation of this aerodynamic effect based on a real time vortex prediction model, could be useful to improve the management of the air traffic control in a specified airport, without losing any degree of security.

Thanks to the big development of the gaming industry, high performing graphic hardware has been produced and, short time later, different companies decided to implement software tools to exploit as much as they could this new type of standalone graphic cards. In this way different Game Engines started to arise, born with the purpose of helping programmers in the process of game developing. However considering the great potential of game engines, some researchers started testing if these frameworks could be used not just for gaming but also to develop scientific simulations. In our project we wanted to do exactly this, using a specific game engine, called Unity3D, to create the graphic visualization of the wake vortices, basing the simulation on real data developed by a mathematical model.

In the following chapters we will discuss in details why we chose Unity3D, how we implemented the simulation, what other tools we used, what results we reached and which development prospects our application can have.

2 Introduction

2.1 Wake vortices issue

As we stated in the abstract, the wake vortices can represent a serious problem for incoming aircrafts. Flying into this turbulence path can lead to unpredictable behavior of the airplane with serious consequential safety issues. Wake turbulence is caused by the higher-pressure air on the underside of an aircraft's wing mixing with the lower-pressure area on its upper side. [1]

Current wake vortices separation standards, follow over-conservative separation times that lead to a significant decrease of the capacity of the airport. (Figure 2.1)

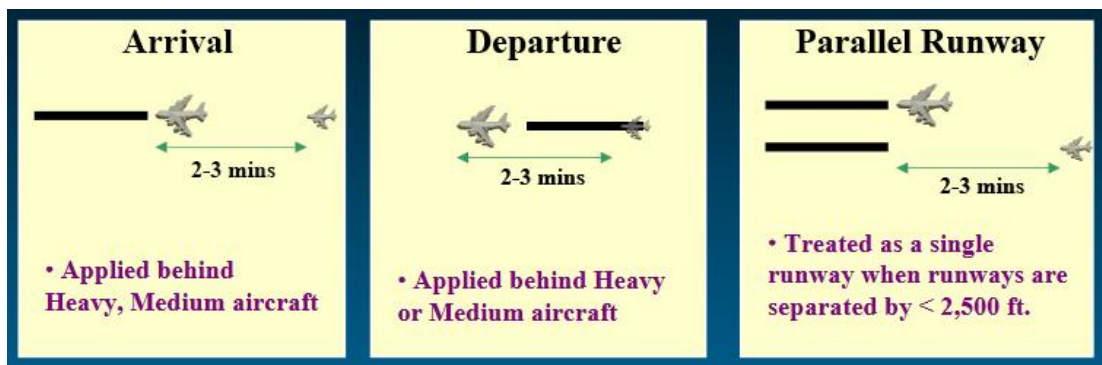


Figure 2.1 [2]

Wake vortices are especially oppressive in proximity of airports runaways where the nearby presence of landing airplanes, together with close infrastructure and aircraft circling overhead, makes these strong whirlwind of turbulence a considerable safety issue. For example in the United States the Federal Aviation Administration, has strict rules which avoid landing small aircrafts to be closer than six miles to a large plane, and large jets to be closer than four miles during landings, even if improvements have been made to vary these separation distances according to certain circumstances. During the years, wake vortices have been the topic of research projects conducted by airport and aerospace centers. One of them is the European Commission's ATC-WAKE project, from 2002 to 2005, which worked to improve airport planning with the goal to obtain variable aircraft separation plans and trim valuable minutes from the schedules [1]. The National Aeronautic Space and Administration (NASA) has been studying for many years the wake vortices. In the recent years, the researches around the wingtip vortices has moved towards the development of mathematical models to create a bigger and more complex view of

vortices and their interaction with atmospheric conditions. In fact, in order to give a detailed graphic representation of this effect, we needed a reliable tool that could provide us proper data suitable to develop the visualization. All these kind of information are developed by a mathematical model called "Probabilistic Two-Phase Wake Vortex Decay and Transport Model" or P2P.

2.2 P2P model and visual representation

The P2P (Probabilistic two phase wake vortex decay and transport model) is a mathematical model that receives some specific data in input and develops confidence intervals of the position and intensity of the vortex at a certain time-step as output. It accounts for the effects of wind, turbulence, stable stratification, and ground proximity. "The model equations are derived from the analytical solution of the spatiotemporal circulation evolution of the decaying potential vortex and are adapted to wake vortex behaviour as observed in large-eddy simulations". [3] Wake vortex degeneration happens in two different phases, a first diffusion phase followed by quick degeneration phase following a non-linear function of the vortex intensity. Probabilistic components of the P2P model account for deviations from deterministic vortex behaviour naturally generated by the stochastic nature of turbulence, vortex instabilities, and deformations, and also uncertainties coming from environmental and aircraft parameters. In order to set a specified degree of probability, the model architecture allows you to always adjust the decay parameters and uncertainty allowances, considering the growing amount of data. [3] The position and strength of the vortices are taken as input from our application, called 3D Virtual Airport, in order to give, as final output the graphic 3D simulation. (Figure 1.2)

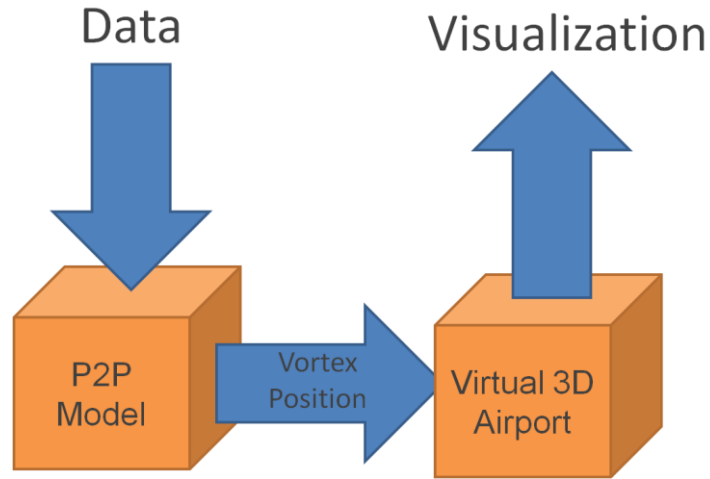


Figure 2.2

The 3D virtual Airport has been developed almost entirely using the game engine Unity3D for what concern the data visualization part. All the calculation of vortex position and intensity are generated by the P2P model.

2.3 Testing Unity3D

Visualization of scientific data can be helpful for the interpretation of results. We developed here a tool to show in particular how the wingtip vortices progress in time and we tried to give an impactful visual representation of it. To reach our scope, we used the high level modern game engine Unity3D, in order to understand if this powerful tool can be used in scientific applications and not just to develop games. Recently the computer gaming industry has grown up exponentially and big amount of money have been invested in the development of game engine frameworks. In contrast to their development costs, the price of final tool is very low if we compare it to professional 3D simulation software. [4] For these reasons we decided to use a game engine to develop our 3D Virtual Airport. Later on we will discuss more in details why we chose exactly Unity 3D.

2.4 Goals

Considering all these premises our main goal was to build from scratch a virtual airport in the 3D world, and create some objects to use as primitive tassels to represent the wake vortices on the screen. Once all the scenario and the primitives

are set up and ready, we wanted to exploit the tools we created to represent all the data that the P2P model send in output.

3 Background

3.1 Aerodynamic effect of the wingtip vortices

The wake vortices (wingtip vortices) are patterns of rotating air, or any other fluid, generated behind the tip of a wing and both the wings release from their tip a different wake vortex. It is also important to understand that wingtip vortices are inevitable because they are the natural consequence of the Lift generation. [5] There are some design choices that can be taken in order to reduce this induced drag, picking a specific wing geometry for example influence significantly the drag generated. Wingtip vortices form the primary component of wake turbulence and it is crucial understanding this particular phenomenon. To have better knowledge of wingtip vortices we show in the following pictures how they appears to our eyes in two specific situations. (Figure 3.1, Figure 3.2)



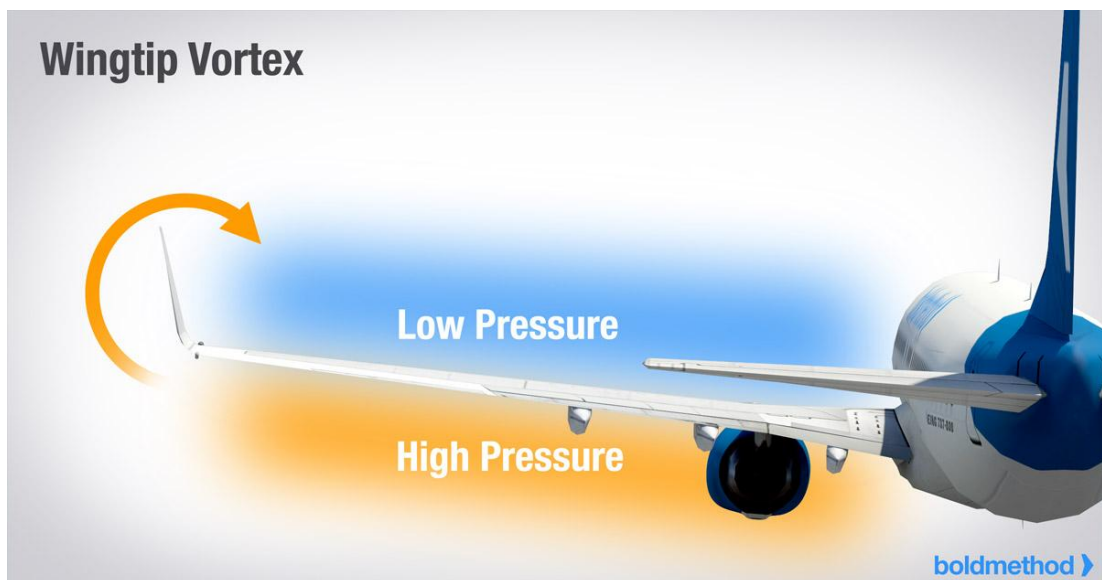
[6]

Figure 3.1



Figure 3.2

We said that wingtip vortices occur when a wing generate a force called Lift. What happens in details is that air from below the wing, is drawn around the wingtip into the region above the wing by the lower pressure, causing a vortex to trail from each wingtip. Wake turbulence exists in the vortex flow behind the wing and its strength is determined primarily by the weight and airspeed of the aircraft. The fluid movement is caused by the natural tendency of air trying to "compensate" sliding from high pressure zones to lower pressure zones as it's shown in figure 3.3.



[8]Figure 3.3

It is not interest of this thesis to explain the physic laws behind wingtip vortices, considering our goal it's sufficient to have an idea of what they are and why they represent a problem. All the data are provided by the P2P model, we use them as an input to create the 3D simulation.

3.2 P2P Program

In the introduction section of this thesis we described briefly what is the P2P model. Since it comprehends many difficult aerodynamic equations, we won't go deep into the details of the model itself, but we will focus more on what kind of data the P2P program produce in order to understand how the 3D Virtual Airport exploit them. P2P model has been developed by Dr. Ing. Frank Holzäpfel from the DLR, German Aerospace Research Center in 2003 and it differentiates from the previous wake vortices models thanks to the probabilistic component. In fact P2P accounts for the effect of wind, turbulence, stable stratification and ground proximity, but it consider as well probabilistic aspects such as:

- Deviation from deterministic vortex behavior caused by the stochastic nature of turbulence
- Vortex instabilities and deformations
- Uncertainties and fluctuations that arise from environment and aircraft parameters

Considering all these aspects it develops confidence intervals of wingtip vortices intensity and position evolving according to time. The P2P has been implemented in Fortran and to understand more how it works, we can see in figure 3.4 the program structure and workflow

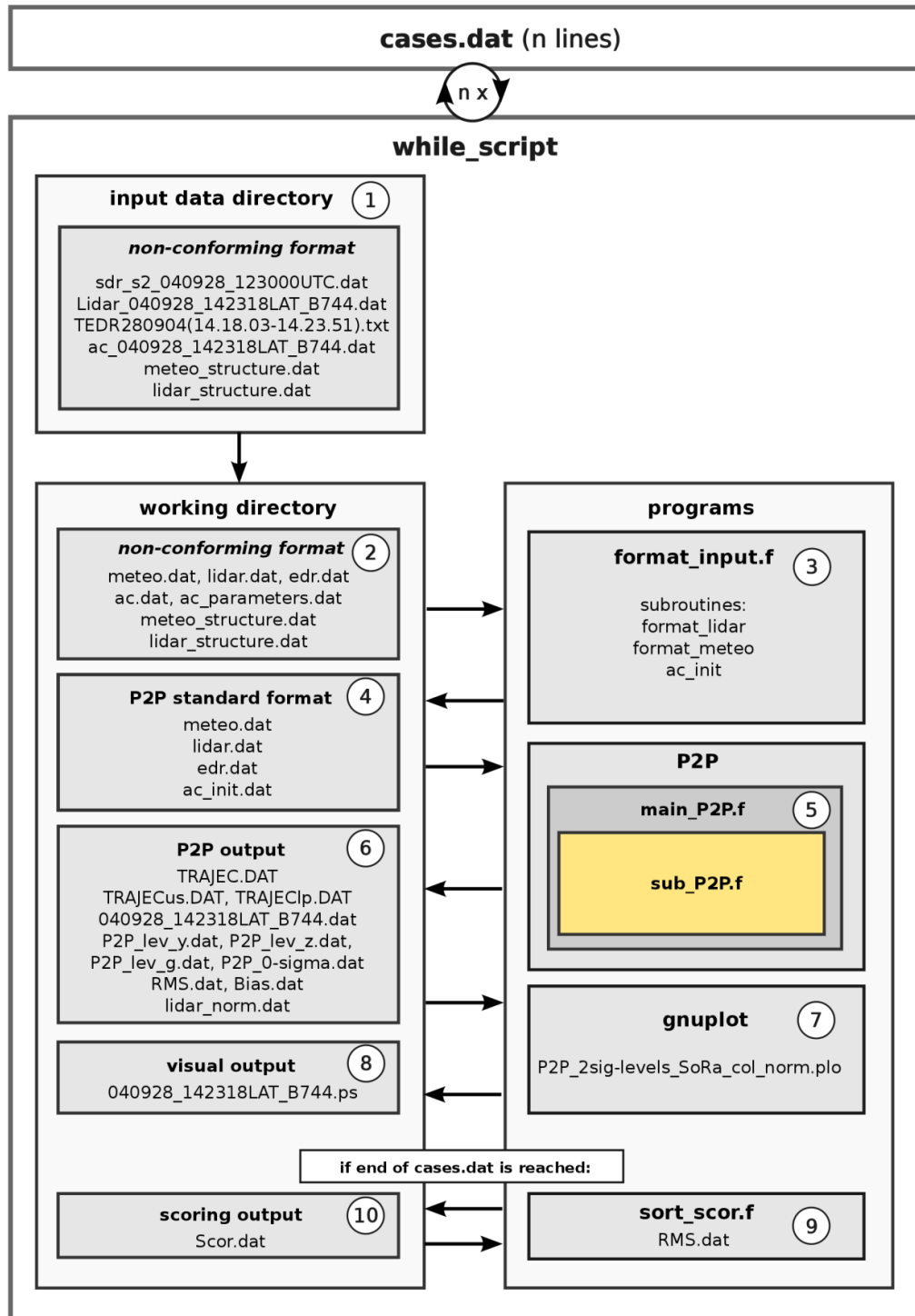


Figure 3.4 [9]

The program takes as input the following files:

- `cases.dat`
- `lidar.dat`
- `meteo.dat`
- `ac_init.dat`
- `ac.dat`

- ac_parameters.dat

All of them contain data and information necessary to the P2P model to generate the wake vortices prevision. Going to deep inside the content of these files fall outside the purposes of this thesis. The output files are in the format "yymmdd_hhmmss_UTC_xxx.dat" and contain the coordinates of the vortex position and the circulation parameter of the vortex itself, all related to a certain time.

3.3 Game engines and Unity 3D

The word "game engine" started arousing in the middle of the 90's with the first person shooter game "Doom" developed by "Id Software". This game was designed with a marked separation between the software core and other components like the art material, the world scenario and the rules of game itself. The importance of this separation line, became more clear when developers started producing new games modifying just the art assets such as the 3D world, characters, weapons, structures, and leaving the engine core software almost unmodified. This is briefly how the development and success of game engines began. Nowadays, game developers can create their own game engine framework and reuse significant part of it to develop and produce new games. This process brings to a remarkable save of time and money. Unity3D is of the multitude game engines available on the market. It is a multiplatform game engine developed by Unity Technologies in 2005 when the first release came out [10]. It supports both 2D and 3D graphics and the scripting languages used are C# and Javascript, and targets graphic API libraries like:

- Direct3D and Vulcan for Windows and Xbox
- OpenGL for Linux, Mac and Windows
- OpenGL ES for Android and iOS

For what concern the 3D development, Unity supports different compression method for textures and it gives support for normal maps, ambient occlusion and reflection maps. One of the major strength point of Unity is the support to create builds for many platforms like: Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft Hololens, Nintendo 3DS line,^{[11][12][13]} Nintendo Switch,^[14] Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS,

Wii, Wii U, Windows, Windows Phone, Windows Store, WebGL, Xbox 360, and Xbox One [10]

3.4 Unity 3D in scientific visualization

So far we analyzed how the game engines arose and why they had such a big success. Their first, and at the beginning the only, purpose was obviously to support developers in games production, making their job faster and easier. However some other important aspects need to be considered. "Effective visualization is the bridge between quantitative information and human intuition" [11] so, having tools able to easily represent complex scientific information, could be of great use in many different fields and could start a new era in scientific visualization. From these premises researchers started investigating if it could be possible to exploit game engines functionalities to obtain a nice, and useful, visualization of their projects. Unity3D itself crossed this path and it has been object of investigation in different scientific area with good results. In 2013 Marc Baaden's research team of the institute of CNRS in Paris, developed an application called "UnityMol", which is a molecular editor viewer and prototyping platform [12] implemented in C# using Unity3D. This application is able to read Protein Data Bank files, Cytoscape networks, OpenDX potential maps and Wavefront OBJ meshes. [12] In figure 3.5 we can see a sample of how UnityMol visualization appears.

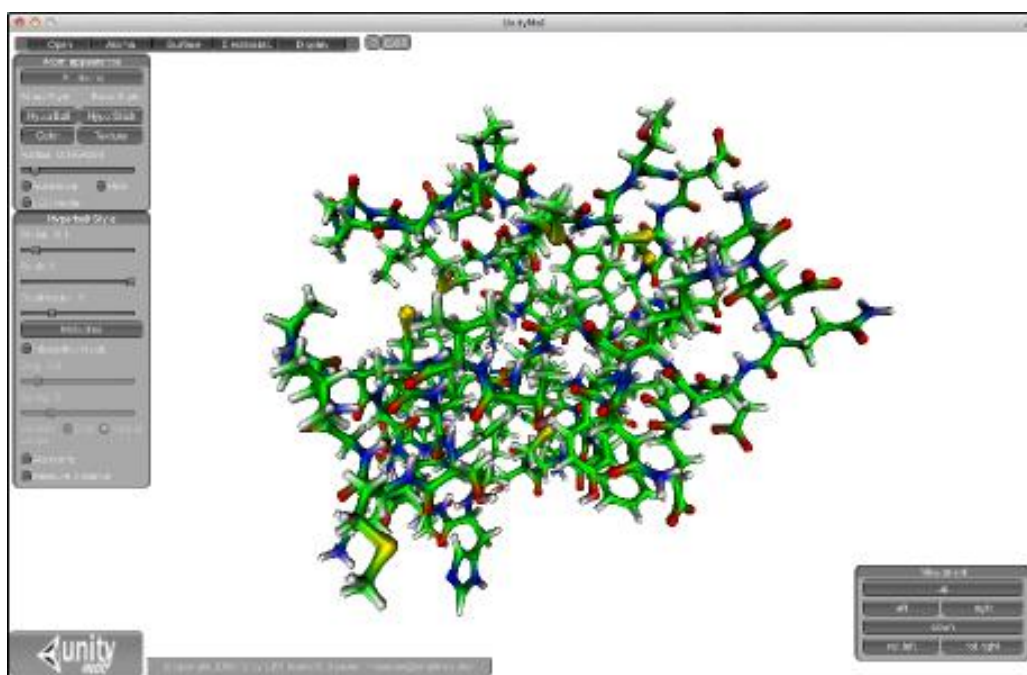


Figure 3.5

At California institute of technology in 2013 they started investigating about using Unity3D in full immersive virtual world to solve "Big" data [13]. According to Alex Cioc article, the prototype developed can render 100.000 data objects in about 15 seconds on a mid-2011 Macbook Air.

Considering these premises we thought that starting a project like the visualization of wake vortices, using Unity3D as graphic tool, could lead us to good results.

4 Development

In this chapter of the thesis we will analyze in details the whole developing process that led us to 3D Virtual Airport, starting from zero until the final application.

4.1 3D world scenario

The first step to develop our scientific simulation was to give birth to the 3D world scenario, where the visualization had to take place. To reach this goal we could have used pre-made free to use meshes and save lot of time, but we considered it a bad option since we wouldn't had control on the number of polygons that compose the 3D models. In real-time application, low latency response is crucial, and this can be reached (together with other considerations) keeping the meshes as much "light" as you can. For this reason we created all the 3D models by ourselves focusing our attention on having 5000 polygons per model maximum, a good trade between low latency and stressing a bit the Unity3D potential. To create the meshes we chose to use a 3D modeling software called Blender for different reasons that we will briefly discuss.



- **Open source:** like many free software, Blender has a really active community ready to help you when you are facing a problem. And with a high probability the issue you have has already been faced by someone else and you can find the solution really quickly.
- **Previous knowledge:** for different university exams I used Blender and I was already confident with tool's shortcuts. This was obviously the most important reason that brought us choosing this software.

- **Good integration with unity:** Blender is perfectly compatible with unity 3D and the integration of the models is easy and fast.
- **Powerful tool:** Last but don't the least, Blender is a really powerful tool with many functionalities that made it suitable for our purpose.

Once all the model were ready the second actor, Unity3D, comes into play, and it is fundamental to understand. why we chose this specific game engines out of the multitude of products available on the market.



- **Not just for gaming:** Unity has already been used to develop scientific visualization and it has already partially proved that it can be useful for these applications.
- **Multi-platform:** This an amazing pro that makes Unity formidable. Once you create your world and finish the application, you can build the project to execute it in many different platform like Windows, Linux, Android, IOS, Sony Playstation, Xbox, Web application, just to name some.
- **Third party devices:** It has a good integration with external devices like Oculus Rift and Kinect
- **Integration with Blender:** It integrates easily with Blender, in fact the 3D model created using Blender are recognized by Unity just pasting them in a specific folder inside our project.
- **Active community:** More than Blender, Unity has a vast community and tutorials that are really helpful during your developing process
- **Asset store:** Within the unity environment there is an Asset store where you can buy, or download for free depending on what you need, premade objects suitable for your project.
- **IDE:** The last important aspect that we considered is that Unity has a really nice and user friendly environment. The cooperation between the editor and the scripting side is handy and not too hard to learn for beginners.

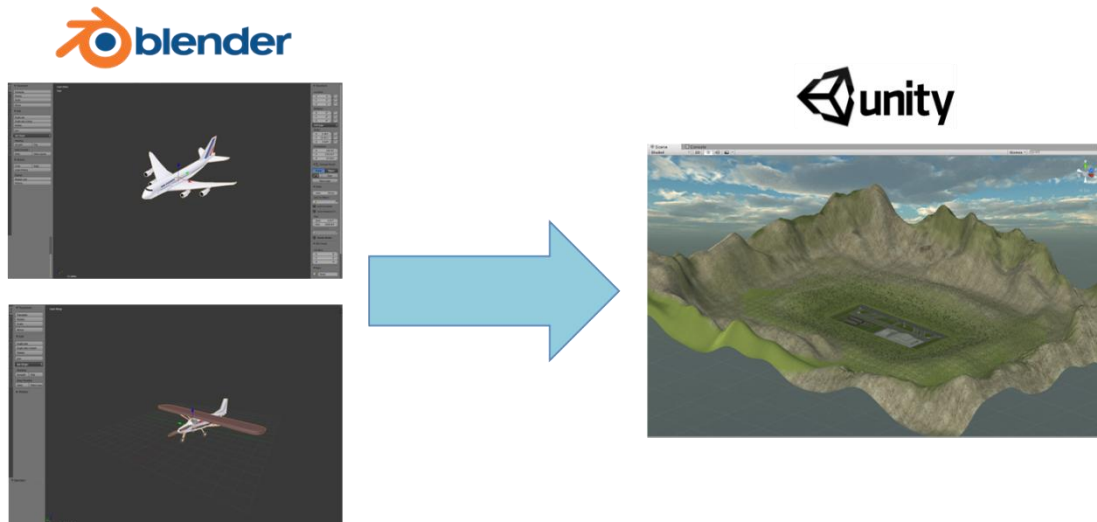


Figure 4.1

Exploiting the cooperation between these tools we have been able to reach our goals and create the 3D Virtual Airport.

4.2 Modeling process

Creating the scenario for the simulation took time and effort. However it was crucial to have a solid base to build on the scientific visualization. The first step was to model all the 3D Objects that will take part to the simulation using, as we already discussed in the previous paragraph, Blender. We started building the landing strip of our virtual airport followed by basic structures like airplanes, hangars, control tower, to make the scenario looking like a real airport.



Figures 4.2 (Mesh modeling samples)

Standard techniques have been used to design the meshes like extrusions, avoid unconnected faces and mirroring. All the meshes are conceived as much low polygons as possible without losing photo-realism. The second part was the texturing phase. Good texturing is essential to have a nice looking scenario, for this reason we applied for each one of the model in the scene a double layer texture: a diffuse

texture plus a normal map texture. Diffuse textures are simple images that we attach as wallpaper to the mesh and they look flat unrealistic if we use just them standalone.



Figure 4.3 (Diffuse texture sample)

Normal maps instead are not just simple images. They are represented using the RGB format (Figure 4.4), but they give information about how the light will be reflected by the mesh. With this method we can reach a realistic effect without increasing the number of polygons that compose the mesh (Figure 4.5)

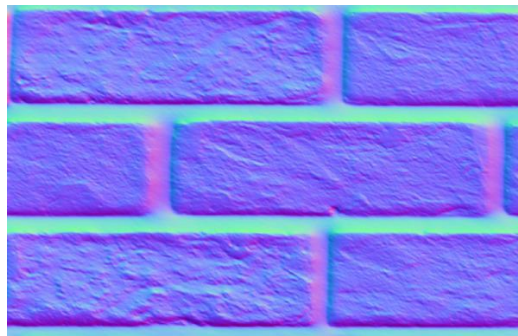


Figure 4.4 (Normal map sample)

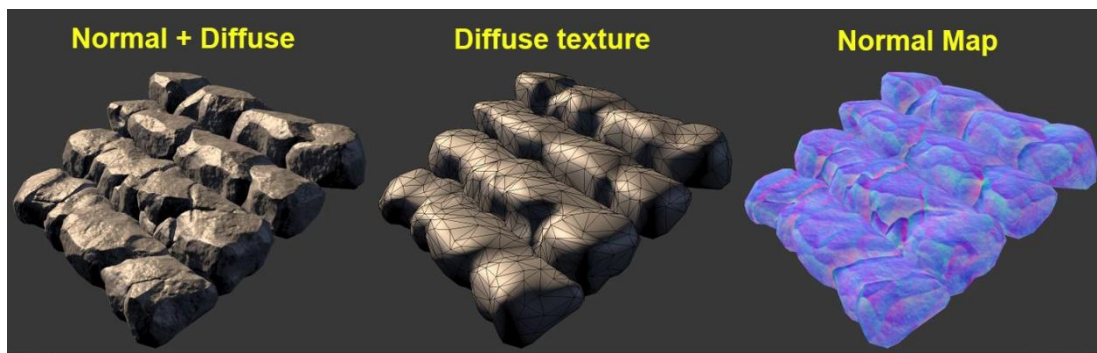


Figure 4.5

The third step was to create the terrain of the scenario and the integration of the objects we designed into the Unity3D world. The terrain has been created all in unity directly because the editor contains specific and very useful tools for terrains and lands design. For what concerns the integration of all we had to import the models in Unity and link all the textures we used in order to allow Unity to recognize them and tell which kind of texture it is. Once all the static objects were set in the right

position in the 3D space, we saved the configuration and, after also the light were set, we started the process of "lights baking". This mechanism is really useful and consists in calculating all the shadows cast by the stationary items, avoiding to calculate them at real time, saving precious time. The final result can be seen in Figure 4.6

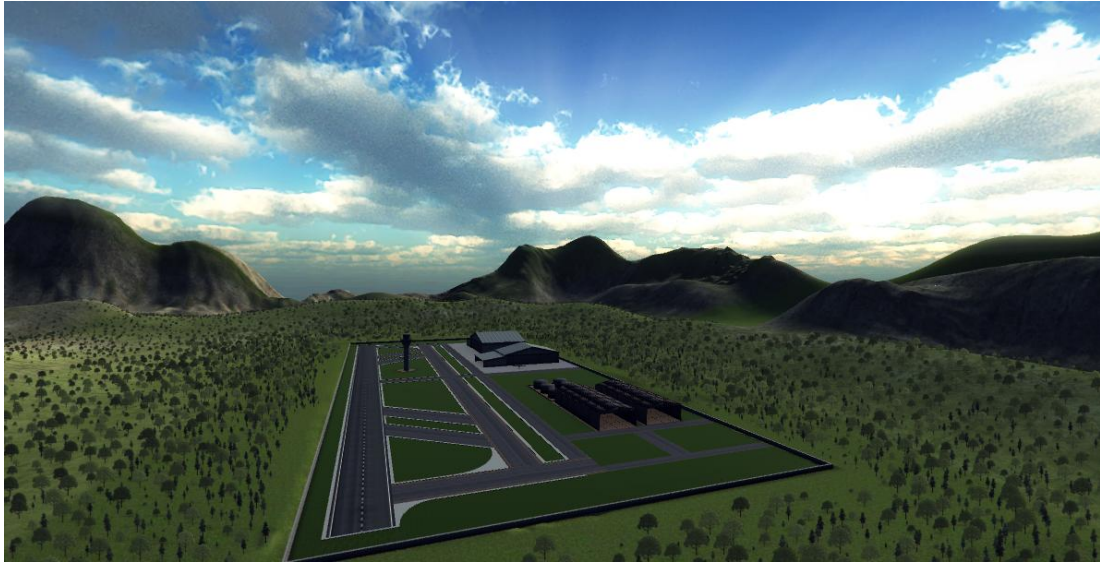


Figure 4.6

4.3 Structure and workflow

Virtual 3D Airport has a linear structure designed to be scalable imagining future developments that the application could have. It consists in five actors cooperating together to give birth to the simulation. In figure 4.7 are represented these five pilasters and the data flow from the beginning until the graphic output.

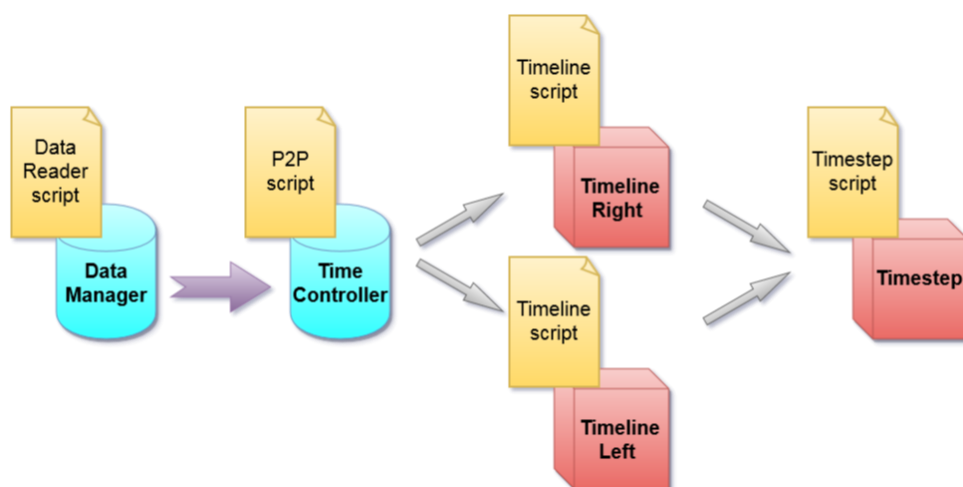


Figure 4.7

The objects in this picture represent objects in Unity. To give a logical behavior and to make them "do something", we must attach scripts to them. In the scripts we code the laws that govern that specific object. Here we have two different type of items:

- **Standard object**

Represented as light blue cylinders. They are unique in the scene

- **Prefabs**

Represented as red cubes. They are items that appear many times inside a scene. A typical example of prefab can be a bullet inside a shooter game. You create the object one time as a prefab, and you can re-use as many times you want.

The **DataManager** had been designed as a standard item and its role is to read the output file created by the P2P program, convert the data in a specific format comprehensible by the TimeController.

The **TimeController** reads the position and the intensity data of the wingtip vortices and assigns the values to the corresponding time-step in order to have, for each step of time, an exact value of the position in the 3D space and an exact value of the intensity of the vortex.

The **Timelines** represents the vortices, one for each wing. They evolve during time according to the data received from the P2P program. Figure 4.8 shows an example of the timelines.

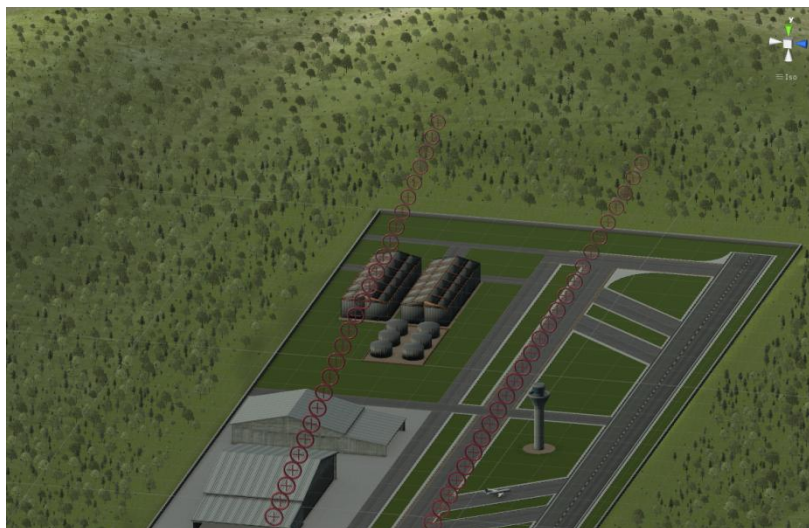


Figure 4.8

The **Timestep** is the basic tassel that compose the Timeline. Each one of them represent the intensity and position of the vortex at that exact time. It has been designed to be able to represent different degrees of information. It has two

independent lines, one horizontal and other vertical, that form a kind of "+" symbol. Around it there is a ring that can vary its radius and thickness, and in the end the Timestep has a color that represent in our case the strength of the vortex (Figure 4.9) With all these possibilities the timestep can represent five dimensions of information, but for our purposes we will just represent intensity of the wake vortices and how it decays over time.



Figure 4.9

4.4 Animation scripts

We already mentioned that in Unity to animate and give some logic to a certain object, you need to attach a script to that item. Scripts can be written in two different languages; C# or Javascript. We chose to code the script in C# because both me and my supervisor (Dipl. Ing. Wito Engelke), are more confident with it respect to Javascript. In this paragraph we will look in details all the scripts that compose the application 3D Virtual Airport

DataReader Script

It is the responsible of reading the output file of the P2P program and store the information in order to be used by the TimeController script.


```

using System.IO;
using System;

public class DataReader : MonoBehaviour
{
    public string file;

    private List<float[]> trajec = new List<float[]>();
    private List<float[]> complete_data = new List<float[]>();

    private const int TRAJEC_FIELDS = 7;
    private const int COMPLETE_DATA_FIELDS = 13;

    void Awake()...

    private void ReadFile(string fileName)...

    /* ...

    public List<float[]> getFile(string fileName)...

    bool isCompleteDataFile(string s)...

    void printList(List<float[]> ls)...
}

```

In this snapshot of code we can see the fields and methods of the DataReader script. The two private fields "trajec" and "complete_data" represent the two different input formats that the P2P program can send as output to 3D Virtual Airport, which is able to read both of them without any issue. So depending on the input, the information about the vortices are stored in the corresponding buffer.

The function "ReadFile" recognize which input it's arriving and read the whole file saving all the data into the list. Each element of the list is an array of float which corresponds to one line of the input file.

130429_053250UTC_HALO.dat

	t	yp1	ypm	ysl	ysm	zpl	zpm	zsl	zsm	Gp1	Gpm	Gsl	Gsm
1	22.38	161.59											
2	4.54	-16.30	-7.34	8.62	16.08	24.34	21.32	24.33	21.32	185.91	158.00	197.28	158.00
3	8.42	-13.01	-3.00	12.37	21.38	20.15	19.06	21.93	19.06	181.88	155.87	176.12	155.87
4	15.80	-4.83	7.13	22.65	37.41	18.58	14.99	20.11	14.79	169.43	150.14	142.68	150.14
5	19.25	4.72	10.26	35.56	44.35	16.77	14.24	16.89	15.31	144.16	147.52	142.82	147.52
6	26.93	19.08	18.01	59.80	59.12	16.24	13.06	15.68	18.52	129.65	137.57	127.32	137.57
7	30.17	33.09	20.56	72.23	64.04	15.60	12.84	15.66	18.78	129.93	132.88	126.77	132.88
8	38.01	36.82	28.24	83.82	80.93	14.98	14.21	15.55	19.92	126.44	119.11	118.90	119.11
9	41.14	35.99	31.79	95.07	87.19	14.96	14.75	16.16	21.31	130.24	113.76	93.44	113.76
10	49.02	38.00	42.03	107.89	101.73	14.83	15.42	17.64	24.55	115.18	99.66	96.99	99.66
11	52.16	40.67	45.66	112.98	106.77	15.72	15.53	17.67	25.33	115.54	94.54	79.40	94.54
12	59.95	40.59	55.44		120.50	16.35	16.77		26.17	111.78	81.52		81.52
13	63.25	41.00	59.44		125.89	17.88	17.31		26.28	114.08	76.90		76.90
14	70.91	43.90	70.81		140.80	17.47	18.50		26.75	98.55	65.30		65.30
15	74.19	49.64	75.25		146.49	16.46	18.84		27.07	96.94	61.21		61.21
16	82.02	54.74	87.33		161.49	15.10	19.49		28.15	87.86	50.96		50.96
17	85.12	62.85	91.88		167.04	15.31	19.66		28.60	94.19	47.35		47.35

Figure 4.10

Considering the file shown in Figure 4.10 for example, the first element of the list will contain an array composed by the values "4.54, -7.34, 16.08, 21.32, 158.00, 158.00". In order these values represent:

- Time

- Y position of the vortex released by the left wing (predicted by P2P model)
- Y position of the vortex released by the right wing (predicted by P2P model)
- Z position of the vortex released by the left wing (predicted by P2P model)
- Z position of the vortex released by the right wing (predicted by P2P model)
- Intensity (circulation) of the vortex released by the left wing (predicted by P2P model)
- Intensity (circulation) of the vortex released by the right wing (predicted by P2P model)

all the other values comes from the Lidar measurements and are not taken in account by 3D Virtual Airport. Here is very important to understand to what coordinates system these Z and Y values refer to. (Figure 4.11)

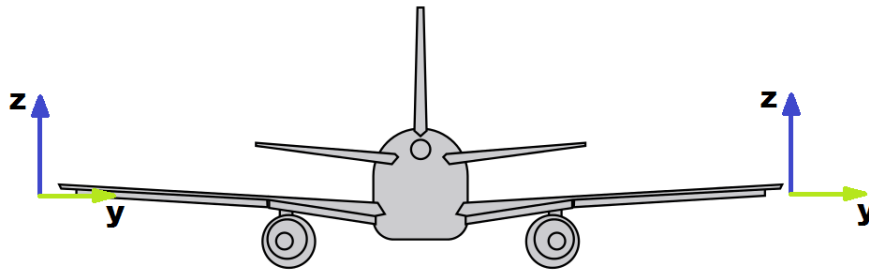


Figure 4.11

P2P Script

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class P2P : MonoBehaviour
{
    public Timeline timeline_left;
    public Timeline timeline_right;
    public Transform target;

    private int lerp_timestep = 0;
    private int current_timestep = 0;
    private float current_time = 0;
    private bool isStarted = false;

    private float starboard_circulation = 0;
    private float starboard_z = 26f;

    private float port_circulation = 0;

    private DataReader data;
    private List<float[]> file;
    private Dictionary<string, List<float>> steps;
    private Dictionary<int, Vector2> normalizer_values;
    private const float SCALE_FACTOR = 1f;

    void Start()...

    void FixedUpdate()...

    void normalizeColumn(List<float[]> values, int index)...)

    float circulationMapping(float circulation, int index)...)

    bool isClose(float time, float simulation_time)...)

    float approximateTimestep(float time)...)

    void findMinMaxValues(List<float[]> fileName)...)

    List<float> findLerpSteps(float initial, float final, float time)...

    void refreshTimelines()...
    void initializeValues()...

    void normalizeValues()...

    void scaleMetersToUnits(List<float[]> values, int index)...)

    void scaleValues()...
}
```

In this snapshot we see all the private fields and the methods implemented in the P2P script attached to the TimeController object. In the "start" section all the values read by the DataManager are initialized. But what does it mean? It means that all the values represented of interest represented in Figure 4.10 are normalized in between a the range [0-1] thanks to the function "normalizeValues". Then the method "initializeValues" takes the first two rows of values and applies the function "findLerpsSteps" to the first two sets of values corresponding to the first two time steps. The function "findLerpsSteps" applies a linear interpolation between the values to obtain a single value every 0.5 seconds

The function FixedUpdate is part of the unity framework and it is called every fixed framerate frame. Inside this method when the timestep correspond to the 0.5 seconds interval, a new tassel of the vortex timeline is added and the whole timeline is refreshed. In the end the function "approximateTimestep" exploit the method "isClose" to round the time values to decimal. All the other functions are just for support and not meaningful to comprehend how 3D Virtual Airport works.

Timeline Script

```
using UnityEngine;

using System.Collections;
using System.Collections.Generic;

public class Timeline : MonoBehaviour
{
    public List<Timestep> timelineList; // List that holds a reference to each tim
    public Transform target;
    public Transform spawnPort;
    public Transform spawnStarboard;
    public string wing;
    public int timestep_number = 1;
    public bool linear_transforms = false;
    public bool exp_scaling = false;
    public bool show_rings = false;
    public float intensity = 0.0f;
    public float ls_horizontal_factor = 1.0f;
    public float ls_vertical_factor = 1.0f;

    private float old_intensity = 1.0f;
    private float old_ls_horizontal_factor = 1.0f;
    private float old_ls_vertical_factor = 1.0f;
    private int old_timestep_number = 1;

    void Awake()...

    void Update()...

    void TimestepUpdate()...

    void LinearScaling(string direction)...)

    void LinearColor()...

    public void ChangeColor(Timestep tsp, float intensity)...)

    void showRings(bool show)...)

    public void Reset()...

    bool ChangeOccured()...

    public List<Timestep> getChildren()...

    public void setPosition(Vector3 starboard, Vector3 port)...)

    public int length()...
}
```

This script is quite simple despite the number of functions that implements. It simply reacts to any change that happens in the Timeline, such as increase or decrease of the Timestep number, or change of the Circulation of the vortex. The function "ChangeOccured" works as a monitor that triggers the method "TimestepUpdate"

just when there is need to refresh the Timeline. The "TimestepUpdate" scan the whole timeline and change the value of each single timestep as required. All this we discussed is obviously valid for both the objects TimelineLeft and TimelineRight since they both have a Timeline script attached.

Timestep Script

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class Timestep : MonoBehaviour
{
    public Transform tassel_vertical, tassel_horizontal;
    public Transform ring;
    public Transform torus;
    public TorusGenerator torusGen;
    public float vertical_scale = 1;
    public float horizontal_scale = 1;

    public Color color_RGBA = new Color(1.0f, 1.0f, 1.0f, 1.0f);
    public bool show_ring = false;

    private Dictionary<string, Transform> children;
    private Renderer torusRend, tasselVertRend, tasselHorRend;

    void Awake()...

    void Start()...

    void Update()...

    public Color ColorMapping(float i)...)

    public Dictionary<string, Transform> getTimestampComponents()...

    public void setHeight(float value)...)

    public void setWidth(float value)...)

    public void setThickness(float value)...)

    public void setHWT(float value)...)
}
```

This script allows to modify the properties of the single timestep. It is always called automatically by upper level scripts.

4.5 Networking

When the application was finished and ready to run, me and my supervisor Wito Engelke, decided to do another step forward questioning the possibility to make 3D Virtual Airport run on a Local Area Network. This idea bumped in our minds because we wanted to see our simulation running on the Powerwall screen of the virtual reality laboratory in the DLR (where I developed the thesis). The Powerwall configuration is shown in figure 4.12.

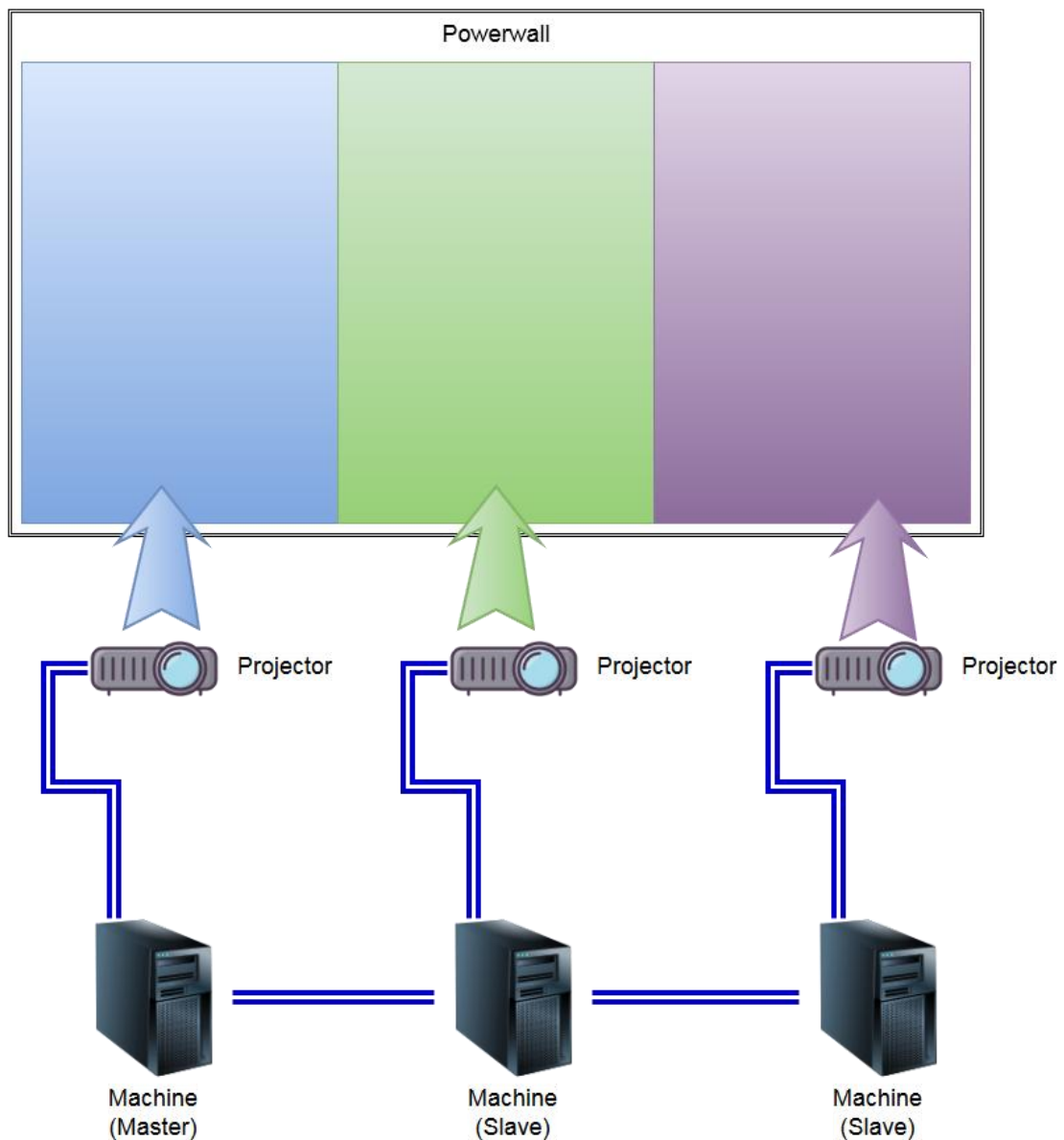


Figure 4.12

So to make the three different instances of the application synchronizing and exchange information, we had to implement a NetworkManager object in Unity able to take care of all this aspects. To keep things initially simple we chose to have a "light" scene that didn't include the vortices simulation avoiding possible network

delays. Using the premade NetworkManager tool of unity we just needed to attach to it a script to create a basic GUI and the application was ready. We tested it using three different machines and three different screens with good results. In figure 4.13 and 4.14 we can see first the application GUI setup, in the second the application running on three windows on single localhost machine.

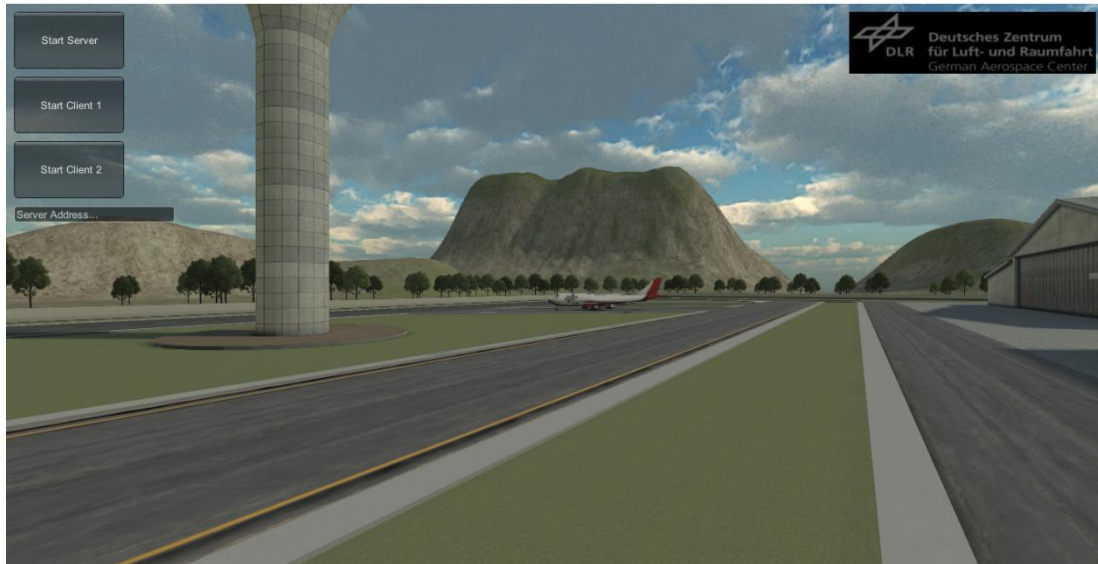


Figure 4.13

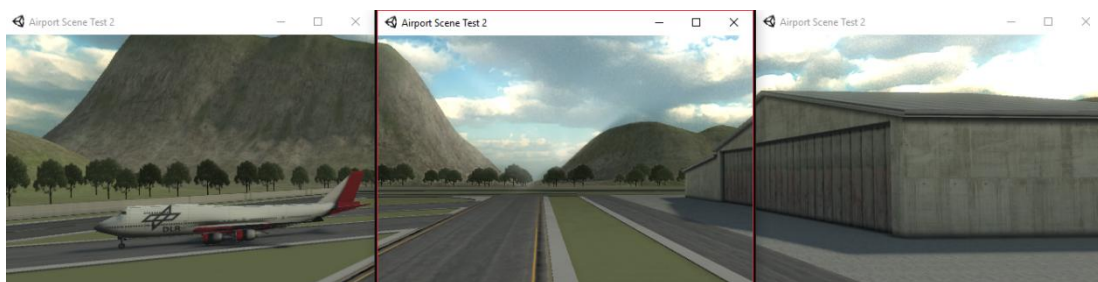


Figure 4.14

5 Use case

5.1 Application set up

Virtual 3D Airport runs inside the Unity editor, it hasn't been built to be execute in standalone mode yet. In order to run the simulation some set up steps must be performed to ensure the expected result. The first thing is to pass to the DataManager the output file of the P2P program. To do this, the mentioned file has to be located in the "Data" folder which is inside the default Unity project location "Assets" as we can see in Figure 5.1.

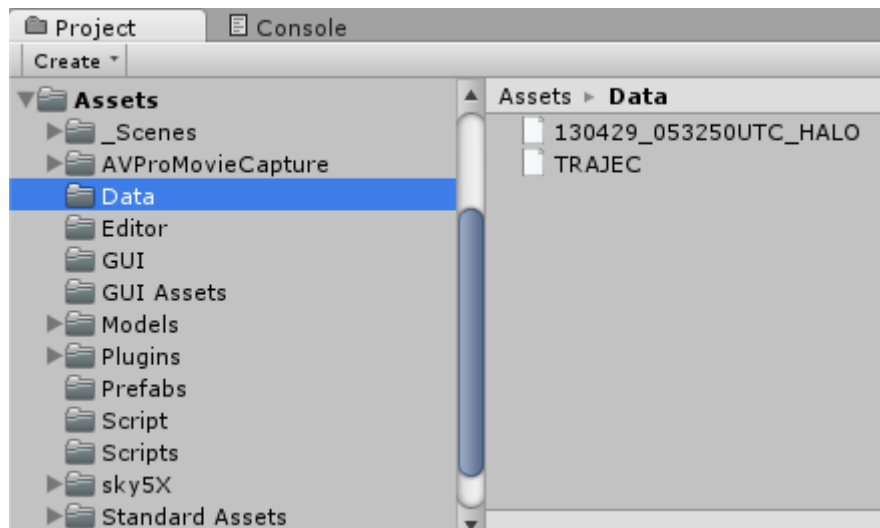


Figure 5.1

Then simply the file can be dragged and dropped inside the designed box area of our DataManager item (Figure 5.2)

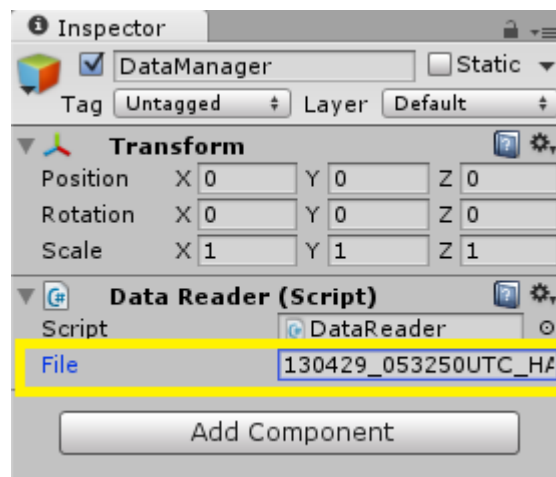


Figure 5.2

It is important that the filename corresponds to one of the two names that the P2P program generates as output. So the accepted filename formats are "TRAJEC.dat" or "yymmdd_hhmmss_UTC_xxx.dat" where the first six characters correspond to the date, the second six to the time and the last three can be random.

The second step is to configure the TimeController. We already explained that the TimeController item has the P2P script attached and what and all its functionalities. To do its job this item needs to know three things:

- Who is the Right Vortex
- Who is the Left Vortex
- Who is the aircraft that produce the wingtip vortices

To retrieve all these information we need to drag and drop in the TimeController boxes the corresponding items as it is shown in Figure 5.3.

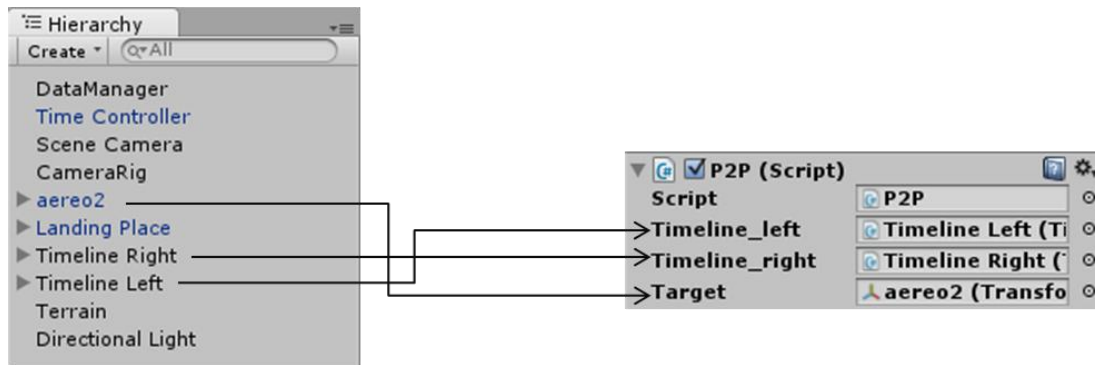


Figure 5.3

The last operation to perform is on both the Timeline objects and it is again a drag and drop action to set the reference of who is the airplane as we just did for the TimeController. Once all these simple steps are done the simulation is ready to run showing how the vortices evolves during time.

5.2 User interface and screenshots

In this section we will see how the user interface looks like and some screenshot of the simulation running will be shown. Let's start with an overview of the Unity editor in the context of 3D Virtual Airport (Figure 5.4).

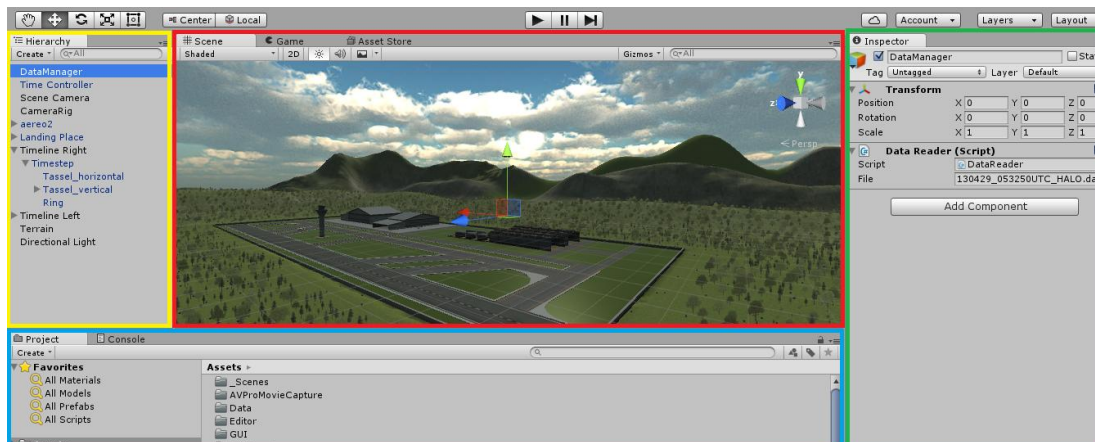


Figure 5.4

In the **Hierarchy tab** there is the list of all the objects (standard or prefabs), that compose the world scenario. The items can have a parent-child link that involves any geometric transformations affecting the parent, to be reflected on the child too. In 3D Virtual Airport an example is the relation between the Timeline item and the Timestep object. In the **Project tab** we are able to navigate through the application folders in order to retrieve all the assets imported in our program, like meshes, scripts, prefabs, store assets. On the right side we have the **Inspector view**, in which are listed all the details and subpart of the item selected in hierarchy tab. The **central**

window is the 3D world itself. In the "scene" tab we can see how our scenario appears while it switches to the "game" tab as soon as we start the simulation.

In the following pictures (Figures 5.5, 5.6) we can appreciate two different point of views taken after 20 seconds of simulation.



Figure 5.5

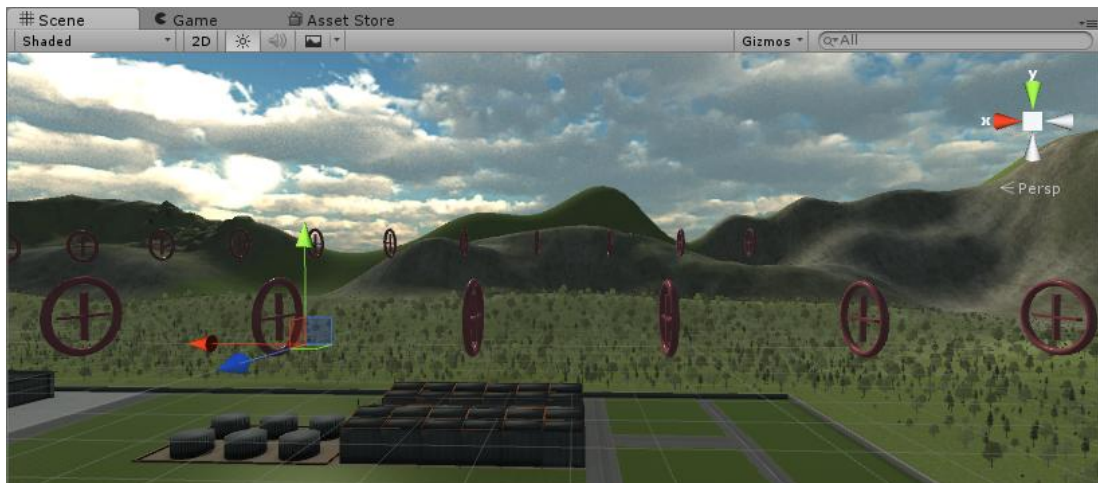


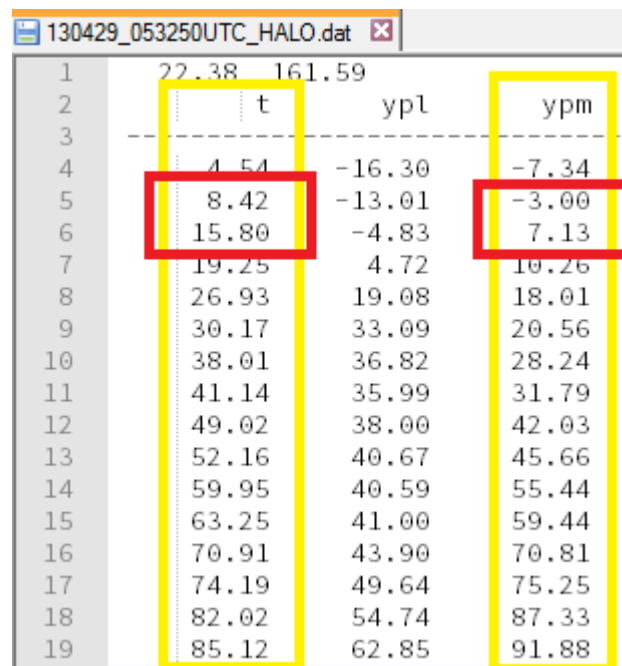
Figure 5.6

In the picture 5.5 we are just behind the aircraft and we are looking in the opposite direction of movement. The vortex decay phase is not dominant yet and the timesteps still appear dark red. In the image 5.6 instead, we look at the first seconds of simulation and, considered that 20 seconds already passed, we see how the shade of red is turning pink, to represent that the wingtip vortices are slowly fading.

5.3 Data congruence validation

The last step to consider our project done successfully, was to check that what we can see in the visual representation of the wake vortices, was congruent with the output data developed by the P2P model program. To this we analyzed eleven

different out files of the prediction model and we used them as input for 3D Virtual Airport. In each one of these instances we noticed significant congruencies between, the position of the vortices calculated by the mathematical model, and where the vortices were located in our simulation. The second match is in relation with the trend of the vortices circulation. As we already explained, according to the P2P model, the wingtip vortices pass through a first phase of stabilization and a second phase of rapid decay. This aspect is obviously reflected by the data output developed by the P2P program and it also has a confirmation in our graphic visualization. To understand all this some pictured are needed:



	t	ypl	ypm
1	22.38	161.59	
2			
3			
4	4.54	-16.30	-7.34
5	8.42	-13.01	-3.00
6	15.80	-4.83	7.13
7	19.25	4.72	10.26
8	26.93	19.08	18.01
9	30.17	33.09	20.56
10	38.01	36.82	28.24
11	41.14	35.99	31.79
12	49.02	38.00	42.03
13	52.16	40.67	45.66
14	59.95	40.59	55.44
15	63.25	41.00	59.44
16	70.91	43.90	70.81
17	74.19	49.64	75.25
18	82.02	54.74	87.33
19	85.12	62.85	91.88

Figure 5.7

In figure 5.7 we have a detail of the P2P output file called "130429_053250UTC_HALO.dat". In the highlighted sections we can notice how, in between 8.42 and 15.8 seconds, the movement of the left wing vortex invert its trend and it starts moving along the positive Y direction. In figure 5.8 we see how this trend change is reflected in the graphic visualization calculated by 3D Virtual Airport.

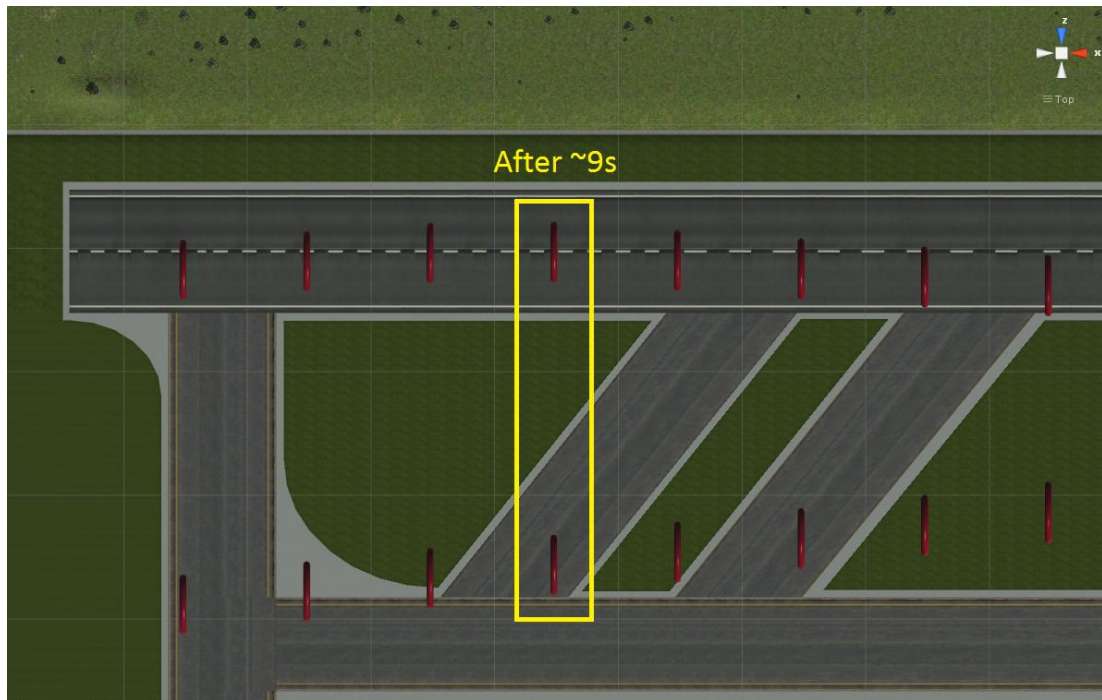


Figure 5.8

The other observed validation refers to the circulation decay after the initial stabilization part.

130429_053250UTC_HALO.dat				
1	22.38	161		
2	t	Gpm	Gsl	Gsm
3				
4	4.54	158.00	197.28	158.00
5	8.42	155.87	176.12	155.87
6	15.80	150.14	142.68	150.14
7	19.25	147.52	142.82	147.52
8	26.93	137.57	127.32	137.57
9	30.17	132.88	126.77	132.88
10	38.01	119.11	118.90	119.11
11	41.14	113.76	93.44	113.76
12	49.02	99.66	96.99	99.66
13	52.16	94.54	79.40	94.54
14	59.95	81.52	NaN	81.52
15	63.25	76.90	NaN	76.90
16	70.91	65.30	NaN	65.30
17	74.19	61.21	NaN	61.21
18	82.02	50.96	NaN	50.96
19	85.12	47.35	NaN	47.35

Figure5.9

Taking the same sample file we notice that, after about 35 seconds, the circulation starts decreasing rapidly according to the P2P model. In our simulation we can see that, after 30s of simulation, the color of the timesteps start turning from red to pink and then slowly almost into white according to how the system has been designed. (Figure 5.10)

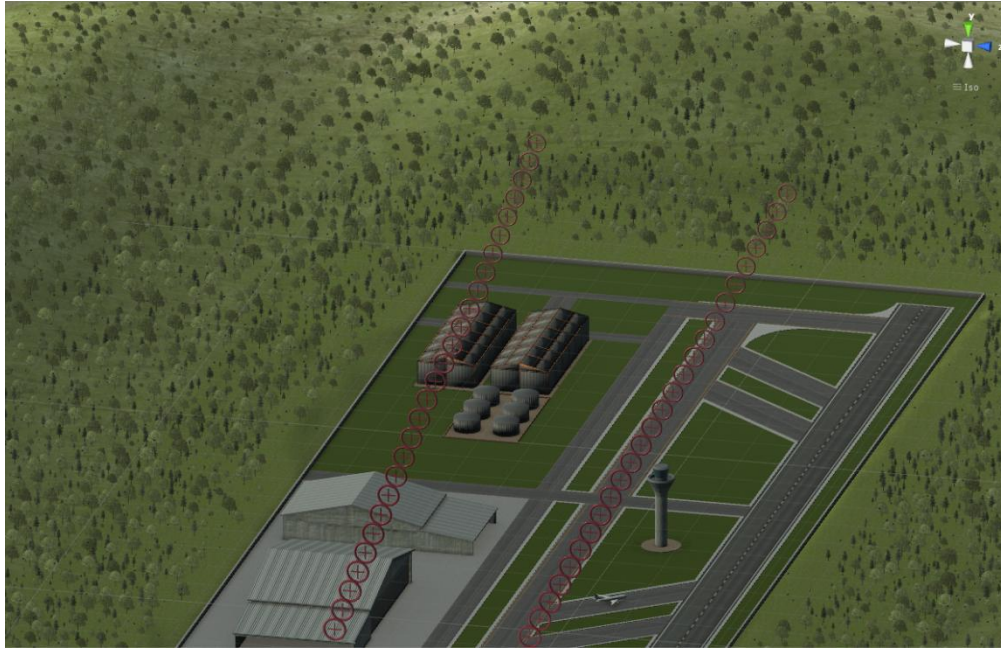


Figure 5.10

6 Results and conclusions

6.1 Results according to goals

At the beginning of this project our goals were well clear; we wanted to test the game engine Unity3D exploiting its functionalities to develop a scientific visualization. We also aimed to create a useful tool able to increase airports capacity and improve the management of air traffic control. To reach all these goals we implemented a 3D representation of a complex aerodynamic effect, such as the wake vortices, using Unity3D to give birth to the simulation. We started from zero and we managed to create a good looking 3D scenario of an airport. Then, cooperating with the P2P program developed by Dr. Ing. Frank Holz Äpfel, we have been to create different geometric primitives to obtain a nice and clear visualization of the wake vortices. All the primitives we created are scalable and reusable according to the game engine philosophy.

Looking at our simulation running bumps directly to the eyes what is happening on the landing strip, how the vortices are evolving with the passing of time, and when they are totally faded away and it is safe for another aircrafts to pass through that same path.

6.2 Can Unity3D used for scientific visualization?

At first, when Wito Engelke talked to me about this project I was a little bit skeptic about the data calculation part of the simulation. Then, when Wito told me that all the prevision data about the vortices would be calculated outside Unity, I started thinking that the project, developed in the proper way, could be a success. After this experience I feel confident saying that a tool like Unity3D can be used to develop scientific visualization, as long as it delegates the data calculation to another tool. I'm not saying that Unity3D can't do it, it's just that we didn't test it and we focused more on the visualization part. The information can be sent to Unity3D as a data stream or as buffer of text files for example; it can quickly interpret them and create the corresponding 3D visualization. The huge advantage of using this game engine for scientific purposes is its scalability and the possibility of reusing the primitives we created in the project. For example if in my future I need for a project to develop a visualization of sound waves, I can easily take the geometric primitives created in 3D Virtual Airport and adapt them to generate a new kind of simulation.

6.3 Future Developments

Every project can keep growing and new features can be added. Virtual 3D Airport is none the less and many nice features can be implemented for the future. The first thing that can be done is to show in the visualization the upper and lower bound of the prediction developed by the P2P model. As we already said P2P is a probabilistic model that designs interval of confidence; these intervals have a floor and a roof in which the vortices are located. So having a visual representation of both the vortices and their limits could be interesting. Another improvement that can be done to increase the Virtual 3D Airport potential is making it "real time" taking as input not just a single file but continuous stream of data. This particular feature should be developed cooperating with the authors of the P2P program because so far, it doesn't support such functionality. Unity3D supports external devices like Oculus Rift for instance. implementing in Virtual 3D Airport the integration for that kind of tool s for sure a nice improvement that can be added. Then, since the core of the networking part has already been implemented, it could be possible to integrate it in 3D Virtual Airport in order to be able to see the simulation running on a powerwall configuration such as that one of the virtual reality lab in the DLR in Braunschweig.

A neat future scenario that this application can face, is to be integrated in real functioning airports and be used to improve air traffic control. The core of the simulation is ready, the only new part that should be developed is creating the 3D scenario of the real airport.

Bibliography

- [1] C. LO, "<http://www.airport-technology.com/features/feature tackling-runway-wake-turbulence/>," 26 jun 2013. [Online].
- [2] L. (. Speijker, "http://www.wakenet.eu/fileadmin/user_upload/WS1/Topic4/WN3E_WS1_Topic4_1_Speijker.pdf," [Online].
- [3] F. Holzäpfel, "Probabilistic Two-Phase Wake Vortex," *Journal of Aircraft*, vol. 40, p. 1, 2003.
- [4] M. H. a. F.-E. W. Karl-Ingo Friese, "Using Game Engines for Visualization in scientific applications," *IFIP International Federation for Information Processing*.
- [5] L. Clancy, *Aerodynamics*.
- [6] NASA, "https://en.wikipedia.org/wiki/Wake_turbulence," [Online].
- [7] Reddit, "https://www.reddit.com/r/aviation/comments/1moh9w/airbus_a340_wingtip_vortices_1920x1200/," [Online].
- [8] Boldmethod, "<http://www.boldmethod.com/learn-to-fly/aerodynamics/winglets-and-wingtip-vortices/>," [Online].
- [9] Dr.-Ing. habil. Frank Holzäpfel, *Probabilistic Two-Phase Wake Vortex Model P2P*, 2014.
- [10] Wikipedia, "[https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))," [Online].
- [11] S. G. Djorgovski, "<https://www.microsoft.com/en-us/research/wp-content/uploads/2012/04/djorgovski.pdf>," [Online].
- [12] Baaden-Chavent, "<http://www.baaden.ibpc.fr/umol/>," 2013. [Online].
- [13] A. Cioc, "<http://www.astrobetter.com/blog/2013/03/25/immersing-yourself-in-your-data/>," 2013. [Online].