Visual interfaces effect on player performance in virtual reality

Software development report

Timothy O’Neill

06/03/2015

Contents

Introduction

To explore fully on the impact of different visual interfaces on a players performance in virtual reality a software artefact must be created, it must be able to provide the necessary results to help us reach a conclusion on the research topic proposed.

The artefact to be produced must allow a user to perform a predefined task using both a virtual reality headset and without the headset, the experiment should take the form of a game to keep the user engaged and interested in the task to be performed, the game must be exactly the same on both mediums thus the only varying factor is whether they are using the headset or not, each of the following interface groups should be implemented and be identical in both mediums, spatial, non-diegetic, diegetic and meta a fifth interface should take the form of making the user play the game with no interface at all, the system should be recording various metrics in the background for analyse for the research to be able to draw a strong conclusion.

The details of how this artefact will be designed, implemented and tested is outlined in the following document.

System Specification

The system to be developed should take the form of a simple game revolving around the user moving between locations guided by waypoints while dodging incoming projectiles, the user will be presented with five different interfaces to aid them in their task and metrics about their performance should be recorded for the researchers benefit.

The System will be used two stakeholder groups myself as the researcher and the subject of the experiment the user as such I have drawn up a set of functional requirements presented as use cases for both groups as outlined below in Fg 1,2.

|  |  |
| --- | --- |
| **User requirements** | |
| Player movement | As a player I want to be able to move around the game world in order to avoid the projectiles and complete my objective. |
| Player camera | As a player I want to be able to view the game world using the Virtual reality headset or a traditional monitor. |
| Player asset | As a player I want a visual representation of my character to view while playing the game. |
| Projectile movement | As a player I want the projectiles to move to various locations to make me have to make an effort to avoid them as they approach. |
| Projectile Collision | As a player I want the projectiles to collide with me and the game world so to create a game of avoiding being hit by the asteroids. |
| Projectile spawner | As a player I want multiple asteroids to come from various directions to create a more challenging experience. |
| Projectile asset | As a player I want the projectiles to have an asset to create an immersive and real experience and to help me easily identify them. |
| Diegetic Interface | As a player I want to be provided with a Diegetic interface so I can use it to help me perform my task. |
| Non Diegetic Interface | As a player I want to be provided with a Non Diegetic interface so I can use it to help me perform my task. |
| Meta Interface | As a player I want to be provided with a meta interface so I can use it to help me perform my task. |
| Spatial Interface | As a player I want to be provided with a spatial interface so I can use it to help me perform my task. |
| Interface assets | As a player I want the interfaces to be presented well so they convey the information clearly and in an understandable form. |

Figure 1. User use cases

|  |  |
| --- | --- |
| **Developer Requirements** | |
| View Switcher | As developer I want to have a means to switch between the normal view and the Virtual reality view at runtime. |
| Menu System | As a developer I want a menu system to be able to switch between the various game modes, tutorial and experiment configurations at runtime. |
| Game Timer | As a developer I want to be able to control the length of time a player has to complete a task and also how much time has passed since the scenario began. |
| Game Mode | As a developer I want to be able to easily define the rules of the player’s task so there are easily followed and extendible in the system. |
| Demo Mode | As a developer I want the player to be allowed a practice run of the game mode to enable them to familiarise themselves with the game itself. |
| Metric Tracking | As a developer I want to be able to track metrics in order to make judgements about user performance. |
| Interface Manager | As a developer I want an easy way to switch between the various interfaces to make the program easy to manage and maintain. |
| Tutorial | As a developer I want the player to easily understand the task they need to fulfil and the entities of the game therefore a tutorial is required to achieve this. |
| Projectile types | As a developer I want the projectiles to vary in order to create a more interesting experience for the player and enable more diverse tracking of metrics. |
| Game Arena | As a developer I want the game to take place in a controlled arena that allows me to restrict the movement of the player a certain amount to make sure the experiment is consistent and challenging. |
| No Interface | As a developer I want to provide the player no interface to perform the task in order to draw a richer conclusion on the subject matter. |

Figure 2. Developer use cases

Hardware Requirements

The virtual reality headset chosen to be used for conducting the experiment has been the Oculus rift due to its availability and how well it is supported by multiple development environments, specifically we will be using an Oculus Rift Development kit two the specification and hardware requirements of which can be seen in Fg 3.

|  |  |
| --- | --- |
| **Oculus Rift Dk2 Specification[1]** | |
| Resolution | 960x1080 per eye |
| Refresh rate | 75Hz, 72Hz, 60Hz |
| **Minimum System Requirements[2]** | |
| Operating system | Windows 7,8 or 8.1, 10 |
| Processor | 2.5+ Ghz Processor |
| Ram | 4GB |
| GPU | AMD Radeon HD 6950 or above  Or  NVIDIA GeForce GTX 560 or above |

Figure 3.

As we can see the DK2 runs at a maximum of 75Hz therefore we need to produce a performant piece of software that will maintain a constant Frame Rate of 75 frames per second to be able to keep a consistent experience for the test subject.

Design

\*COME BACK TO NEEDS TO BE MORE ABSTRACT\*

At this point it was chosen that Unreal Engine 4 would be used for the implementation of the system the rationale behind that is further described in the implementation section, below are the UML Diagrams for the desired system to be produced, unreal engine has excellent interfaces for playing games they come with a rich set of functionality and well defined where possible we will extend of these to add our own functionality, the UML diagrams for the system are shown below in Fg 4, 5,6,7,8.

Interface Design [4]

For the experiment the player will use the five interfaces outlined above excluding “No Interface” each of these must be designed in such a way that they make sense both on an Oculus and a standard monitor, the following section highlights the design choices for these interfaces.

Interfaces general design

The interfaces should follow a consistent theme of the projectiles being identified by red colours and the Players Objective being displayed by Green, any interface that isn’t based in the game world should also show a dot to indicate the player shown in Yellow, the interfaces should never be intrusive as to reduce the viewable game world of the player, in addition the interfaces should always be relative to the players position updating with their movement and rotation to allow the player to be aware of their relative surroundings.

Diegetic interface

Diegetic interfaces are part of the players narrative and appear in the game world, I have chosen to emulate a diegetic interface recently shown in a recent game Alien Isolation [3], In the game the player is given a handheld scanner with which they can track the position of the enemy pursuing them, an image of the interface is shown in Fg 4.

Figure 4. Alien Isolation Hand scanner

Non diegetic interface

Non diegetic interfaces are your classic radar that appears in the top left of your screen, these are well defined and commonly appear in first person games, an example of which is shown in Fg 5. While I will not use this as direct inspiration most of these interfaces follow a similar design pattern.

Figure 5. Runescape Minimap

Spatial interface

The spatial interfaces should show the target of the projectiles, they will be placed on the ground of the game world and displayed for the lifetime of the projectile, the interface is widely used in MMORPG’s during boss fights to signify where a boss is going to cast a fireball for example, most recently Wild star a MMO used spatial interfaces for all forms of attacks from players and enemies an example of which can be seen in Fg 6.

Figure 6. WildStar Telegraphs

Meta interface

Figure 7. Goldeneye 2010 remake

Meta interfaces provide temporary contextual information to a player, the design of this interface will follow those presented in first person shooter games to highlight where a player is being attacked from, Games such as call of duty have shown blood spatters on the players interface to indicate an enemies point of conduct an example of which is shown in Fg 7.

Player Movement

Due to the infancy of virtual reality many core gaming systems are still in the teasing stage none more so that player movement, many games currently ported to VR have all very different control schemes for their players, for the design of player movement I will present previous existing games that have different control schemes all using an Oculus rift and a Xbox360 controller.

Rail Movement [5]

Many old arcade shooters used a system where the player was moved between locations and presented with enemies to shoot, this has made a resurgence in VR, Bullet train a VR exclusive built in unreal engine uses this system, the player can move to fixed teleport locations and engage with enemies in their current view, this system while having some initial learning of where to teleport provides a clean immersive experience allowing for the user to perform the current task without getting taken out of the experience.

Separated Head Movement

This is the mostly commonly used control system for ported games, TF2 has many control systems available for VR however using its standard controls of player movement and then allowing the Oculus user to look in any direction leads to a certain familiarity for hard-core players however the mixture of the two for non-gamers can be a steep learning curve meaning their performance may be negatively affected meaning their results may not be representative of their true performance.

Connected Head Movement

This is the simplest form of player movement allowing for the player to move around in the direction they are looking, the oculus controls the rotation of the player in the game world and such any forward movement will be in that given direction, this may be negatively received by hard-core participants but ultimately it is the most intuitive for non-gamers and hard-core alike, meaning their results should show a true reflection of their actually performance.

Metrics Tracking

The metric tracking class will be implemented using a singleton pattern, this has several advantages to it, the metrics tracking will only have ever one instance since we are writing files to disk this will help make sure there’s no clashes on file writes or accesses, it also makes it globally available to other classes allowing them to write metrics specific to them with minimal coding required.

Due to the nature of the experiment each interface that the user is performing with will have metrics tracked against it, the number of metrics being tracked will be relatively small therefore it is reasonable to store them in memory until that particular interface is finished and then write the results to disk, this will save constant disk read writes and be a small performance optimization for our piece of software.

Implementation and testing

The following section outlines the final implementation of the artefact and the testing performed during the development of the project.

Engine Choice

For undertaking any games development project choosing a game engine foundation is fundamental to producing a solid artefact, while there is many options available I made the decision ultimately between three.

Unity 5

Unity comes with Oculus integration built in it also allows for fast game development due to it handling a lot of the low level systems leaving the high level details of the game two the user, unity gameplay functionality is primarily written in C# while I have previous experience in the language it would not be my preference. Unity is very well documented and therefore problems faced during development could easily be addressed, also the initial learning curve of picking up the engine would be simple.

Bespoke

Producing my own framework to work in is an option, using the Oculus Sdk I could easily interface with its functionality, it would also give me the freedom to choose the programming language I would wish to use, however producing the gameplay on top of a game engines core functionality would require an excessive amount of time and add unnecessary complexity, to an already large project.

Unreal Engine 4

Unreal engine 4 similar to unity comes with Oculus Integration built in, it too offers tools in order to handle rapid development of games, the primary language with which to produce gameplay is a mixture of two things, C++ programming and a Unreal visual scripting language called blueprints, C++ is my preferred language, blueprints are very useful to accomplish simple tasks such as menu navigation, they also could be used in the implementation of some of the game interfaces due to the focus being more on the visual presentation of the interface, one downside of unreal engine is that documentation is not widely available and can be outdated also the C++ programming has very few tutorials due to the infancy of this most recent distribution of the platform, this would make the initial learning and development in Unreal potentially tricky.

With all the above considered the artefact was chosen to be produced in Unreal Engine 4 using a mixture of blueprints and C++, due to the language it uses and the integration it already has with the Oculus along with the suite of tools it provides to produce games.

**Oculus Rift and Unreal Engine**

The Unreal engine out of the box has a plugin to easily enable usage of game engine with the Oculus Rift, the unreal engine team has implemented the various technical pieces of the Oculus SDK into their game engine and thus makes the usage in our project much simpler, any created game in unreal engine can be launched into VR mode allowing us to easily test and iterate on the project.

Key Functions are accessed using C++ through the HMDDevice Reference a list of the available functions can be found here [6].

Developed Classes

The following UML Diagrams outline the main C++ and Blueprint classes and their composition.



Figure 8. Player classes.



Figure 9. Interface classes. 

Figure 10. Game mode classes



Figure 11. Projectile classes



Figure 12. General classes.



Figure 13. Blueprint Classes.

Unreal Engine Classes

The unreal engine provides a number of interfaces to gain certain functionality in your classes from update loops to HUD interfaces I’ll outline base classes in this section.

|  |  |
| --- | --- |
| Unreal Engine Class | Description |
| AActor | Actor is the base class for an Object that can be placed or spawned in a level. |
| AGameMode | The GameMode defines the game being played. It governs the game rules, scoring, what actors are allowed to exist in this game type, and who may enter the game. |
| APawn | Pawn is the base class of all actors that can be possessed by players or AI. They are the physical representations of players and creatures in a level. |
| UPawnMovementComponent | Movement component meant for use with Pawns. PawnMovementComponent can be used to update movement for an associated Pawn. It also provides ways to accumulate and read directional input in a generic way. |
| FTableRowBase | Base class for all table row structs to inherit from, row structs define the setup of a data table for storing game variables such as health per level of an enemy or XP tables. |
| UObject | The base class of all objects such as Actor or Pawn. |
| FTickableGameObject | This class provides common registration for game thread tickable objects. It is an abstract base class requiring the Tick() method to be implemented. |
| PlayerController | The Player Controller is the interface between the Pawn and the human player controlling it. ThePlayerController essentially represents the human player’s will. |
| HUD | Base class of the heads-up display. This has a canvas and a debug canvas on which primitives can be drawn. It also contains a list of simple hit boxes that can be used for simple item click detection, can also be accessed from blueprints. |
| UserWidget | Are pre-made functions that can be used to construct your interface (things like buttons, checkboxes, sliders, progress bars etc.) Widgets are edited in a specialized widget Blueprint. |

Figure 14. Unreal Engine Classes

Blueprints

Blueprints were used in areas where visuals where primarily required compared to coding, another use of the blueprints is for example shown with the Asteroids, Because they are being spawned at runtime it made it difficult to set their model using the normal Ue4 tools therefore by defining it in a blueprint the C++ code can pick this up at runtime making for a much tidy, reusable code base, other classes also use this technique for their various meshes and visual assets.

Blueprints for Widgets

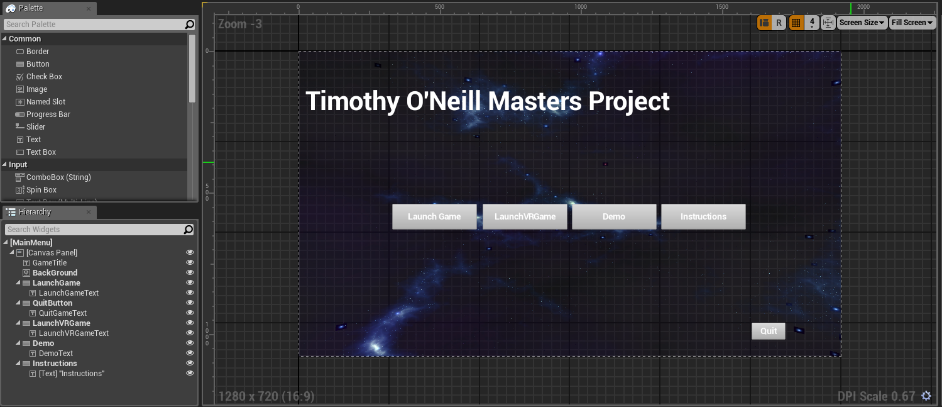
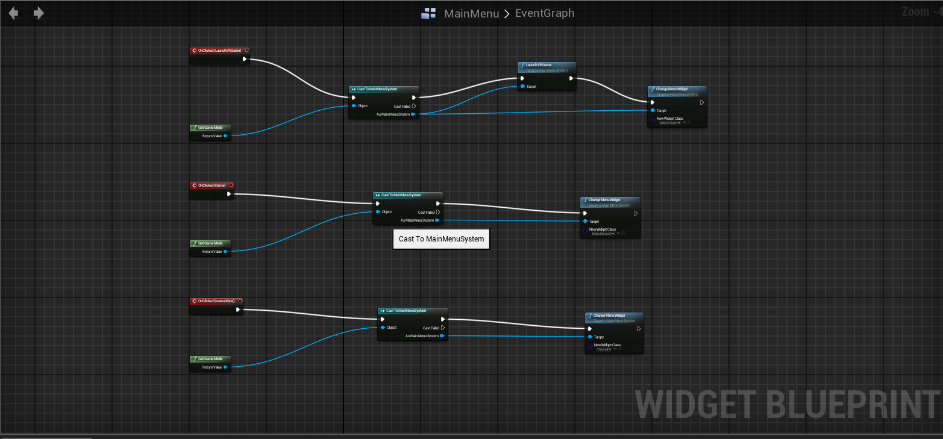
As mentioned before one of the main uses of blueprints was for visual elements the various menu systems of the program where visually created using the unreal tools and the light amount of scripting to move between menus for example, was done in the background an example of this is shown in Fg 15,16. Which shows the main menu blueprint and its corresponding blueprint code, As you can see the various buttons and components can of the interface can be easily laid out and designed to suit the interface and their event functionality defined in the visual scripting of blueprints removing any complexity of using C++, The performance over head of using blueprints [7] is negligible for small processes such as a button event and as such is perfectly suited for this kind of functionality.

Figure 16. Main menu code.

Figure 15. Main menu visual.

Blueprints for HUD’s

Two out of the four HUD’s are made purely by Blueprints, the rationale behind this choice is similar to menu widgets, also unreal allows for the ability to easily switch between HUD’s at runtime by calling   
 APlayerController::ClientSetHUD(TSubclassOf<class AHUD> NewHUDClass);

Both the Non Diegetic and Meta interfaces are created using blueprints.

Diegetic Interface

The diegetic interface is probably the most complex of all the interface implementations, it comprises of several components.

* C++ class controlling when it renders and how it attaches to the player Model.
* Widget class that controls how the interface appears visually.
* Actor Class for the Hand held model.
* Actor Class to place the widget in making it a 3D widget it so it can be placed anywhere in the world.

Fg 17. Shows the Various components of the interface.



Figure 17. Diegetic Interfaces Components

(Discuss test plan and outline key best practices)  
(“Is there a realistic assessment of the state of the software”: it works)

Conclusions

(“Has a review of achievements been provided?” – I’m presuming this outlining what I have learned and what I have “Achieved” with the project)

(Future Work)

References

[1] <http://riftinfo.com/oculus-rift-specs-dk1-vs-dk2-comparison>

[2] <https://developer.oculus.com/documentation/pcsdk/0.4/concepts/dg-sdk-setup-requirements/>

[3] <http://www.alienisolation.com/age-gate>

[4] <http://www.thewanderlust.net/2010/03/29/user-interface-design-in-video-games/>

[5] <https://www.youtube.com/watch?v=vVQ49XrdpJo>

[6] <https://docs.unrealengine.com/latest/INT/API/Runtime/HeadMountedDisplay/IHeadMountedDisplay/index.html>

[7] <https://wiki.unrealengine.com/Blueprint_FAQ_and_Tips>