Design features of a VR software system for personnel training in aviation

Andrei Bulai, Diana Andronache, Dorin-Mircea Popovici

Ovidius University of Constanta

124 Mamaia Bd, 900527, Constanta Romania

bulai.andrei10@gmail.com; diana.andronache7@gmail.com; dmpopovici@univ-ovidius.ro

DOI: 10.37789/rochi.2020.1.1.20

ABSTRACT

This paper presents the steps that must be taken to develop a Simulator type software using VR technology and the presentation of our own software system for the training aviation personnel. We present the state of the art of VR and describe the developed application through the chosen solution, meeting all the criteria of a learning and testing environment for the aviation personnel, simulating the targeting operations of a helicopter and adjacent elements. In fact, we make a description of both the data structure and the technical design. Logical design and system architecture, use cases and conceptual classes are also presented. The technical part describes in detail the implementation stages of the application.

Author Keywords

Virtual reality; Virtual Simulation; Software development; Triggers and Gestures; User Interface for virtual reality; User Experience for virtual reality; 3D animation and modeling

ACM Classification Keywords

H.4.m. Miscellaneous; H.1.2. User/Machine Systems; D.2.m. Miscellaneous; D.2.10 Design; I.3.m. Miscellaneous; I.3.8; I.3.7; I.3.6; I.3.5

General Terms

Human Factors; Design; Measurement.

INTRODUCTION

At the moment, VR technology has gone from the early stage to the expansion stage to a small-scale on a large scale. The concept is well known, and the main equipment manufacturers that are part of VR systems are investing resources in the development and improvement of new technology. Therefore, today, the past VR engineering ideas can be restructured and homogenized both with the new market requirements and with the new equipment. This is why we afford to accept the challenge of developing a relatively low-cost training virtual environment dedicated to aviation personel.

Predecesors

In terms of aviation, since the first flight simulator in 1966, developed by military engineer Thomas Furness for the US Air Force [1], in a project to present the concept of VR, the market has evolved very quickly. Thus today we have the opportunity to use professional simulators both in terms of the complexity of simulating the real aviation environment and in terms of graphic quality.

Next we present the state of the art in the field of flight simulators, stating some of the most relevant projects launched on the market.

Microsoft Flight Simulator X – is a simulator produced by Aces Game Studio, being among the first VR simulators with advanced graphics. The first versions appeared in 2006, on the Microsoft Windows platform. It is constantly evolving, with new concepts being implemented with each update [2].

DCS (Digital Combat Simulator) World – is a simulator produced by Eagle Dynamics, the first version was released in 2008, the current version can be obtained for free from the Steam platform. It is a simulator dedicated to the military environment, with users having countless aircraft models and more. The focus is on specific operations and missions [3].

Aerofly FS 2 Flight Simulator – is another flight simulator with an emphasis on photorealistic scenarios, produced by IPACS, running on the platforms: Android, MAC, Windows and iOS. It was launched in 2014, and users can enjoy a global elevation in image, of over 300 handmade airports in West America, 3D construction, bridges, highways and detailed cities. In terms of aircraft models, the detail is impressive, the equipment and tools in the cockpit being animated, which gives a high level of interaction with the environment. The implementation of the physical elements is performed with great accuracy, and together with the fluidity of the frames with which they managed to display the image, they created an even stronger immersion effect of the user. Other navigation functions that we can find are: Route planning, ILS (Instrument Landing System), VOR (Omni Directional Radio Range) and NDB (Non-directional Radio Beacon). Moreover, this simulator provides intuitive support for Oculus Rift and HTC Vive, without any additional software [4].

X Plane 11 – it is perhaps the best rated simulator on the market at the moment. The first version appeared on November 25, 2016, and the most current on December 12, 2019. This simulator, also perceived as a video game, runs on the platforms: Windows, MAC and Linux. Users can

enjoy similar graphics and experience to reality, enjoying an impressive number of airports: 13.000 [5].

Virtual Marshalling Simulator – is an stand-alone training system produced by Virtual Simulation Systems used by the Royal Australian Air Force, the Royal Australian Navy, and Australian Army Aviation. It is specializes for ground based support crew, offering a high fidelity immersion into the world of flight deck operations with the ability to control and direct aircraft that are about to deploy or have returned from a mission and require ground marshalling. The software has dynamic weather effects with a range of settings, including wind, rain, fog, lightning, real-time scenario editor with vast control, ability to trigger aircraft emergencies such as engine fires, hydraulic leaks, hot brakes on fighter planes, and tarmac incursions by personnel or vehicles, high-fidelity virtual simulation with authentic graphic, sound and more. Simulated procedures include pilot signals, refueling, deck lashing, power, take-off and landing, and many others. The main purpose of the system is to reduce the rate of effort on already scarce resources, reliance on pilot availability and expenditure on fuel and other consumables. Otherwise reduces logistic setup for FARP (Forward Arming and Refueling Point) marshalling exercises [6].

A VR-BASED TRAINING SYSTEM COMPONENTS

The structure of a VR system, which aims at the interaction of the user subjected to the training or testing process we explain it in the fallowing way (Figure 1).

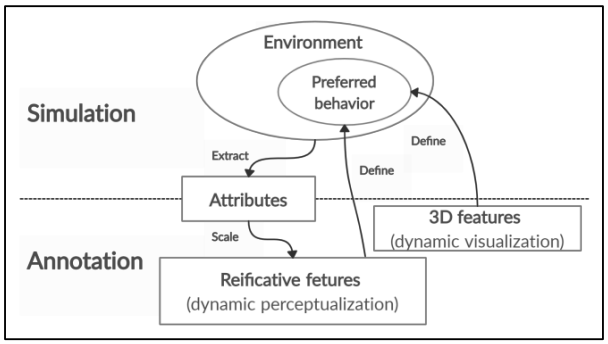

Figure 1 - Annotation of interactive simulation. Adaptation of [7].

Preferred behaviour of the simulated environment is annotated as 3D and reificative features [7]. The annotation action is one of the most natural ways for human beings to analyze and interact with documents, images and different objects [8]. Depending on the nature of the objects and events that take place in the simulated environment, the attributes of the simulation play the role of scaling factors on the reifying characteristics. In our case, a gesture or a movement of the user's hand will automatically adapt to the length of each user's arm. 3D functions are naturally calibrated to a correct scale, without modifications. In fact, 3D features can only be perceived.

A VR system consists of four main components. Dedicated software or engine for producing and managing graphics, Tracking system that constantly knows where the user or users are on stage, the Hardware part that allows Video and Audio viewing of the 3D scene and, most importantly, data content 3D.

Dedicated software or engine has several tasks to perform. First, it is the one that produces or takes over the 3D graphics. In addition, it is the one that takes the input information from the input and tracking devices and also provides the platform or environment in which the 3D scene can be developed.

The tracking system determines the position of users in the virtual world and usually uses a camera with tracking sensors that record movement. In the case of Vive or Oculus, these tracking sensors are provided as part of the HMD (Head-Mounted Display) package, but not every HMD model has this integrated part.

The visualization system represents the Hardware part, which allows the visualization of the 3D virtual scene in stereo. In the case of Vive or Oculus, this is also done via the HMD.

The 3D data or the 3D scene itself makes it possible to view and interact the user in VR. This data is the basic resource of a VR environment. If we are talking about a game, then the 3D data represents both their models and animations, as well as the background code responsible for the user's interaction with them. All the functionalities within an application, as well as the models that take part in the action are the primary resource in a VR System.

In fact, in addition to these components, the user is also part of the whole virtual environment [9]. Virtual objects are subjects in the users' direct or indirect interactions and may enhance collaboration between users. In other words, the virtual space must be constructed, first of all, considering the user's cognitive and empirical attributes. When we create virtual space models, the base criterion should be the accuracy of the human representation of reality which may not necessarily correspond with reality. To this end, the human experience is first constructed by situating the user in the virtual context, then tested through the user's direct interaction with the environment, and reconsidered, in a recursive process [10].

OUR PROPOSED VR SYSTEM FOR PERSONAL TRAINING IN AVIATION

In the following we present the context of choosing the solution, the stages we went through in the realization of the software system, as well as the technical exemplification of the important processes. We chose to make this application after a visit to Tuzla International Airport, where after some discussions with the management team, we decided that an optimal scenario, out of several possible ones, of a software product would be the ground work of the aviation personnel. The development on the part of VR in terms of flight simulation is growing. From this point of view, but also

because such a project involves a fairly large and well-trained team with specialists in the field (pilots, aeronautical engineers and so on).

We chose as main objective to focus on the activities of staff in the field and more precisely, of those activities that have not been approached by the production studios until now.

In addition, we chose to create a unique setting, the location and context of the scenario being a real one (Cliff of the Casino in Constanța, Romania).

The application meets all the conditions of a learning and testing environment for aviation personnel, so it can simulate the aiming operations of a helicopter and the adjacent tasks.

Figure 2 - Models in the foreground

The final version of the application contains the complete modeling of the chosen scenario and the general environment: background models, complete and complex models in the foreground (the flight deck of a military frigate model T22, a helicopter model Puma Naval - IAR 330 and the Casino) (Figure 2).

LOGICAL DESIGN

In view of the work plan, the next step is logical design. Here we have made decisions regarding the application architecture, technologies and concepts that we will continue to use and most importantly the use cases of the software system.

Architecture up to a certain point is an art, but from a certain point it is a science. To build something real, which will withstand the time factor requires knowledge. We preferred to give extra time to the minor details, but at the same time to keep the simplicity of the product, without detailing unnecessary parts that do not have their purpose in the application.

We chose a minimalist, simplistic, suggestive design, which does nothing but emphasize the important parts of the simulated environment, just to eliminate all the risks of the development plan, so all use cases can be achievable and the entropy factor software should not appear in future versions.

In addition to the general environment, which involves the sea plan, the geographical plan, the SkyLight and the Skybox, using Blueprints (visual scripting technology provided by Unreal Engine), we created: materials, objects, object instances, all with unique properties for each one.

Following a taxonomy process applied to the "Helicopter" concept, we developed the fully functional model / asset of the IAR-330 helicopter. Its animation is activated sequentially by triggers and gestures.

The connection of HMD to the application is done both by Blueprints and by control classes (C++). The dedicated physical aspects of the simulated context were also implemented by coding.

At the level of user interaction, friendly communication techniques were addressed, such as: assisted execution of tasks through instructions, help tools, dialogue, sound and relevant and intuitive noises. In addition, the application contains a menu that fully covers the range of use cases.

Use cases

Following an analysis of the scenario, we made a series of use cases:

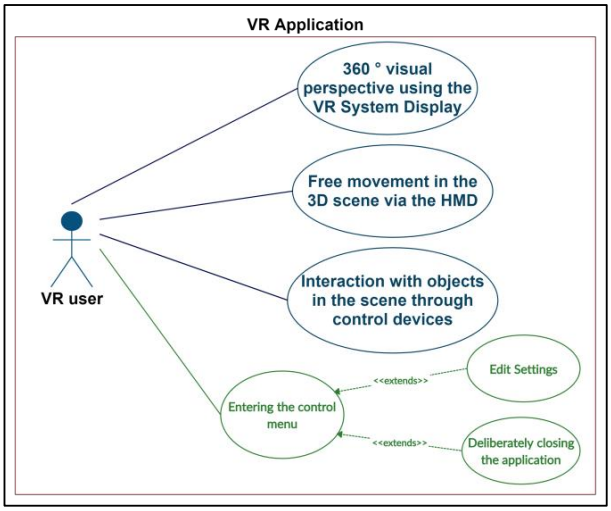

Figure 3 – Use cases Diagram

We identified the main characteristics of the actor (unique in this case), this being the user, based on a plan to simulate communications with the system (Figure 3). The use cases do nothing but model the system as desired from the end user's point of view.

They clearly describe how the user will interact with the system, then provide us with a basis for testing the status of the application. Of course, there may be multiple users of the application, but for the system they all play the same role.

First, the system must provide the user with three essential, basic things of the simulated framework, which are:
- 360 degree visual perspective, through the display provided by the VR System.
- Free movement in the 3D scene via the HMD or other tracking system.
- Interaction with objects in the scene through control devices and provoked events.

In addition to these use cases that achieve the purpose of the application itself, to provide a simulated reality environment for aviation personnel, there are others, which are part of any other system of interaction with a human actor through an interface like entering the main menu and the control menu, enterering the editing mode of the settings wich is an

exceptional behavior, being optional in addition to the previous use case and completion of the current process and exit from the application wich is also an exceptional behavior.

Conceptual classes

To identify the conceptual classes of the context of the problem, we applied a grammatical analysis on the description of the functioning of the system.

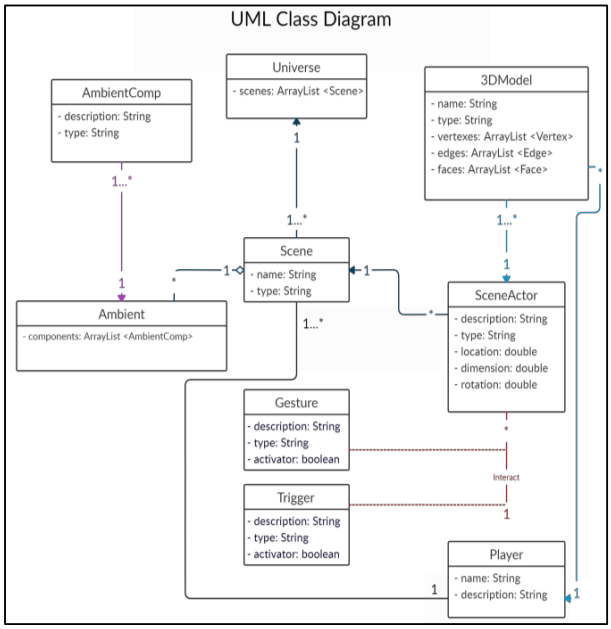

Figure 4 – Conceptual class Diagram

Thus, nouns become potential objects, the classes being identified through the following criteria [13]:

• retained information - the data retained by the object is important for the functionality of the system.

• necessary services - the object must have the ability to change the value of its attributes in a certain way through identifiable sets of operations.

• multiple attributes - objects with a single attribute can be better represented as attributes of other objects.

• common attributes - object attribute sets apply to all instances of this object.

• joint operations - sets of object operations apply to all instances of this object.

• essential requirements - external entities that consume information that is essential for the proper functioning of any system solution, will almost always be defined as objects in the requirements model.

A potential candidate should meet all these selection characteristics in order to be included in our model.

The class diagram (Figure 4) is made in a conceptual perspective, but even more so when we talk about a software product made in an engine like Unreal Engine, where, in the case of a single project, hundreds or maybe even thousands of unique objects can be created. Indeed, these objects have common attributes and can be correctly framed under the definition of the same general concept, the language of the field being captured much faster. All of our 3D assets models are instances of objects represented by the conceptual class defined Model3D, which has common conceptual attributes.

Data structure

The first step in carrying out the project was to gather as much useful information as possible. For a good understanding of the problem, first, we needed a good detailing of work tools. Both in concept (blackbox) and in detail (where it is vital).

Unreal Engine is the software engine through which we were able to develop our system. It is also developed by Epic Games, first introduced in 1998.

Unreal Engine 4.24 [11] was the engine version available when we started developing the application, and Unreal Engine 4.25 is the latest stable version released by Epic Games so far.

The visual system of Blueprints provided is a complete system of scripts, based on nodes and graphical interfaces to create game elements inside the editor. As in many other common scripting languages, it is used to define object-oriented (OO) classes or objects [12].

This system is extremely flexible, being dedicated to both designers and programmers, either for its practical utility, which provides you with the full range of concepts and tools that normally only programmers have at their disposal, or for its ability to create basic systems in C++ that can be extended by designers.

Although they are called matrices, blueprints are actually lists. Structures are used to make complex data. It is preferable to avoid creating structures within other structures, in favor of creating basic methods. In fact, the Epic Games team's own architecture is specially designed to facilitate both the effort of programmers and the work of designers in creating projects.

TECHNICAL DESIGN

In view of the work plan, the next step is the technical design. Now that we have gathered the necessary materials and laid the foundations for the construction of the application, we can move on to implementation.

Communication of the user through triggers and gestures

When we started to put ideas together at a conceptual level, we set out to make the transition as easy as possible from what classical software systems engineering involves, to the new requirements that come in approaching a software system using VR technology. Thus, in addition to the

usefulness of triggers, from classic applications, we also used gesture recognition.

A trigger activates an event or series of events. If we are talking about a movie sequence or an animation, then it is very possible that we want to activate it in a special context. Getting a reference to the sequence we made, we created a Blueprint which in turn generated a C++ code sequence that we can set so that it starts and stops when we want it to.

Regarding the sequential activation of the animation, we chose to do it by implementing triggers and gestures.

Thus, in addition to use cases and conceptual classes, we can complete the basic architecture with the sequence diagram (Figure 6).

Before the image reaches the display, triggers are activated, which in turn activate functions for gesture recognition. If the gestures correspond to gestures already saved, then an ID will be used to activate an animation sequence. Thus, only now, the provoked events can be visualized (Figure 6).

Helicopter movement animation can be controlled by the ground fly deck personnel support using hand signals such as: direction correction, landing and deck lashing.

There are several types of gestures which we also used for this application, so they fall into several categories: directional movement gestures, flow control gestures, spatial orientation gestures, multifunctional gestures (which can trigger multiple events) and tactile gestures [14].

The events caused by classic gestures activated by the buttons of the keyboard, mouse or even VR controllers are the flick, the pinch and rotation.

Based on them, their combinations, but also other imports of gestures registered by dedicated systems, an almost unlimited software implementation can be reached; the only limitations being those of physical laws in reality.

We chose to work with components dedicated to VR systems by MotionController. Using the MotionController Blueprints we added a gesture tracking component and four functions that work in parallel:

- the start function of the gesture recording
- the end function of the gesture recording
- gesture recognition start function
- the end function of gesture recognition

The first two functions are related to a gesture recording input action, which more precisely, begin to record the gesture performed by the sensors. The next two are linked to a trigger recognition action for gesture recognition which verifies if the gestures recorded by the sensors are valid or not (Figure 5). The recognized result of the gesture has an ID, which will help us in establishing the event followed by the correct recognition of each gesture.

Figure 5 - Gesture recording by activating a trigger

For the space tracking functionality of the gestures, we will also use a tracking component, which we will connect to a space drawing function and another one for predicting the drawn gesture.

If a continuous recognition of gestures in space is desired, without this taking place during the pressing of a trigger (from the controller), although in this case it is not preferable, an input action of their continuous recording will be used.

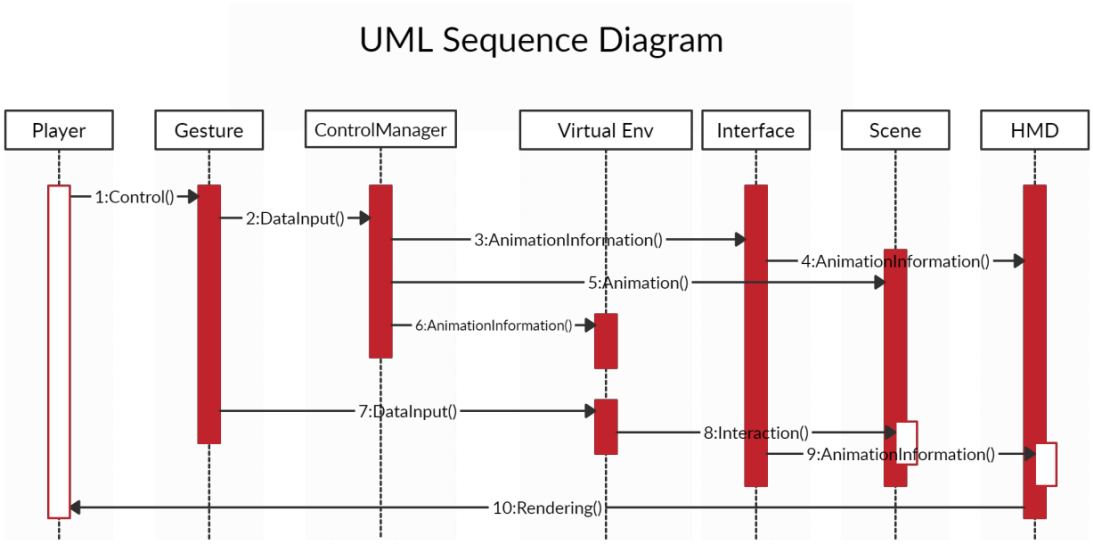

Figure 6 – Sequence diagram

Obviously, even if we had approached the continuous recognition of gestures, activating this trigger from the application at an inappropriate time, it would not have produced an event, because the animation sequence of the helicopter would not have ended. Therefore, both by recognizing by means of triggers (by pressing a button on the controller) and by continuous recognition (of the attempt to permanently recognize gestures), the helicopter will first finish its "process" started.

From the beginning of the development of the application, we started from the idea that the final product should look as simple as possible, but with a greater graphic impact. Therefore, we kept the minimalist concepts of simplicity in terms of user interaction with the system.

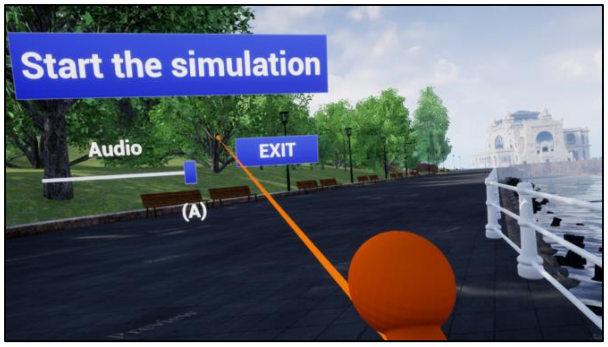

Figure 7 - Perspective from the first scene

Conceptually speaking, the universe within the application is made up of a single scene. However, technically we use the same scene, duplicated to give the user a double perspective in the created universe (Figure 7). The menu follows the user at every moment within the environment (Figure 8). It consists of a simple panel with two buttons: one that makes the transition of the scenes, in our case from scene A to scene B, but also from scene B to scene A, and the second button makes it possible to close assisted, secure application.

Figure 8 – Perspective from the second scene

The menu provides intuitive usage information, communicating to the user by its nature, but also by text indications. In addition, it is possible to edit the settings by changing the parameters. The simple and friendly aspect of the interface was made with the help of the graphical editing mode, familiar to the one from Visual Studio.

The application can be used via any HMD, we using Oculus Rift S. The hardware configuration of the HMD was done through functions implemented using Blueprint technology.

The audio element is vital in the composition of any final product of this type. The user lives the visual experience differently if it is introduced in the virtual environment and through a stereo system. In terms of sound, we created a main class and a sound mixer. All sound effects used in the application have been introduced in the parent class.

To add sound effects to the scene, we used replicas of the sounds, placed as sub-objects of the actors in the scene. It should be noted that a sound replica does not necessarily have to be related to an existing object in the scene. It can also be introduced as a singular element.

Modeling

The scene represents the space and place where the user is at a given time. This is part of the universe of a game or, in this case, the simulator, as defined in the class diagram. The scenes are populated with actors, and the actors, in turn, are made up of one or more 3D models. In a project with complex graphic content, modeling is not done directly by handwriting code. Instead, automatic code generation techniques are used, similar to the Blueprint graphical visualization, through which methods are used that contain methods, which in turn write code automatically. Developers can later access the C++ code of those classes, and can make changes or troubleshooting in case of unstable versions or plug-ins.

The Unreal modeling plug-in offers functions similar to those in dedicated software. With the help of modeling techniques such as: NURBS (Non-Uniform Rational Basis Spline) modeling, polygonal modeling and NURMS (Non-Uniform Rational Mesh Smooth) modeling, we managed to reach results similar to those obtained in professional applications.

Animation

Once the 3D models of all the actors are created, we can move on to their animation processes (where appropriate). Perhaps the most rudimentary form of animation is that of the material applied itself. In the case of a larger animation, each mesh to be manipulated in different ways must be analyzed separately. In this case, we could use the taxonomy created on the model of the IAR 330 Puma Naval helicopter, applying animations on each essential element. Thus, from understanding the main components, we were able to move more easily to understanding in detail, at a more granular level of the concept. So we decided on the type of approach we would take in the separate animation of the main rotor assembly, the tail rotor assembly, and the entire helicopter body.

At the scenario level, we have developed a series of possibilities for the trajectories of the helicopter to the point of control of its animation by the user (Figure 9).

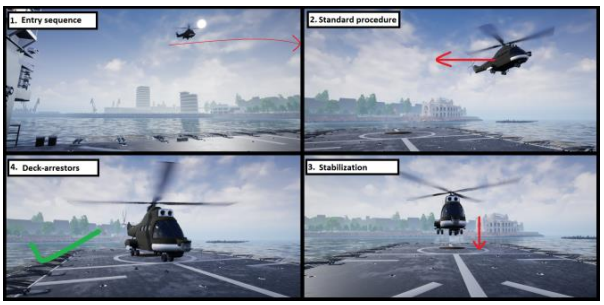

Figure 9 – Animation script

The next step in the animation process was the actual animation of the helicopter parts. Due to the nature of the 3D Model studied, there was no need to go through the process of Rigging and Skinning, but we needed again the taxonomy created in the evaluation of the rotor parts (Figure 10).

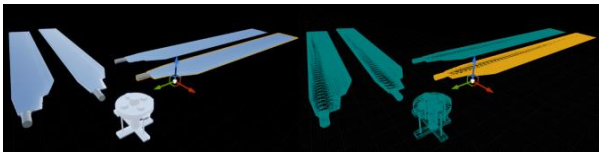

Figure 10 - Parts of the helicopter to be animated based on the taxonomy

The rotation function of an object is implemented via the RotatingMovementComponent class in the standard GameFramework Unreal Engine package. This function can be added to the actors of a scene and can be used for different purposes. In principle, if the pivot of the animated object is not changed, then the composition of the new location is done by default through it.

Figure 11 - Animation of the main rotor and the tail rotor together with their propellers

Thus, using the rotational component of the movement on the main rotor block and the secondary rotor block (tail) we were able to animate the essential components in question of the helicopter (Figure 11).

Both in the elaboration of the animation script and in the placement of the actors on stage we focused on: the correct lighting of the scene and its interaction with the materials of the 3D models, the positioning of the camera and the user's perspective to simulate the immersion in the virtual environment and balancing the speed at which the animation happens in the development phase with the number of frames processed per second of the VR system. Given that such a display does not have the ability to process a very high number of frames per second, we had to make a compromise in terms of the actual rotational speed of the helicopter blades. Thus, on the display of the VR system you can perceive as naturally as possible their rotational movement.

At the level of composition, we recreated the panorama of the real cliff scenario, using textured materials in the same style, thus completely homogenizing the chosen context. In addition, we used techniques specific to creating niche games and graphics applications. Thus, in order to keep a better optimization of the data flow processed by the system and to obtain as many FPS (frames per second) as possible, we opted in the detailed modeling of the foreground elements and the important elements in image register, leaving the background populated with texturally poor objects, but especially polygonal, for a faster rendering of the image.

In terms of animation, at least half of the effect created in an application is sound. Although the process of creating images itself is more expensive, the emphasis should be equally on the audio side. Most of the time, as in the case of film production, a good image combined with poor sound quality is much worse than a bad image, but it comes with a high sound quality. The audio part can save the final product from failure or throw it even harder into the abyss.

For the sound caused by the engine, the rotor and the blades of the helicopter we used an audio material in wav format.

The last step we took in finalizing the animation was the final editing and the general aspect of the image-sound fluidity. Once we changed certain parameters that were not exactly in place, through a subjective analysis of duality we arrived at the desired final version of the animation of the main actors.

CONCLUSION

Following the stages of content creation in a logical order of the modeling and animation processes, but also of the texturing and material creation processes, we managed to create the basic component of a VR system. By understanding the related concepts and technologies we were able to connect the created component to the tracking and viewing components provided by the HMD Oculus Rift S device together with the Oculus touch devices. Thus, adding to the equation the Unreal Engine and the processing hardware, we created the complete framework of the VR System infrastructure.

In designing the use cases, we approached a realistic attitude, calculating all the variants and risks of the development process. Thus, we did not intend to implement use cases, which require a higher level of experience, precisely in order to finally reach a final product, with its positive parts, even if it still has limitations. Their technical design and implementation were the main learning ingredient. During the implementation there were times when we had to make decisions, with varying degrees of importance. After making the wrong decisions, we were able to go back and learn from them. Moreover, through testing sessions, we redesigned some of the basic architecture so that we can achieve better

results. Our user perspective has helped us fix most of the issues that have arisen during implementation.

The software system fully responds to the desired and proposed appearance issues. However, both its nature and the technologies used lead to a list of limitations that can be reduced by later versions of the application or other interpretations as the technology evolves. First, the nature of the simulated virtual environment will never be compared to reality. The user's immersion in the virtual environment will never be equivalent to that in reality, without direct interventions from the outside (for example: the underwater environment). Second, we are often limited by the space we are in.

As same as Virtual Marshalling Simulator, the application offers the advantages of learning processes, avoiding potential risks (damage 0 after the learning or training process), can preview a candidate's behavior following events, but can also serve for entertainment purposes.

The expansion of the software system in later versions may include more complex general visual areas, such as a systematic database used in the customization of tools and meshes, but also the choice of more complex scenario and more diversified simulation operations.

ACKNOWLEDGMENTS

Thanks go to the CeRVA team from the "Ovidius" University of Constanta. We also thank the management team of Tuzla Aerodrome in Constanta with special mentions to Regional Air Services.

REFERENCES

1. Leslie Mertz, *Virtual Reality Pioneer Tom Furness on the Past, Present, and Future of VR in Health Care*, Publisher: IEEE Pulse 10(3):9-11, 05 2019, ISBN: 2154-2287, DOI: 10.1109/MPULS.2019.2911808.
2. Cristiano Rodrigues, Rosaldo J.F. Rossetti, Daniel Castro Silva, Eugénio Oliveira, *Distributed flight simulation environment using flight simulator X*, Published in CISTI journal, 07 2015, DOI: 10.1109/CISTI.2015.7170615.
3. Digital combat simulator homepage, https://www.digitalcombatsimulator.com/en/.
4. Roganov V.R., Roganova E.V., Micheev M.J., Kuvshinova O.A., Zhashkova T.V., Gushchin S.M., *Flight simulator information support*, Published in journal: Defence S and T Technical Bulletin, 01 2018, ISBN: 1985-6571.
5. Fernando Soares Carnevale Ito, Roberto Santos Inoue, Luiz Carlos Querino Filho, Kalinka Castelo Branco, *Cooperative UAV formation control simulated in X-plane*, conference: ICUAS, 06 2017, DOI: 10.1109/ICUAS.2017.7991421.
6. Virtual Simulation Systems: Virtual Marshalling Simulator homepage, https://virtualsimulationsystems.com/site/index.php/simulation/fixed-wing/aircraft-marshalling.
7. Mikko J. Rissanen, Yoshihiro Kuroda, Tomohiro Kuroda, Hiroyuki Yoshihara, *A Novel Interface for Simulator Training: Describing and Presenting Manipulation Skill Through VR Annotations*, conference: HCI 07 2007, ISSN: 0302-9743, DOI: 10.1007/978-3-540-73335-5_57.
8. Stefanut T., Gorgan D., *Graphical annotation based interactive techniques in eTrace eLearning environment. "eLearning and Software for Education"*, eLSE 2008, The 4th International Scientific Conference, Ed. Universitara, ISBN: 978-973-749-362-0.
9. Popovici D.M., *A foray into 3D virtual environments (in romanian – O incursiune în mediile virtuale 3D)*, Ed. Muntenia, 2007, ISBN: 978-973-692-191-9.
10. Popovici D.M., Hamza-Lup F.G., Polceanu M., Zagan R., Gerval J.P., Querrec R., *3D Virtual Spaces Supporting Engineering Learning Activities*, Published in IJCCC, 11 2009, DOI: 10.15837/ijccc.2009.4.2456.
11. Paul Oliver, *Unreal Engine 4 Elemental*, 08 2012, DOI: 10.1145/23411836.2341909.
12. David Nixon, *Beginning Unreal Game Development. Foundation for Simple to Complex Games Using Unreal Engine 4*, Publisher: Apress, 02 2020, ISBN: 978-1-4842-5639-8 DOI: 10.1007/978-1-4842-5639-8_5.
13. Bogdan C., *Concern-oriented and ontology-based Modular Architectural Design of Software Systems*, 1st International Conference on Economics and Information Technology e-Society Knowleadge and Innovation, Bucharest, Romania, 2008, ISBN: 978-973-749-491-7.
14. Kasper Rise, Ole Andreas Alsos, *Human-Computer Interaction. Multimodal and Natural Interaction: Gesture-Based Interaction: Visual Gesture Mapping*, conference: HCI 07 2020, ISBN: 978-3-030-49061-4, DOI:10.1007/978-3-030-49062.