Employing Binocular Eye Tracking for the Control of a Custom-made Immersive Multimedia User Interface (MUI) in Unity3D.

Nikolaos Sidorakis

Thesis submitted for the degree of
Bachelor of Science in Electronic and Computer Engineering

Department of Electronic and Computer Engineering
Laboratory of Distributed Multimedia Information Systems and Applications - MUSIC
Technical University of Crete, Greece
CHANIA 2015

Acknowledgments

Θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου στην επιβλέπουσα καθηγήτρια μου κ. Μανιά Αικατερίνη που με εμπιστεύτηκε και μου έδωσε την ευκαιρία να ασχοληθώ με το συγκεκριμένο θέμα και υλικό.

Ένα τεράστιο ευχαριστώ στον Γιώργο Κουλίερη για την αμέριστη υπομονή και βοήθεια από μεριάς. Αναπολώ τις στιγμές που περάσαμε στο γραφείο από νωρίς το πρωί μέχρι και τα ξημερώματα για να προλάβουμε deadlines. Ο καλύτερος συνάδελφος, συνεργάτης και φίλος.

Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές κ. Χριστοδουλάκη Σταύρο και κ. Μπάλα Κωνσταντίνο, για το χρόνο που θα αφιερώσουν στην ανάγνωση του κειμένου. Θα ήθελα να ευχαριστήσω επίσης όλους όσους συμμετείχαν στην διαδικασία των πειραμάτων τόσο των τελικών αλλά και κυρίως των αρχικών που βοήθησαν με παρατηρήσεις και αξιολογήσεις για την βελτίωση της διεπαφής.

Ένα μεγάλο ευχαριστώ στην οικογένεια μου για την αμέριστη και απόλυτη υποστήριξη που μου παρείχε και συνεχίζει να μου παρέχει καθώς και στους ανθρώπους εκείνους που νιώθω κοντά μου, για την κατανόηση, αγάπη, υπομονή και έμπνευση που μου δίνουν τόσο στις καλές όσο και στις δύσκολες στιγμές. Ευχαριστώ τους Θανάση, Χρόνη, Μητσάρα, Μανωλάκη, Βασιλάκη, Βουτσάκο, Στελάκη, Πιαλόγλου, Στέλιο, Γιάννη, Φαίη και Τιφανούλα, και όλους όσους ήταν δίπλα μου!

Abstract

In this thesis, we present an innovative approach to design a gaze controlled Multimedia User Interface for modern, immersive headsets. The wide-spread availability of consumer grade Virtual Reality Head Mounted Displays such as the Oculus Rift™ transformed VR to a commodity available for everyday use. However, Virtual Environments require new paradigms of User Interfaces, since standard 2D interfaces are designed to be viewed from a static vantage point only, e.g. the computer screen. Additionally, traditional input methods such as the keyboard and mouse are hard to manipulate when the user wears a Head Mounted Display. We present a 3D Multimedia User Interface based on eye-tracking and develop six applications which cover commonly operated actions of everyday computing such as mail composing and multimedia viewing. We perform a user study to evaluate our system by acquiring both quantitative and qualitative data. The study indicated that users make less type errors while operating the eye-controlled interface compared to using the standard keyboard during immersive viewing. Subjects stated that they enjoyed the eye-tracking 3D interface more than the keyboard/mouse combination.

Publications

- Sidorakis, N., Koulieris, G., Mania, K. (2015). Binocular Eye-Tracking for the Control of a 3D Immersive Multimedia User Interface. In Everyday Virtual Reality (WEVR), 2015 IEEE 1st Workshop

- Koulieris, G.A., Drettakis, G., Cunningham, D., Sidorakis, N., Mania, K. (2014). Context-aware Material Selective Rendering for Mobile Graphics. Poster, ACM Siggraph 2014, Vancouver, Canada [selected among 25 best posters of ACM Siggraph 2014]

Table of Contents

Acknowledgments 2
Abstract 3
Publications 4
Table of Contents 5
List of Figures 7
List of Tables 9
Chapter 1 – Introduction 10
1.1 Purpose 10
1.2 Thesis Structure 11
Chapter 2 – Background 12
2.1 Human Computer Interaction 12
2.2 Virtual Reality 12
2.3 Virtual Reality Systems 14
2.3.1. Immersion 14
2.3.2. Stereoscopy 15
2.3.3. Virtual Reality Gear - Head Mounted Displays 15
2.4. The Human Eye 16
2.4.1. Anatomy of the Eye 17
2.4.2. Vision: The Eye's Functionality 18
2.4.3. Vision Types 20
2.4.4. Eye Movement and Control 21
2.4.5. Perception in an Immersive Virtual Environment 23
2.5. Eye Tracking 23
2.5.1 History & Methods 24
2.5.2. Features that can be tracked 25
Chapter 3 – Software Architecture and Development 26
3.1 Game Engine: Unity 3D 26
3.1.1 The Third Dimension (3D World) 26
3.1.2 Unity's Essential Concepts 30
3.2 Overview of Eye Tracking Device's Software 31
3.2.1. Interface 31
3.2.2 How the software works 34
3.2.2. Binocular Method 35

3.2.3. Calibration 35
3.2.4. SDK and Communication with Unity3D 36
3.3. Head Tracking Device - Software 37
3.3.1. Overview 37
3.3.2 SDK and Communication 38
Chapter 4 – UI Implementation 39
4.1 Creating the 3D Visual Content 40
4.2 Setting up the Virtual Scene 47
Chapter 5 – Implementation (Scripting) 55
5.1 Stereoscopic View - Setup 55
5.2 Handling User Input 57
5.3 Communication with the Eye - Tracking Device (.dll) 58
5.4 Communication with the Head - Tracking Device (.dll) 62
5.6 Deprojection of eye's gaze data 63
5.7 Logging of Events 65
Chapter 6 – User Study 66
6.1 Material and Methods 66
6.1.1 Apparatus 66
6.1.2 Visual Content and Experimental Procedure 66
6.1.3 Simulator Sickness 68
6.2 Results and Discussion 68
Chapter 7 – Conclusions 70
Chapter 8 – Reference & Bibliography 71
Appendix A: Publication 73
Appendix B: User Study Questionnaire 77

List of Figures

Figure 1: Human Computer Interaction was not considered 12
Figure 2: Real Image Display vs. Virtual Image Display 13
Figure 3: The NVisorTM SX111 HMD used in this thesis 16
Figure 4: The Basic Anatomy of the Human Eye 17
Figure 5: How Each Part of the Eye Interacts and Adjusts to the Incoming Light/Image Display 19
Figure 6: The Human's Eye Muscle Anatomy 19
Figure 7: (a) Horizontal Field of View (b) Vertical Field of View 20
Figure 8: Binocular vs. Monocular Vision 21
Figure 9: Perception of a plane object in monocular vision 21
Figure 10: A Spacecraft Model. Left: A Simple Mesh Collider with a Small Polygon Count. Middle: No Mesh Component Right: A Higher Polygon Count Mesh 27
Figure 11: Unity's 3D Basic Components of a 3D Model 28
Figure 12: Unity 3D Primitive Shapes and Colliders(Cube, Sphere, Capsule, Wheel) 29
Figure 13:Startup Interface of the ViewPoint EyeTacker Software 31
Figure 14: EyeCamera Window 32
Figure 15: EyeSpace Window 32
Figure 16: The Controls Window 32
Figure 17: Status Window 33
Figure 18: Stimulus Window 33
Figure 19: Pen Plot Window 33
Figure 20: Schematic of the ViewPoint EyeTracker System (Head Fixed) 34
Figure 21: Calibration Attempts. The Left Picture Depicts a Successful Calibration. The Right Picture Depicts a Wrong Calibration 36
Figure 22: The Inertia Cube3 and its Processor 37
Figure 23: Left – ISDEMO application showing real-time Yaw, Pitch and Roll data. 38
Figure 24 : The Application's Main Interface from a Higher Angle to Display the Individual Applications 40
Figure 25: The Folder, Picture and Preview Box Models of the Explorer Scene 41
Figure 26: The Keyboard Models. All Three Types are shown. Char Layout, and both Number and Symbol Layouts 43
Figure 27: The Song Model, Music Player Model and the Speakers 44
Figure 29: A New Terrain Gameobject and Unity's Terrain Toolkit 45
Figure 30 : Using Unity's Terrain Toolkit. The First Attempt of Creating the Environment 45
Figure 31: Applying Ground Textures to the Terrain 46
Figure 32: House and Tree Models Imported into the Scene 46
Figure 33: A Random View of The Final Action Game Scene After Having Imported All the Models 47
Figure 34 : The Green Box is the Folder's Collider Component 48
Figure 35: A View of the Pictures Explorer Scene 48
Figure 36: A View of the Music Player Scene 49
Figure 37: A View of the Email Composer Scene 50

Figure 38: A View of the Text Editor Scene when the User Created a New Document File..... 51
Figure 39: (a) A View of the Scene when the User Selects to Load Document File from the Windows Directory (b) The User Selects to Edit the File myDoc.txt. It's Content and Path are Displayed and the User is Allowed to Edit, Save or Discard the Changes he makes 52
Figure 40: A View of the Tiles Game Scene Before the User has Chosen a Image to Start Playing With. 53
Figure 41: A View of the Tile Game Scene While the User has Selected to play the Puzzle with the Polar Bear Image and Has Placed a Few Blocks at the Correct position..... 54
Figure 42: Stereoscopic Cameras' Field of View. The Left and Right Cameras are Rotated by 13^o with a Field of View 90^o each. The Field of View of the Left and Right Cameras are in a Yellowish and Orange Shade Respectively. The Stereo View of the Scene is in a Pink Shade. 56
Figure 43: ViewPoint Software Detecting a Blink Once the Eye Lid Closes (Left) but unfortunately it Almost Immediately identifies the eye lashes as a pupil. (Right)..... 58
Figure 44: The Status Window of the Viewpoint Software Shows the Total Number of Registrations from Third Party Applications..... 60

List of Tables

Table 1: Quality Codes and the Respectively Information about the New Data Received 60
Table 2: (Left) Comparison of Task Completion Times for the Eye Tracker Input Versus Blind Keyboard Input for all Subjects. (Right) Comparison of Type Errors for the Eye Tracker Input Versus Blind Keyboard Input for all Subjects 68
Table 3: Responses on the Four Questions of the Qualitative Questionnaire for the 3D UI Keyboard..... 69

Chapter 1 – Introduction

1.1 Purpose

Virtual Reality (VR) is soon to become ubiquitous. The widespread availability of consumer grade VR Head Mounted Displays (HMDs) such as the Oculus Rift™ [8] transformed VR to a commodity available for everyday use. VR applications are now abundantly designed for recreation, work and communication. VR has a wide range of applications which range from gaming and entertainment through to medicine, engineering, military training, scientific visualization and business. But as with any new technology there are issues degrading the usability of this system. Interacting with VR setups requires new paradigms of User Interfaces (UIs), since traditional 2D UIs are designed to be viewed from a static vantage point only, e.g. the computer screen [2]. However, user interaction in a 3D spatial context introduces constraints due to the multiple degrees of motion freedom, requiring novel interaction metaphors such as “fly” and “zoom”. These metaphors are not applicable in a standard 2D interface [1, 2]. Adding to this, traditional input methods such as a keyboard and mouse are hard to manipulate when the user wears a HMD. Using a keyboard and a mouse while immersed in a VR HMD is an erroneous extension of the desktop paradigm to VR, constituting a fundamental challenge that needs to be addressed [2].

There have been attempts of employing eye-tracking methods as an input device to interact with a Virtual Environment (VE). Eye trackers have existed for a number of years, but their use has largely been confined to laboratory experiments. The equipment is gradually becoming sufficiently robust and inexpensive to consider use in real user-computer interfaces. Recently, various companies (e.g. SensoMotoric InstrumentsTM, [10]) announced an eye-tracking add-on to the Oculus Rift Development Kit 2 (DK2) HMD. What is now needed is research in appropriate interaction techniques, in our incorporate eye movements into the user-computer dialogue in a convenient and natural way. It has been used in the past as an input device to interact with a 2D UI [2, 5, 12], but never for an immersive 3D UI. By moving their eyes, users can manipulate an onscreen cursor to point virtual items on the interface and then activate them via prolonged fixations or blinking. Previous research investigated eye-tracker based interfaces for disabled users [4, 5, 12]. Physically disabled users can benefit the most from VR technology since they usually greatly depend on computer aid for recreation or basic communication. In this sense, VR can be viewed as a form of HCI in which information flows and interacts between the user and the technology.

The simplest solution would be to substitute an eye tracker directly for a mouse. In simple words, install an eye tracker and use its x, y output stream in place of that of the mouse. Changes in the user’s line of gaze, where he “looks”, would directly cause the mouse cursor to move.

In this thesis, we present an innovative approach to design an immersive gaze-controlled Multimedia User Interface (MUI) [11] for an eye-tracked headset. We focused on a Multimedia Interface and not a simple UI since we anticipate that VR will allow users to experience VR in everyday environments such as their own home or office. Multimedia applications such as mail composing, picture viewing and gaming that are very coming during everyday usage. Thus, the presented interface covers five commonly daily operated actions; picture viewing, listening to music, mail composing, text editing and gaming; a tiles game and a 3D rendition of the classic flappy bird game [3].

1.2 Thesis Structure

This thesis is organized to provide a continual narrative to be followed. There are seven chapters that are set out as follows.

After this Introduction, the 2nd Chapter acts as a prologue and introduces the reader to the fields that this thesis is seated and relevant. Fields like VE's, the human eye's anatomy and eye/head-tracking.

The 3rd Chapter introduces the software, API's and Tools, used to for the implementation and design of the Gaze Controlled Multimedia User Interface (MUI). Those are the Eye-Tracking and Head-Tracking Software and the game engine Unity3D.

In the 4th and 5th Chapter the implementation of the Multimedia User Interface (MUI) as presented to the users wearing the HMD is described. The steps taken to create the 3D scenes, and in general all the adjustments that where made within Unity and the API's and tools which were used, are presented. Also, all technical issues that occurred are explained as well as decisions taken to address them. Some basic examples and code sources are demonstrated.

The 6th Chapter describes the tests that were made in order to check the accuracy and the results of the project. Explains what changes and adjustments were made after tests with users; changes in code, parameters, data, MUI components and design.

The 7th Chapter draws the conclusions from the work done for this thesis as also some comments of potential future work and the 8th Chapter is the Bibliography.

Chapter 2 – Background

2.1 Human Computer Interaction

Human Computer Interaction (HCI) involves the study, planning and design of the interfaces between a user or people and computers or electronic devices. It is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of phenomena surrounding them. The study of HCI focus on creating systems that minimize the barrier between the human's mental model of what we want to accomplish and the computer's support of the user's task. HCI is becoming more important everyday as computer systems become more integrated into everyday life. It is often an underestimated aspect of programing or system architecture that can have significant impact on how a system is used by the majority of users.

Figure 1: Human Computer Interaction was not considered

2.2 Virtual Reality

VR is a computer-generated simulation of an environment that is interacted with a physical or seemingly real way by a person using special electronic equipment. Equipment such as a helmet with a screen inside or gloves fitted with sensors allow that person to become part of this virtual world and whilst there allow him to manipulate objects or perform series of actions. The concepts behind VR are based upon theories about a long held human desire to escape the boundaries of the 'real world' by embracing cyberspace. Once there we can interact with this VE in a more naturalistic manner which will generate new forms of HCI.

VR allows one to get full blast of information to the senses and hence can allow us to enjoy the world and satiate our senses at a fraction of the cost in terms of energy and resources. Most current VR environments are primarily empirical experiences, displayed either on a computer screen or with special stereoscopic displays, and some regulated simulations include additional sensory information and emphasize real sound through speakers or

headphones targeted towards witnesses. Many newer environments include touch or force feedback through a haptic device such as a 'data glove' which further enhances the experience.

VR allows someone to walk around a three-dimensional building, perform a virtual operation, play a multi-user game, take part in a theatre of war, interact with an artwork and much more! Plus the fact that they can do this in a 3D environment means that they replicate an experience similar to that in the real world but without many of the dangers and is much more preferable to trying to simulate these experiences in a two-dimensional setting, e.g. a computer desktop.

Virtual interfaces and the information environments they produce provide new alternatives for communicating information to users and considered to have beneficial impact in the field of medicine, for example surgery simulations or in dentistry, but also in education, sports, gaming, architecture and the military.

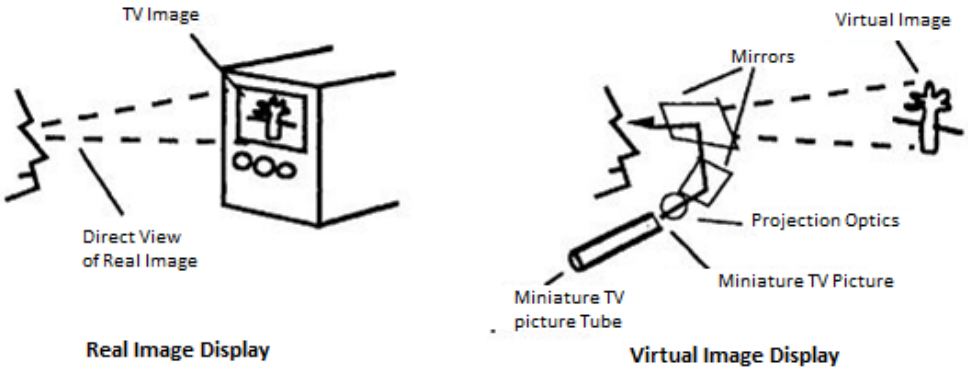


Figure 2: Real Image Display vs. Virtual Image Display

A virtual display leads to a different visual experience than viewing an image using a standard computer monitor. Instead of viewing directly one physical display screen, it is projected into the eyes by optical lenses and mirrors (HMD) so that the original image appears to be a large picture or full scale three-dimensional scene suspended in the world [7]. The aim is to move beyond standard forms of interaction such as the keyboard and mouse which most people work with on a daily basis. This is seen as an unnatural way of working which forces people to adapt to the demands of the technology rather than the other way around. But a VE does the opposite. It allows someone to fully immerse themselves in a highly visual world which they explore by means of their senses. This natural form of interaction within this world often results in new forms of communication and understanding.

2.3 Virtual Reality Systems

The systems used to present and interact with a VE can vary widely. The system hardware often consists of a number of input devices and one or more displays, along with the central processing units.

There are many different types of VR systems but they all share the same characteristics, such as the ability to allow the person to view three-dimensional image. However the main concern regards the VE's content and form. These images appear life-sized to the person. Plus, they change as the person moves around the environment which corresponds with the change in their field of vision. The aim is for a seamless join between the person's head and eye movements and the appropriate response, e.g. change in perception. This ensures that the VE is both realistic and enjoyable. A VE should provide the appropriate responses – in real time- as the person explores their surroundings. The problems arise when there is a delay between the person's actions and system response or latency which then disrupts their experience. The person becomes aware that they are in an artificial environment and adjusts their behavior accordingly which results in a stilted, mechanical form of interaction. The aim is for a natural, free-flowing form of interaction which will result in a memorable experience.

2.3.1. Immersion

Immersion is the user's experience of feeling as if being a part of the virtual world. It is a description of a technology that describes the extent to which the computer displays are capable of delivering an inclusive, extensive surrounding or a vivid illusion of the reality to the senses of a participant. It is basically a unique experience that is connected with the world of VR. When we say that a user is effectively immersed into a VR we simply mean that his senses create the illusion of being in the real world and becomes unaware of his real surrounding.

The level of immersion provided by a system is determined by the following "constraints": the ability of the technology, the ability of the senses, and perception, which is made through interpretation. Two are the main components of virtual reality immersion, depth of and breadth of information. The depth of information is the amount and quality of data the user is given during interacting with the virtual environment. This can refer to the display resolution, the complexity of graphics, and the clarity of audio output. Breadth of information is the number of sensory dimensions that are presented. To have a wide breadth of information, the virtual reality immersion must stimulates all the senses. Virtual reality immersion experiences prioritize audio and visual components over the other sensory factors. Scientists and engineers are studying ways to incorporate the sense of touch into

virtual reality immersion. Known as haptic systems, they provide users feedback and touch interaction.

The most out of immersion is gained when the user explores life-size VR environments and thus forgets about his real world scenario, forgets his present identity, situation and life and immerses him in a world of imagination, adventure and exploration. He gets more focused about his newly created identity inside the VR world.

2.3.2. Stereoscopy

Stereoscopy is a technique for creating or enhancing the illusion of depth in an image by means of stereopsis for binocular vision. Most stereoscopic methods present two offset images separately to the left and right eye of the viewer. These two-dimensional images are then combined in the brain to give the perception of 3D depth. This technique is distinguished from 3D displays that display an image in three full dimensions, allowing the observer to increase information about the 3-dimensional objects being displayed by head and eye movements. It is important to note that since all points in the image focus at the same plane regardless of their depth in the original scene, the second cue, focus, is still not duplicated and therefore the illusion of depth is incomplete. There are also primarily two effects of stereoscopy that are unnatural for the human vision: first, the mismatch between convergence and accommodation, caused by the difference between an object's perceived position in front of or behind the display or screen and the real origin of that light and second, possible crosstalk between the eyes, caused by imperfect image separation by some methods.

In chapter 2.4.3 we will give more emphasis on what stereoscopy is.

2.3.3. Virtual Reality Gear - Head Mounted Displays

In this paragraph we will describe the input devices that are used in VEs. As an input device or sensor we refer to those devices that capture the participant’s actions and send this information to the computer/system which is in charge of the interactive simulation. An input device is considered as a virtual input device or VR gear when they use the paradigm of the implicit interaction or they give 3D input to the system. So, we refer to the devices, clothing or equipment, which are worn by the user that engages in VR.

Such devices are the Virtual glasses or goggles, data gloves, HMDs, data suits, workbenches, joysticks, keyboards, but also haptic devices which enable the sense of touch when manipulating an object the VE. Since this project was based on a HMD we will show emphasis only on that VR input device.

A HMD is generally a display of computer that one wears on his head. It is a display device, worn on the head as a part of a helmet that has a small optic display on one (monocular HMD) or both eyes (binocular HMD). Mostly all HMDs consist of a screen for individual eye and that is what creates a sense any images the user looks at has some depth [7]. It appears to be highly three dimensional or real-life like sized. HMDs also contain a tracking device to make sure that wherever the user points his head at and see wherever direction he feels like the monitor will remain in front of his eyes all the time. This also changes their point of view. A typical HMD has either one or two small displays with lenses and semi-transparent mirrors embedded in a helmet, eyeglasses (also known as data glasses) or visor. The display units are miniaturized and may include CRT, LCDs, Liquid crystal on silicon (LCos), or OLED. Some vendors employ multiple micro-displays to increase total resolution and Field of View (FoV). Some HMDs include speakers or headphones. This is included to make sure both audio and video output is received. Most HMDs are attached to the CPU of the VR systems through cables but also there are wireless systems. However wireless systems do not have the eligibility to avoid lag.

Figure 3: The NVisorTM SX111 HMD used in this thesis

The Head Mounted Display (HMD) used in these experiments is the NVIS SX111 . One of its significant characteristics is the wide field-of-view of a total 102^{o} for both eyes. A 3-degrees of freedom (rotational) head-tracker was attached to this HMD acquiring the user's head rotational direction. The HMD makes use of partial overlap stereoscopic method in order to produce stereoscopic vision

2.4. The Human Eye

"The eye is the window of the soul... The eye is the window of the human body through which it feels its way and enjoys the beauty of the world." (Da Vinci, 1452-1519)

It is obvious that our visual system is an essential human factor to be taken into account when designing VR hardware and software. The Eye has been called the most complex organ in our body. It accommodates to changing lighting conditions and focuses light rays originating from various distances from the eye. Light is converted to impulses and conveyed to the brain where an image is perceived.

2.4.1. Anatomy of the Eye

In order to understand how eye tracking works, we must understand the human eye's anatomy, its components and operation. The eye is made up of three coats, enclosing three transparent structures. The outermost layer, known as the fibrous tunic, is composed of the cornea and sclera. The middle layer, known as the vascular tunic or uvea, consists of the choroid, ciliary body, and iris. The innermost is the retina, which gets its circulation from the vessels of the choroid as well as the retinal vessels, which can be seen in an ophthalmoscope.

Within these coats are the aqueous humor, the vitreous body, and the flexible lens. The aqueous humor is a clear fluid that is contained in two areas: the anterior chamber between the cornea and the iris, and the posterior chamber between the iris and the lens. The lens is suspended to the ciliary body by the suspensory ligament (zonule), made up of fine transparent fibers. The vitreous body is a clear jelly that is much larger than the aqueous humor present behind the lens, and the rest is bordered by the sclera, zonule, and lens. They are connected via the pupil.

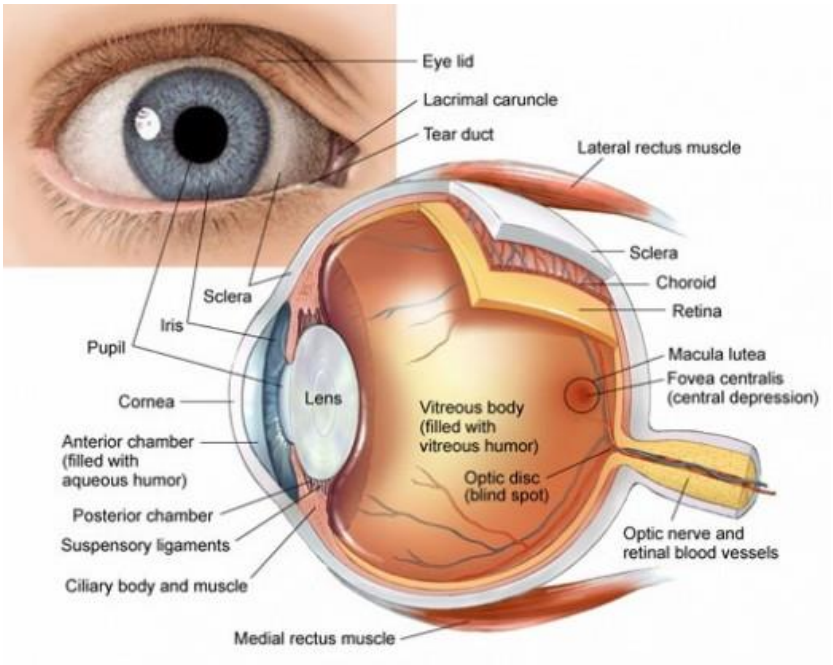


Figure 4: The Basic Anatomy of the Human Eye

Below, the basic front layer parts of the eye are listed and described.

The cornea, is the transparent, outer "window" and primary focusing element of the eye. The outer layer of the cornea is known as epithelium. Its main job is to protect the eye. The epithelium is made up of transparent cells that have the ability to regenerate quickly. The inner layers of the cornea are also made up of transparent tissue, which allows light to pass.

The pupil is the dark opening in the center of the colored iris that controls how much light enters the eye. The colored iris functions like the iris of a camera, opening and closing, to control the amount of light entering through the pupil.

The part of the eye immediately behind the iris that performs delicate focusing of light rays upon the retina, is called the lens. In persons under 40, the lens is soft and pliable, allowing for fine focusing from a wide variety of distances. For individuals over 40, the lens begins to become less pliable, making focusing upon objects near to the eye more difficult. This is known as presbyopia.

Retina is the membrane lining the back of the eye that contains photoreceptor cells. These photoreceptor nerve cells react to the presence and intensity of light by sending an impulse to the brain via the optic nerve. In the brain, the multitude of nerve impulses received from the photoreceptor cells in the retina are assimilated into an image.

The fovea is the most central part of the retina. This area is responsible for the clearest vision with sharpest colors and details.

2.4.2. Vision: The Eye's Functionality

The human eye works much like a digital camera. Light rays reflected off an object enter the eye through a transparent layer of tissue known as the cornea. As the eye's main focusing element, the cornea takes widely diverging rays of light and bens them through the pupil, the dark, round opening in the center of the colored iris.

The lens of the eye is located immediately behind the pupil. The purpose of the lens is to make the delicate adjustments in the path of the light rays in order to bring the light into focus upon the retina, the membrane containing photoreceptor nerve cells that lines the inside back wall of the eye. The central part of the retina is named the macula and the most central part of the macula is the fovea. The fovea is the area at which we have the sharpest vision. When looking directly at an object, the light form it is projected onto the fovea. Although light is admitted through the pupil it is attenuated by the iris, which controls the level of light falling on the retina. The lens of the eye changes its shape to focus the light it passes through towards the retina. The outer, white part, of the eye is the sclera. The

photoreceptor nerve cells of the retina change light rays into electrical impulses and send them through the optic nerve to the brain where an image is perceived.

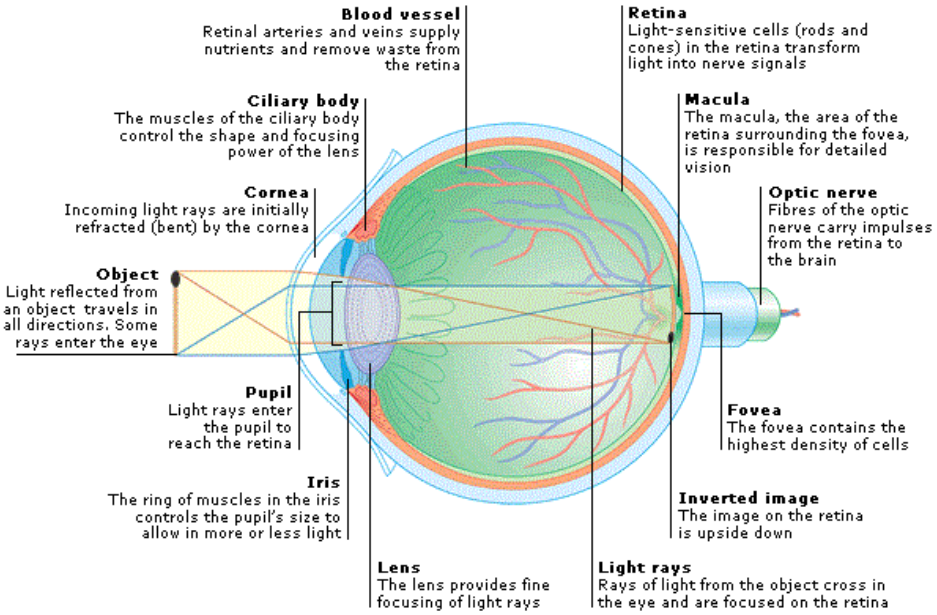


Figure 5: How Each Part of the Eye Interacts and Adjusts to the Incoming Light/Image Display

The entire eye is rotated by six muscles allowing it to move (up, down, side to side, rotate) with great flexibility. An inner muscle, the Medial Rectus, an outer, the Lateral Rectus, an upper, the Superior Rectus, a lower, the Inferior Rectus, an upper and running obliquely, the Superior Oblique, and a lower and running obliquely , the Inferior Oblique.

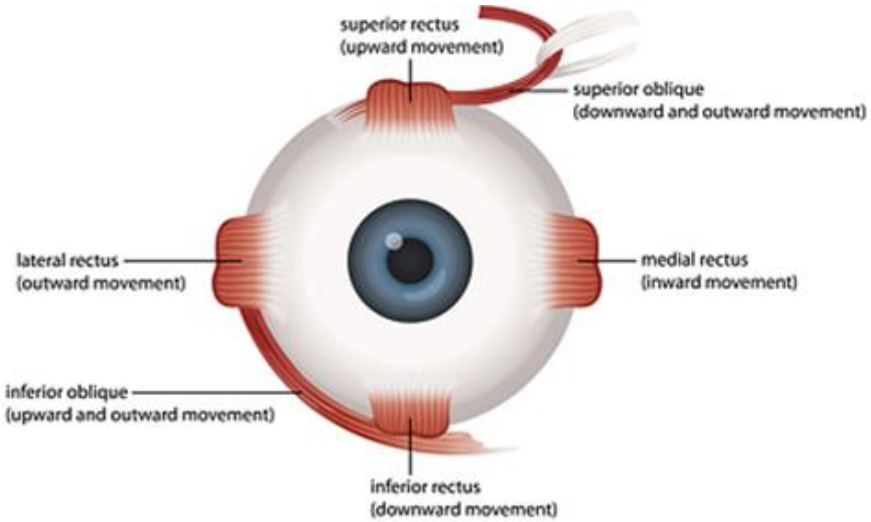


Figure 6: The Human's Eye Muscle Anatomy

2.4.3. Vision Types

There are two types of vision, binocular and monocular vision. Binocular is the vision in which both eyes are used together and on the other hand, monocular vision, is the vision in which each eye is used separately.

Humans have a maximum horizontal FoV of approximately 180-190 degrees with both eyes, 120 degrees that make the binocular FoV (seen by both eyes) flanked by two monocular fields (seen by only one eye) of 40 degrees. One basic advantage of binocular vision is that it gives stereopsis; in which binocular disparity (or parallax) provided by the two eyes' different positions on the head precise depth perception. As for the vertical FoV, it is approximately 120-130 degrees.

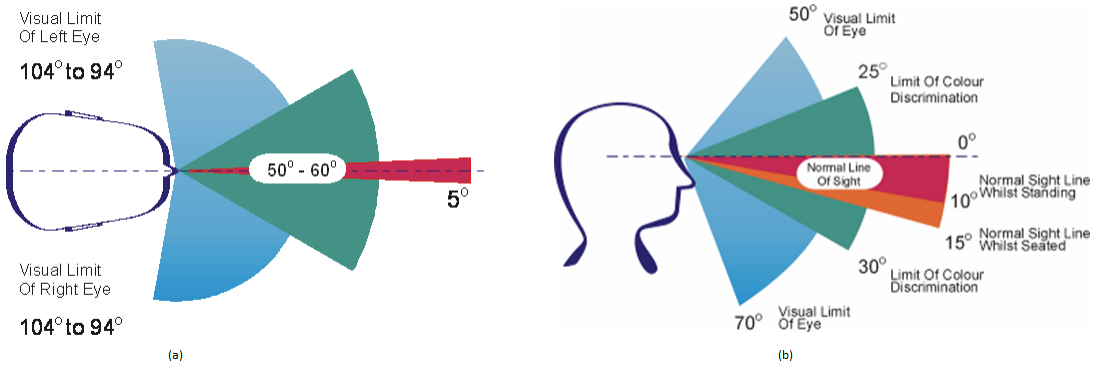


Figure 7: (a) Horizontal Field of View (b) Vertical Field of View

Stereopsis is used to refer to the perception of depth and 3-dimensional structure obtained on the basis of visual information deriving from two eyes by individuals with normally developed binocular vision. Binocular vision results in two slightly different images projected to the retinas of the eyes because of the different lateral positions the eyes are located on the head. These positional differences are referred as binocular disparities. These disparities are processed by the brain to yield depth perception. While binocular disparities are naturally present when viewing a real 3-dimensional scene with two eyes, they can also be simulated by artificially presenting two different images separately to each eye using the method of stereoscopy.

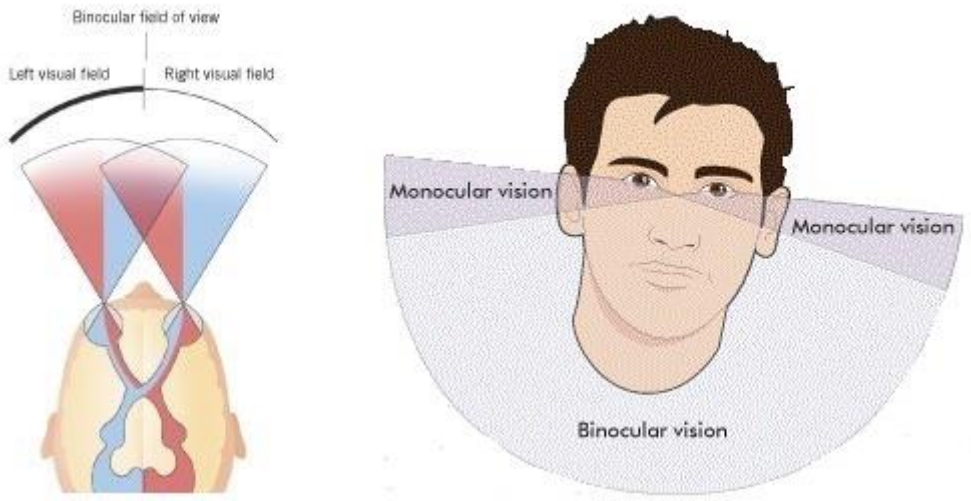


Figure 8: Binocular vs. Monocular Vision

On the other hand using the eyes separately, monocular vision, the FoV is increased while depth perception is limited. Monocular vision implies that only one eye is receiving optical information, the other one is closed. The perception of depth and 3-dimensional structure is, however, possible with information visible from one eye alone, such as differences in object size and motion parallax (differences in the object over time with observer movement), though the impression of depth in these cases is often not as vivid as the obtained from binocular disparities.

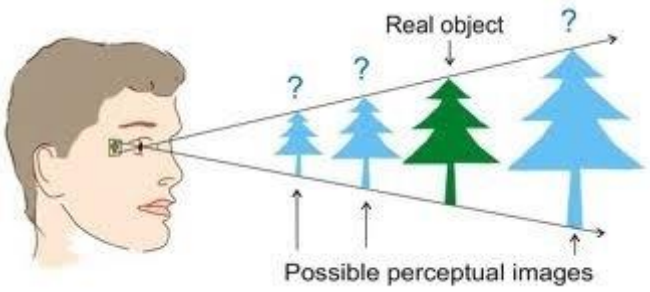


Figure 9: Perception of a plane object in monocular vision

Therefore, the term stereopsis can refer specifically to the unique impression of depth associated with binocular vision; what is colloquially referred to as “seeing in 3D”.

2.4.4. Eye Movement and Control

As eye movement both voluntary and involuntary movement of the eyes that help acquiring, fixating and tracking visual stimuli are considered. The human eye has numerous parts that

must be controlled. Below there will be a simple reference of the most major types of eye movements and only the most commonly used in past research and in this thesis will be described. The main types of eye movements is a saccade, a miniature, a pursuit, a smooth, a compensatory, a vergence, a nystagmus and a blink.[23,24,25]

The main measurements used in eye-tracking research are fixations and saccades.

A **saccade** is a rapid eye movement, a jump, which is usually conjugate and under voluntary control but ballistic; once they are initiated, the path of motion and destination cannot be changed. It takes about 100 to 300 milliseconds to initiate a saccade, i.e. from the time a stimulus is presented till the eye starts moving, and another 30 to 120 milliseconds to complete the saccade. The purpose of these movements is to bring images of particular areas of the visual world to fall onto the fovea. Saccades are therefore a major instrument of selective visual attention. It is often convenient to consider both that a saccadic eye movement always occurs in a straight line and also that we do not 'see' during these movements.

Fixation is the moment, in average 218 milliseconds, when the eyes are relatively stationary, between saccades, taking in or encoding information. Fixation duration provides an index of the speed with which information is processed. Increasing fixation duration is associated with tasks that require more detailed visual analysis. Frequency of fixations often serves as a measure of sampling quantity. They can reveal the amount on processing being applied to objects and therefore studying them can tell us about the complexity or salience of an object in an interface.

A **scanpath** describes a complete saccade-fixate-saccade sequence. In a search task, for example, an optimal scan path is viewed as being a straight tile to a desired target, with relatively short fixation duration at the target. For both long-lasting and long scanpaths, when detected, less efficient scanning is indicated. Also, comparing saccade times to fixating times helps in research.

Blinking is the automatic rapid closing of the eyelid. If detected can be used as an input device, for example as a mouse click. Blink rate can be used as an index of cognitive workload. A lower blink rate is assumed t indicate higher workload and a higher blink rate fatigue. However, blink rate may be determined by other factors such as ambient light levels

Drifts are slow movements away from a fixation point. Flicks or micro saccades reposition the eye on the target. Predominantly these are corrective movements, correcting for the off-center foveal position produced by a drift eye movement. Irregular slow movements of the eye also occur. High frequency tremor causes the image of an object to constantly stimulate cells in the fovea.

2.4.5. Perception in an Immersive Virtual Environment

Perception of our immediate environment is not based on what we actually see or what is there. It is based upon little actual sensory information and is for the most part illusory. Theoretically everything we 'see' around us exist as a model in our minds. We rely upon our perceptive system so much that it enables us to be fooled.

When immersed in a VE our compelling senses are presented with an alternative view of our local environment whilst the real world is shut out. Our perceptual system is trained over many years to recognize our everyday reality, thus has no experience to distinguish it from the VR. An IVE simply provides cues that are a sufficient match for our inner conceptual models of what it is to be in an environment. For instance, stage magicians rely upon this fact by providing basic cues that purposely misinform our perceptual system and leave us wondering how we have apparently jumped from one world-state to another. Just as when we see an illusion and are able to accept the perhaps 'odd' perspective that is implied, when we view an IVE we can accept the virtual world perspective implied over the real world.

2.5. Eye Tracking

Eye tracking is a technique whereby an individual's eye movements are measured so that the researcher knows both the point of gaze ("where a person is looking") at any given time and the sequence in which their eyes are shifting from one location to another. Eye tracking can be used in two main ways, to improve UIs and to understand human behavior. Tracking people's eye movements can help HCI researchers understand visual and display-based information processing and the factors that may impact upon the usability of system interfaces. In this way, eye-movement recordings can provide an objective source of interface-evaluation data that can inform the design of improved interfaces. Eye movements can also be captured and used as control signals to enable people to interact with interfaces directly without the need for mouse or keyboard input, which can be a major advantage for certain populations of users such as disabled individuals[4, 5, 12].

Eye trackers are used in visual system research, in psychology, in cognitive linguistics and in product design. There are a number of methods for measuring eye movement. The most popular variant uses video images from which the eye position is extracted. Other methods use search coils or are based on the electro-oculography.

The HMD employed in this research included embedded binocular eye tracking by Arrington Research.

2.5.1 History & Methods

Many methods have been used for eye tracking since a 100 years ago. Eye trackers measure rotations of the eye in one of several ways, but principally they fall into three categories:

One type uses an attachment to the eye, such as a special contact lens with an embedded mirror or magnetic field sensor, and the movement of the attachment is measured with the assumption that it does not slip significantly as the eye rotates. Measurements with tight fitting contact lenses have provided extremely sensitive recordings of eye movement, and magnetic search coils are the method of choice for researchers studying the dynamics and underlying physiology of eye movement.

The second broad category uses some non-contact, optical method for measuring eye motion. Light, typically infrared, is reflected from the eye and sensed by a video camera or some other specially designed optical sensor. The information is then analyzed to extract eye rotation from changes in reflections. Video based eye trackers typically use the corneal reflection (the first Purkinje image) and the center of the pupil as features to track over time. A more sensitive type of eye tracker, the dual-Purkinje eye tracker, uses reflections from the front of the cornea (first Purkinje image) and the back of the lens (fourth Purkinje image) as features to track. A still more sensitive method of tracking is to image features from inside the eye, such as the retinal blood vessels, and follow these features as the eye rotates. Optical methods, particularly those based on video recording, are widely used for gaze tracking and are favored for being non-invasive and inexpensive.

The third category uses electric potentials measured with electrodes placed around the eyes. The eyes are the origin of a steady electric potential field, which can also be detected in total darkness and if the eyes are closed. It can be modelled to be generated by a dipole with its positive pole at the cornea and its negative pole at the retina. The electric signal that can be derived using two pairs of contact electrodes placed on the skin around one eye is called Electrooculogram (EOG). If the eyes move from the center position towards the periphery, the retina approaches one electrode while the cornea approaches the opposing one. This change in the orientation of the dipole and consequently the electric potential field results in a change in the measured EOG signal. Inversely, by analyzing these changes in eye movement can be tracked. Due to the discretization given by the common electrode setup two separate movement components – a horizontal and a vertical – can be identified. A third EOG component is the radial EOG channel, which is the average of the EOG channels referenced to some posterior scalp electrode. This radial EOG channel is sensitive to the saccadic spike potentials stemming from the extra-ocular muscles at the onset of saccades, and allows reliable detection of even miniature saccades.

The Eye-tracker from Arrington Research, Inc. that was used in this thesis is based on an optical method and relies on two infrared video cameras for the estimation of eye motion and position by detecting saccades, fixations, drifts.

2.5.2. Features that can be tracked

Gaze direction and gaze point. It is used in interaction with computers and other interfaces, and in behavioral research/human response testing to better understand what attracts people's attention.

Eye-presence detection. The eye-tracking system must first find the eyes, so it is the most fundamental part of eye tracking. It is also used in specific features such as power saving by dimming the screen when eyes are not detected.

Eye position. The ability to calculate the position of the eyes in real time makes the eye tracking system accurate and precise while allowing the user to move freely. Eye position is also used in gaming and auto stereoscopic 3D-display systems.

User identification. The eye tracking system can be used as a multimodal biometrics sensor, such as for logging on to a computer or for car-driver identification. It can combine face identification with physiological eye features and eye movement patterns.

Eyelid closure is used to monitor the user's sleepiness, for instance in advanced driver assistance or operator safety solutions.

Eye movement and patterns are studied to understand human behavior and to assess and diagnose injuries or diseases. You can, for example, perform hearing tests on infants or identify markers of diseases such as Alzheimer's or autism at a very early stage. The study of micro saccades is also central in neurological research.

Pupil size and pupil dilation. Pupil dilation is an indicator of excitement. In combination with eye movement patterns and facial expressions, it can be used to derive emotional reactions, for instance in creating innovative user experiences. Pupil dilation can also serve as a marker of impairment, such as concussion, or drug or alcohol influence.

Chapter 3 – Software Architecture and Development

3.1 Game Engine: Unity 3D

Game Engines are power-tools behind all games we know and love. Unity is one of the most widely-used application for game development and with a huge variety of packages to offer. It is used by everyone, from a hobbyist level to large studios, to create games and interactive experiences for the web, desktops, mobiles, and consoles. With its intuitive and combination of easy to learn toolset and rapid productive workflows, Unity 3D allows the user to minimize the time, effort and construction-cost of interactive content.

Even though Unity is basically used for game development, in this thesis was used to employ an eye tracking control to a unity custom made 3D MUI interactive environment.

3.1.1 The Third Dimension (3D World)

Let's understand how Unity 3D allows you to develop 3D games and interactive environments.

Coordinates

In a screen, each point is uniquely specified by a pair of numerical coordinates (X-axis, Y-axis) which are the signed distances from the point to two fixed perpendicular directed lines measured in the same unit of length. In Unity, and as in all 3D applications, there is the concept of the Z-axis. The Z-axis in addition to the existing X for horizontal and Y for Vertical, represents depth. In 3D applications information on objects are laid out in X, Y, Z format, knows as the Cartesian coordinate method. Dimensions, rotational values and positions in the 3D world can all be described in this way.

Vectors

3D vectors are described in Cartesian coordinate. They are simply lines drawn in the 3D world that have a direction and a length and can be moved in world space but remain unchanged themselves. Vectors are useful as they allow us to calculate distances, relative angles between objects and the direction of them.

Cameras

Cameras act as the viewport for the screen. They can be placed at any point of the 3D world, animated or attached to characters or other objects. There can be multiple existing cameras but only a single main camera can render what the user sees.

Vertices and Meshes

All objects are made up of 2D shapes, known as polygons, and are converted into polygon triangles by Unity itself. A polygon triangle is made up of 3 connected edges and the location at which these edges meet are knows as vertices. By combining many linked polygons Unity allows us to build complex shapes, known as meshes. In 3D projects, the importance of the number of polygons used in it is of utmost as they define the details of it.

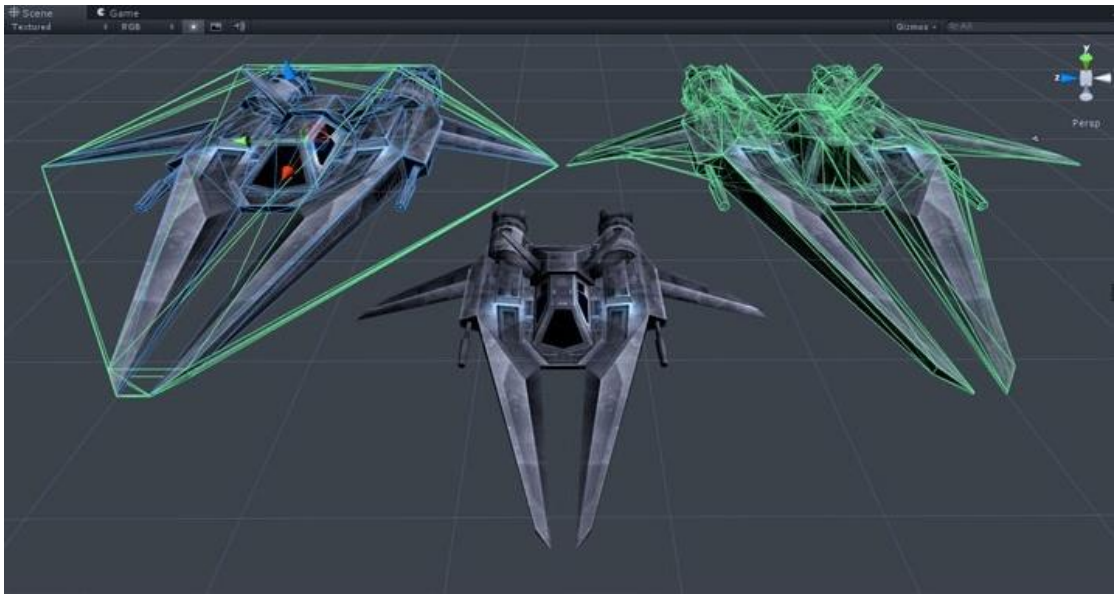


Figure 10: A Spacecraft Model. Left: A Simple Mesh Collider with a Small Polygon Count. Middle: No Mesh Component Right: A Higher Polygon Count Mesh

The polygon count it's usually referred to models, but also to props or the entire scene ('game level').The higher the polygon count the more computer sources are needed to render the project. Because of this, when modeling, developers must consider polygonal detail, and where it is most required.

Materials, Textures and Shaders

Graphical Assets are broken down into materials, textures and shaders.

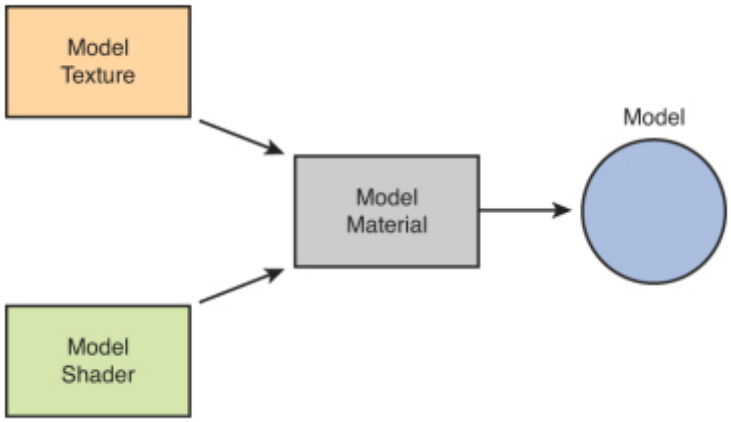


Figure 11: Unity's 3D Basic Components of a 3D Model

Materials provide the means to set visual appearance to 3D models; from basic colorings to image-based surfaces.

Textures are flat images that are applied to 3D objects. They are responsible for models being colorful and interesting instead of blank and boring. How textures work is simple, it's like a label that is "wrapped" around the 3D model. Textures must be scaled to a power of 2 (64px x 64px, 128px x 128px, etc.). Larger textures will give the chance to add more detail to the textured models, but more intensive to render.

In a single material, a script, the shader, is in charge of the style of rendering. For instance, in a reflective shader the material will render reflections of surrounding objects but maintain its color or the look of the image applied as its texture.

Rigidbody Physics

Physics refer to the simulation of certain real world physical phenomena and systems. A physics engine is required in order to provide this real-time simulations and in our case, Unity uses the NVidia's PhysX engine. In simple words, physics engines give games a mean of simulating realism in physical terms, not only physical reactions such as mass and gravity but realistic responses to friction, torque and mass-based impact. This realistic motion is implemented by using the Rigidbody dynamic system.

The influence of the physics engine on objects is called force. Forces are applied through scripting or by applying components. So in order to invoke physics on an object in Unity a Rigidbody component must be added. There is no assumption that the object should be affected by physics; physics consume a lot of processing power and in general computer resources. Rigidbody physics component is not given to any object but only to those you wish to be control under the physics engine, and ideally to any moving object, so that the physics engine is aware of it and to save on performance.

Having added a Rigidbody component u can adjust the Mass parameter, the weight of the object in kg, the Drag parameter, the amount of air resistance affecting an object as it moves, the Angular Drag parameter, affects the rotational velocity, the Use Gravity parameter, it sets whether or not and in what degree the object is affected by gravity, the Is Kinematic parameter, option that allows you to have a Rigidbody object that is not affected by the physics engine, the Interpolate/Extrapolate parameter, that smoothens the transform movement of an object, and the Constraints parameter, that locks objects so that they do not move or rotate as a result of forces applied by the engine.

Collision Detection

Objects in games and 3D interactive applications interact with the player, the environment and each other. By giving an object a Collider component we place an invisible net around it. This net usually mimics its shape, mesh shaped colliders, but also primitive shapes can also be chosen, Box, Sphere, Capsule, Wheel colliders, and is in charge of reporting any collisions with other colliders, making the game engine respond accordingly.

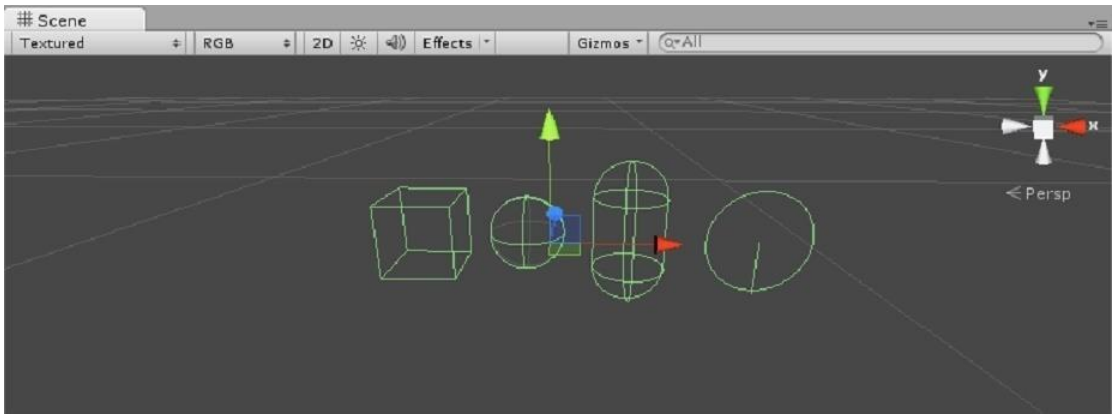


Figure 12: Unity 3D Primitive Shapes and Colliders(Cube, Sphere, Capsule, Wheel)

The act of detecting and retrieving information from these collisions is knows as collision detection. Instead of detecting when two collisions interact we can also detect when colliders are intersecting (trigger mode collision detection) and even pre-empt collision detection. There is also the Raycasting technique, that doesn't detect intersecting colliders but draws an invisible (non-rendered) vector line between two points. Raycasting can also be used by reading the length of the ray and therefore get the distance of 2 colliders/objects and also the point of impact of the end of the line, in other words the object that the ray faces and lands on.

In this thesis, Raycasting is used to detect what and where the user is looking at by translating the eye-tracking data.

3.1.2 Unity's Essential Concepts

Unity establishes the use of the Game Object concept. By this, the developer breaks down part of the project into easily manageable objects that are made of many individual Component parts. After introducing functionality to each individual object and to each component attached to it the project can expand infinitely in a logical progressive matter.

Here is an overview of some of the key-basic Unity concepts.

Assets

The Assets are the building blocks of Unity projects. From textures in the form of image files, through 3D models for meshes, and sound files for effects, Unity refers to those files you use for your project as Assets.

Scenes

As a Scene, in Unity 3D is considered a level of a game or areas of a project content or even a whole project, if the developer decides to dynamically load all the content at through code in a single scene. By using different Scenes you will be able to distribute loading times and test different parts of the project individually. Only one scene at a time can be open and edited.

GameObjects & Components

All active objects in a Scene are called GameObjects. Each Asset (for instance a model) when placed or instantiated into the current Scene becomes a GameObject. Every GameObject contains at least one Component, the Transform component. The Transform component tells the game engine about the position, rotation and scale of an object. Further required functionality will be applied by adding more components.

Also, Game Objects can be linked to each other. They can be nested in order to create parent-child relationships.

Components are used to apply new parts of the game engine to your object. They can be for creating behavior, defining appearance and influencing other aspects of an object's function. Those are the Rigidbody component, textures, lights, cameras, particle emitters, animations, audio sources, scripts and more.

Scripts

Scripts are components that extend or modify the existing functionality available in Unity. In this thesis our scripts are written in C Sharp (C#) and JavaScript (JS).Scripts are added to objects, also empty objects, and there are no limitations on how many can be used on one.

Prefabs

A prefab is a stored version of an object that can be reused in different parts of your project. By using prefabs, complex objects with various components and settings can be instantiated at any time allowing each instance of it to be individually modifiable.

3.2 Overview of Eye Tracking Device's Software

The software used in this thesis in relation to the eye tracking device embedded in the HMD is the ViewPoint EyeTracker® of Arrington Research, Inc. This software provides a complete eye movement evaluation environment including integrated stimulus presentation, simultaneous eye movement and pupil diameter monitoring, and a Software Developer's Kit (SDK) for communication with other applications. It incorporates several methods from which the user can select to optimize the system for a particular application and methods of mapping position signals extracted from the segmented video image in EyeSpace™ coordinates to the participant's point of regard GazeSpace™

3.2.1. Interface

Only the most basic features of the software will be presented here. For full documentation refer on the user manual of the ViewPoint application found in Chapter 7.Once the program is running, it displays several windows arranged as shown below.

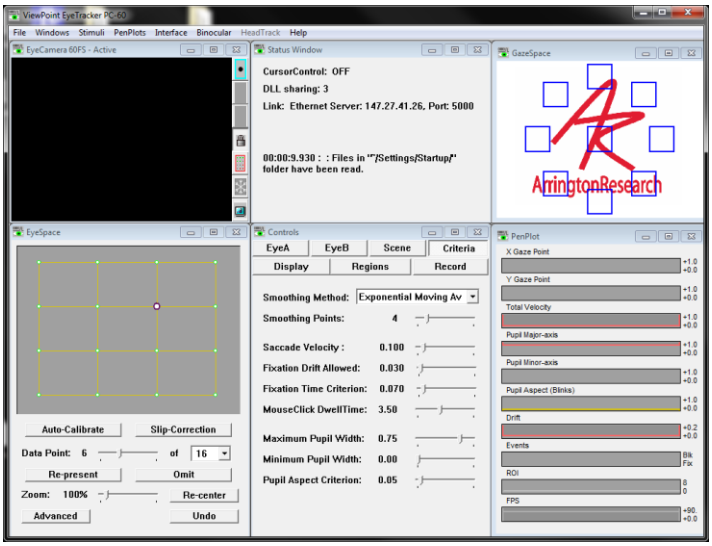
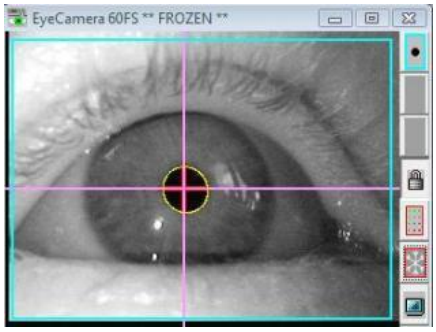


Figure 13:Startup Interface of the ViewPoint EyeTacker Software

There is the EyeCamera window, the EyeSpace window, the Status window, the Controls window, the GazeSpace and the PenPlot windows



The EyeCamera window displays the video image of the eye and the image analysis graphics. It provides controls to make the eye tracking results more reliable and to extend the range of the trace area. Thus, it makes it easy to limit the areas within the software searches for the pupil and corneal reflection to exclude extraneous reflections and shadows.

Figure 14: EyeCamera Window

The EyeSpace window corresponds to the geometry of the EyeCamera image. It displays an array of the relative locations of the pupil, glint, or difference vector, which were obtained during calibration. This provides information about calibration accuracy and allow rapid identification and correction of individual calibration errors by allowing manual recalibration of individual points or the ability to omit problem points. The number, color and presentation rate can be set from here.

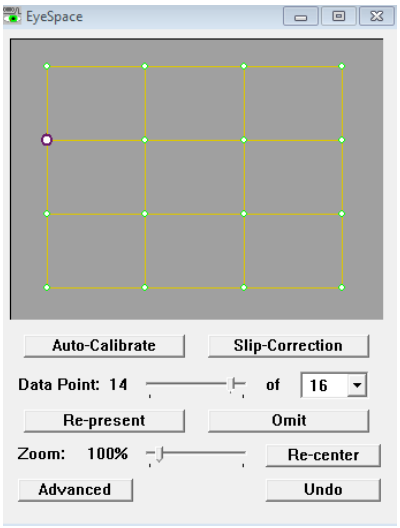
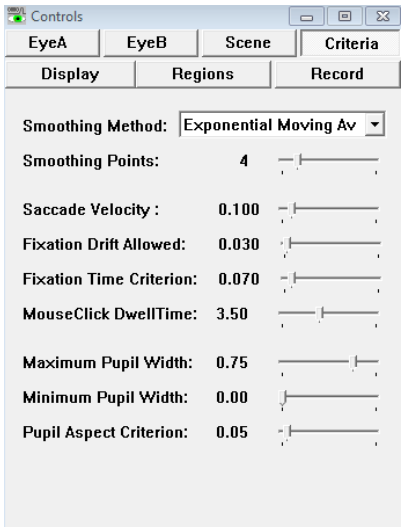


Figure 15: EyeSpace Window



The Controls window allows the user to adjust the image-analysis and gaze-mapping parameter settings and to specify the feedback information to be displayed in both Stimulus window and the GazeSpace window. Eye Image quality adjustments can be made and tacking method can be specified. Also, smoothing and other criteria can be applied to data, has parameters to setup the regions of interest and calibration regions as well as adjust brightness, contrast, hue, saturation of the scene image and to open, pause, close data files or insert markers.

Figure 16: The Controls Window

The Status window gives details about processing performance and measurements. Also, shows the applications that share the same .dll of the application; thus the threads that ViewPoint communicates with.

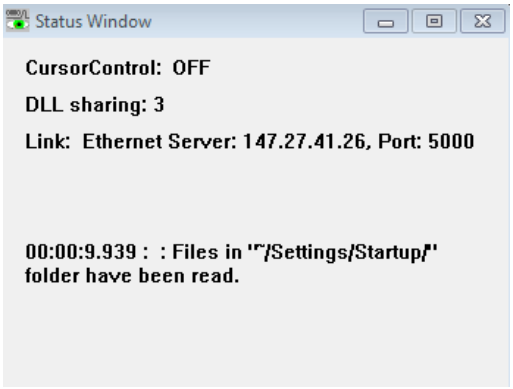


Figure 17: Status Window

Figure 18: Stimulus Window

The Stimulus window is a new window that pops up when calibration starts. It is designed to be full screen, preferably on a second monitor. Upon which may be displayed the subject's calculated position-of-gaze information and region of interest boxes.

The Pen Plot window displays plots of X and Y position of gaze, velocity, ocular torsion, pupil width, pupil aspect ratio, drift, etc. in real time.

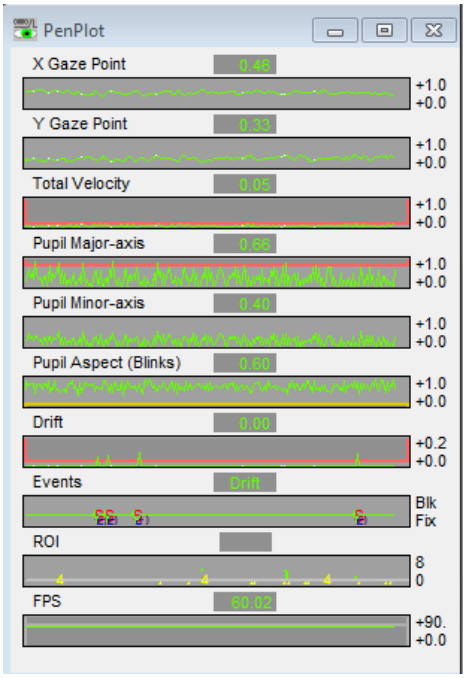


Figure 19: Pen Plot Window

3.2.2 How the software works

In general, how the Viewpoint EyeTracker® works in a typical head fixed configuration will be described. The numbers in this section refer to the items in the figure below.

The infrared light source (*item 1*) serves to both illuminate the eye (*item 2*) and also to provide a specular reflection from the surface of the eye. In dark pupil mode, the pupil acts as infrared sink that appears as a black hole. In bright pupil mode, the "red eye" effects causes the pupil to appear brighter than the iris.

The video signal from the camera (*item 3*) is digitized by the video capture device (*item 4*) into a form that can be understood by a computer. The computer takes the digitized image and applies image segmentation algorithms (*item 5*) to locate the areas of pupil and the bright corneal reflection, the glint. Additional image processing (*item 6*) locates the centers of these areas and also calculates the difference vector between the center locations. A mapping function (*item 7*) transforms the eye position signals (*item 6*) in EyeSpace coordinates to the subject's GazeSpace coordinates. (*item 8*) Because the eye movements are rotational, for example the translation of the eye position signal that is apparent to the camera is a trigonometric function of the subject's gaze angle, the best algorithms are non-linear. Next, the program tests to determine whether the gaze point is inside of any region of interest (ROI) that the user has defined.

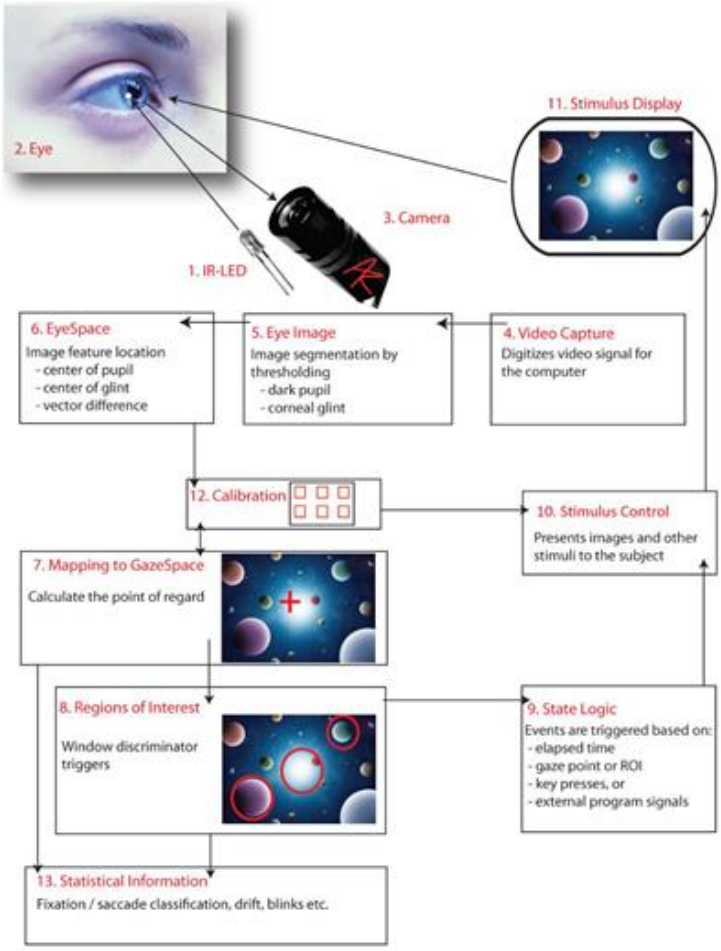


Figure 20: Schematic of the ViewPoint EyeTracker System (Head Fixed)

The calibration system (*item 12*) can be used to present calibration stimuli via (*item 10*) to the user and to measure the eye position signals for each of the stimulus points. These data are then used by the calibration system to compute an optimal mapping function for mapping to position of gaze in GazeSpace.

3.2.2. Binocular Method

The eye-tracker used in this thesis supports both monocular and binocular eye tracking.

By default, ViewPoint is set for monocular eye tracking. So we have to switch from monocular to binocular operation. Once this is done, we will see that another EyeCamera window will appear, for the second eye and in the GazeSpace window there is an additional drop down box to specify which eye the calibration process is to apply to.

In our project we applied binocular eye tracking in order to accomplish no mouse input interaction in our project. The left eyes' point gaze data are used to map where the user is looking at in out virtual scene and the right eye is used to check blinking to apply mouse click by detecting blinks.

3.2.3. Calibration

In order to know where the participant's eyes are fixating on the computer screen, we must first "teach" the computer, in our situation the ViewPoint software, what the eye looks like when the participant is fixated on known locations on the screen. So, prior to using the eye tracker the user needs to undergo a personal calibration process. The reason for this is that each person has a different eye characteristics, and the eye tracking software needs to model these in order to estimate gaze accurately.

The tracker operates by tracking the pupil of the eye, the "dark/black" part of the eye, which will appears as filled-in with blue in the camera-view of the eye, as well as the "corneal reflection", the reflection off of the cornea that appears in yellow on the eye-view. The relative positions and size of these two landmarks are measured as the participant looks at specific points on the screen during calibration. During the rest of the experiment, the tracker figures out where the eye must be looking, depending on the relative size of the pupil as well as the relative location of the pupil and corneal reflection. Anything that disrupts the pupil capture, or interferes with the corneal reflection will cause calibration to be very difficult or even impossible.

In our situation, calibration takes approximately 1-2 minutes to complete and consists in square green target that is displayed at different locations of the screen on a blank background. The user is warned with a “Get Ready” message on the screen to draw the subject’s attention to start the calibration process. During the calibration process, we must ensure that the pupil is accurately located at all times by monitoring the green dots and the yellow oval, i.e., monitoring the image segmentation. As for the calibration points, we must use at least 9 but tests showed us that best calibration results are provided by using 16 or 20 points; for this thesis we used 20. Successful calibration will be indicated by a rectilinear and well separated configuration of green dots corresponding to the locations of the pupil at the time of calibration point capture; the green and yellow dots are show in the EyeSpace window.

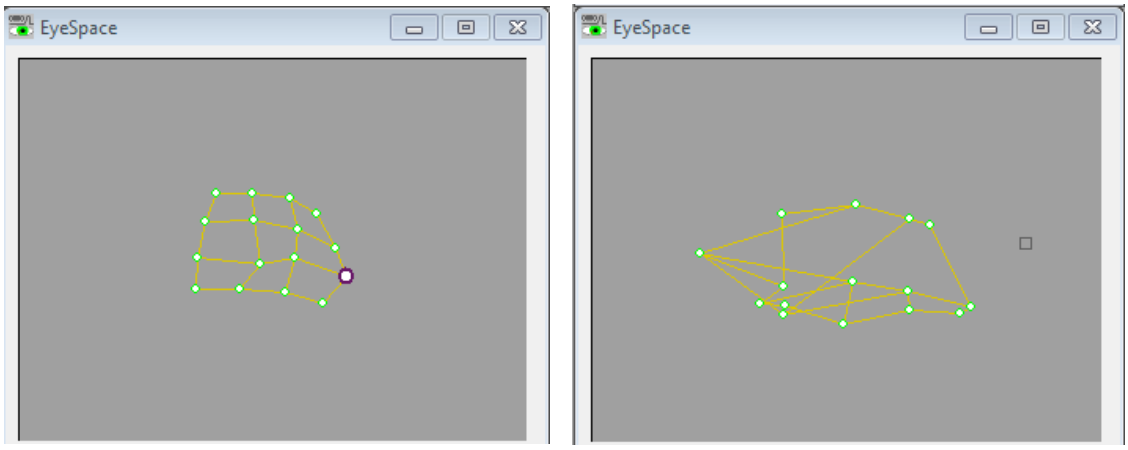


Figure 21: Calibration Attempts. The Left Picture Depicts a Successful Calibration. The Right Picture Depicts a Wrong Calibration

Stray calibration data points can be identified and re-calibrated or omitted. The EyeSpace window allows the user to select stray calibration points to be recalibrated. If a point is selected to be re-calibrated, then it will be re-presented in the screen and the participant will be asked to look at the center of it; to calibrate. This can be repeated with many calibration data points as necessary. If the calibration points are not rectilinear, for example, there are lines crossing then complete re-calibration is necessary. If a particular point cannot be recalibrated, then that specific point can be omitted.

3.2.4. SDK and Communication with Unity3D

In order to implement eye tracking in Unity3D, we should allow communication to be made between both Unity3D and the ViewPoint software.

ViewPoint software includes a powerful developer’s kit (SDK) that allows interfacing with ViewPoint in real-time, giving real-time access to all ViewPoint data. Allowing complete

external control of the ViewPoint EyeTracker, the SDK is based on shared memory in a dynamic-linked library (DLL). The SDK is event/message driven so there is no CPU load from polling and provides microsecond latency.

So, Unity3D interacts with ViewPoint by compiling the VPX_InterApp.lib file, a library file. We will extend further in how communication is established and how constants, data types and functions are accessed by Unity3D in Chapter 5.3.

3.3. Head Tracking Device - Software

3.3.1. Overview

The head tracking device used in this thesis is the InertiaCube3 and the InertiaCube Processor by Thales. It is an inertial three degree of freedom (3-DOF) orientation tracking software. It obtains its motion sensing using a miniature solid-state inertial measurement unity, which senses angular rate of rotation, gravity and earth magnetic field along three perpendicular axes. The angular rates are integrated to obtain the orientation (yaw, pitch, and roll) of the sensor. Gravimeter and compass measurements are used to prevent the accumulation of gyroscopic drift.

Figure 22: The Inertia Cube3 and its Processor

3.3.2 SDK and Communication

The head tracking device has a test software and the InterSense Software Development Kit (SDK). The core of all InterSense software that are associated with the head tracker is a dynamic-linked library, the isense.dll, which must be saved in the Windows system directory. This library, along with all other InterSense libraries provide a standard interface for the device.

Before the tracker is used, it must be configured and a diagnostic tool must be run. During this thesis, these test were run once, right before the testing and experiment phase. This testing tool, is the ISDEMO, validates the communication of the InertiaCube3 to the PC and tests the performance through the above mentioned DLL. In this phase, the compass, perceptual enhancement, sensitivity, prediction, and in general all the tracker's sensor parameters, can be modified and checked so that the device is working properly. Also, tools such as self-system test and compass calibration.

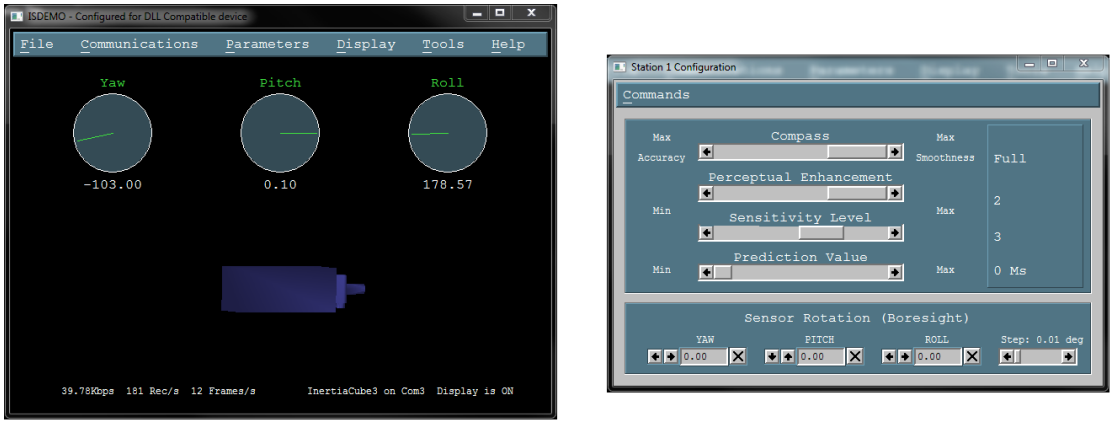


Figure 23: Left – ISDEMO application showing real-time Yaw, Pitch and Roll data. Right – Setting Adjustments for the head tracking device

There is also the InterSense Server Application, ISERVER, which provides multiple services to applications requiring tracker data. It is the link between the head tracker's data output and third party applications, in our situation Unity3D. ISERVER runs in the system tray, reading the data from the connected device at the maximum speed allowed by the operating system. That data is then made available to Unity3D through the InterSense DLL. Inter-process communication will be explained in detail in Chapter 5.4

Chapter 4 – UI Implementation

Having explained the background and all the necessary theories and methods that are used in this thesis, it's about time to explain the implementation of the UI. The way the 3D visual Scenes were designed and how everything was wrapped together in order to get the desired functionality. The first paragraph 4.1 refers to how the interface was modeled and the second paragraph 4.2 refers to how the models composing each scene were placed and configured.

To begin with, let us state again what this project is about. The project aims to create a 3D MUI that will allow the user to control it only with his eyes; it will be mouse free. Six applications based on the eye tracked 3D UI paradigm have been implemented.

A Photo Gallery, allows the user to browse the user's Picture's directory. The application searches through the image folder and visualizes folders, sub folders and files in a virtual 3D slide. All types of images are supported (jpeg, bmp, gif, .png, etc.)

A 3D Music Player, explores the user's Music folder and exposes virtual 3D geometry for audio and playback control. It consists of a 3D Player and virtual speakers that visualize the music and vibrate according to the music tempo, like in real life. Once the user picks a song, he can listen to it, pause it, stop playing it or choose another one. He I also allowed to adjust the volume through the volume panel.

A 3D Email Composer, consists of a 3D custom-made keyboard and a 3D email form. The keyboard is based on a standard mobile device keyboard layout supporting only Latin characters and a set of symbols. In order to compose an email, the user fills his email and password, the receiver's email, the subject and the main body of the mail. After filling the form, the user can either send the mail or clear the form and start over.

A Word Processor, consists of the above mentioned keyboard and a text editor form. The word processor allows the user to create, and therefore save, or edit a .txt file located at the user's Documents directory.

Finally, two immersive mini games were implemented; a puzzle game and an action game. The puzzle game in which a user-selected picture is fragmented in tiles. The tile layout is then randomized and the user has to re-arrange the tiles to form the original picture and solve the puzzle. The action game, is a 3D rendition of the classic flappy bird game [3] but instead including an airplane and a custom made virtual world, representing a valley with cabins, mountains, trees, etc.

4.1 Creating the 3D Visual Content

In order to create the 3D Virtual Scenes that the application would render, several steps were required, from creating the actual 3D object to importing them inside Unity3D and placing them to a virtual scene, as well as creating and assigning the materials to these objects.

A custom outdoor environment was chosen as the rendered displayed environment, divided into six zones according to the developed applications of this project. As mentioned above, the six zones are the pictures' explorer, the music player, the email composer, the text editor, the puzzle, and an action game similar to flappy bird. The 3D MUI elements are considered to be on static locations in the virtual space while a moving observer may interact with them from various locations. Specifically, the six developed applications are centered on a main pivot point while the observer can turn around and select the desired applications. Thus, the main menu is a circular application menu. The 3D models used in this project were either custom made or downloaded from 3D model repositories [19].

The interface, separates each application with a custom made semi-transparent layer or frame displaying the title of each individual application. This frame is actually a flat 3D wall by adjusting the scale of a primitive cube object to meet our needs. Having created the model, then a dark gray transparent material was created and assigned to it. Therefore, our main menu consists of six layers that are decorated with models depending on what each scene represents.

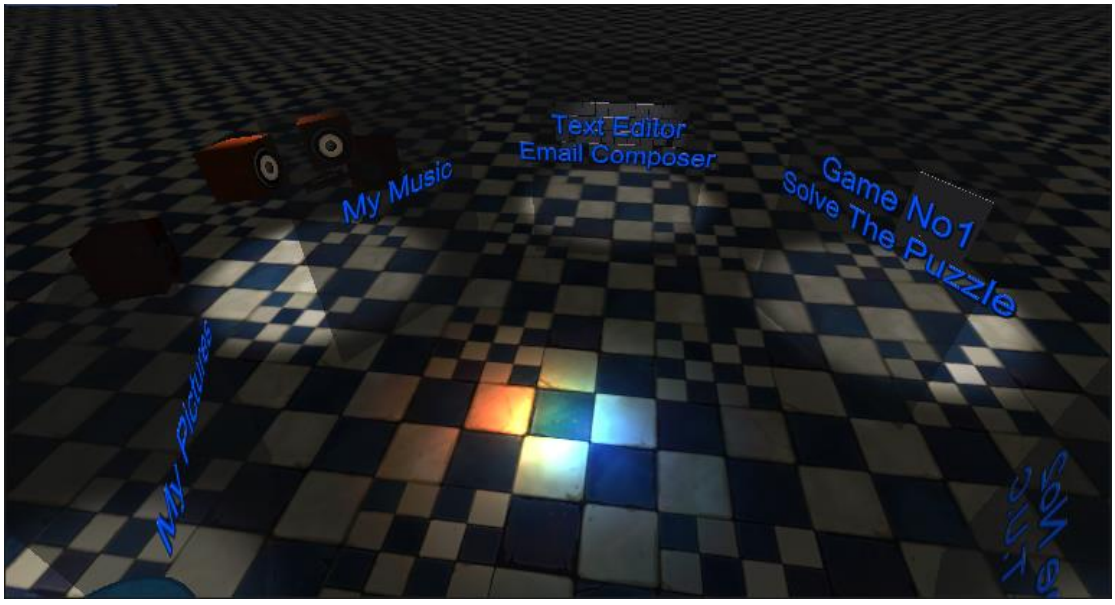


Figure 24 : The Application's Main Interface from a Higher Angle to Display the Individual Applications

For the Picture Explorer, folder and files objects needed to be modelled. As for the scene layout, we represented exactly the preview of the Windows Directory. A folder object, representing a Windows Folder of yellowish color and a file object representing the pictures and assigning the image itself as a texture on it, like a small preview of the picture. Also, in order to view the pictures of a directory, a preview slide model must be created. It is a simple primitive cube object that was modelled in such way to allow us to attach different textures on each face or in other words different picture preview on each side. Finally, we modelled a Back button that would allow the user to jump back to available directories until he reaches the root Directory which is the Windows Picture folder.

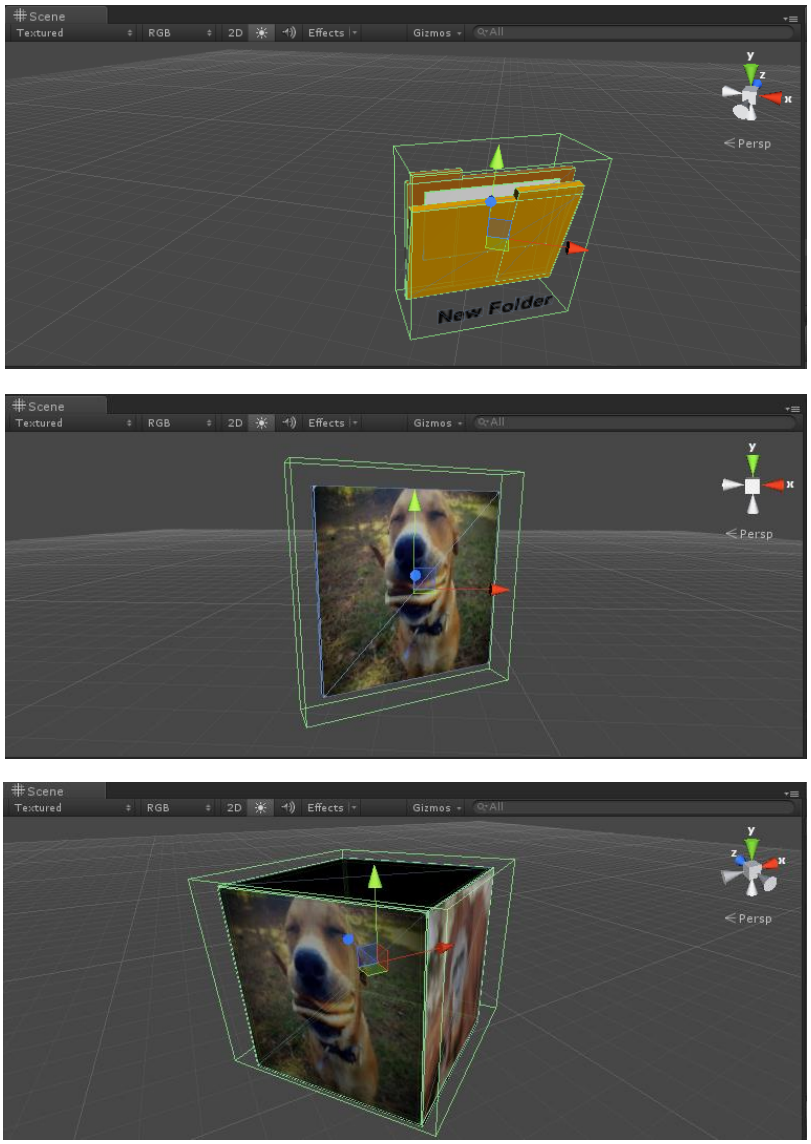
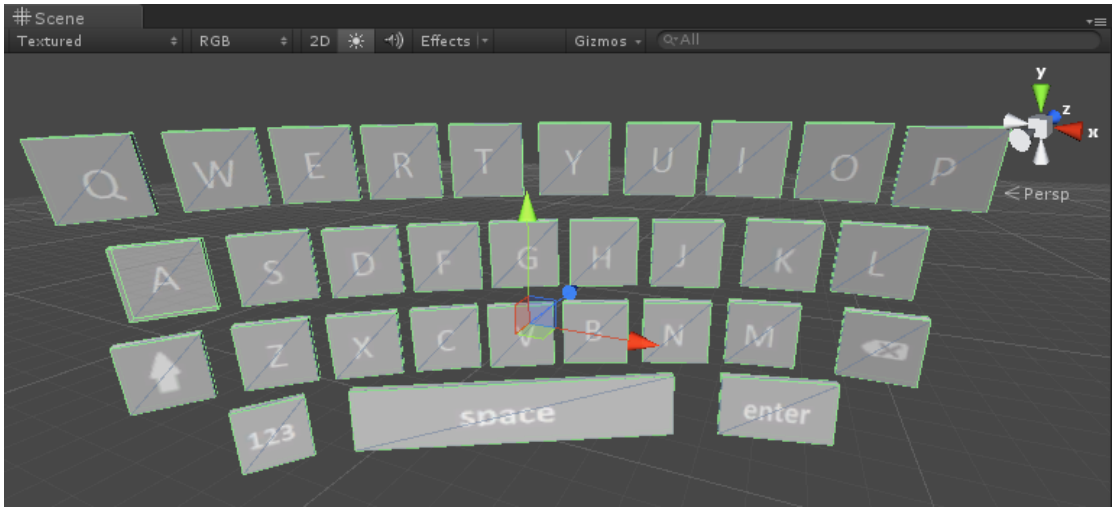
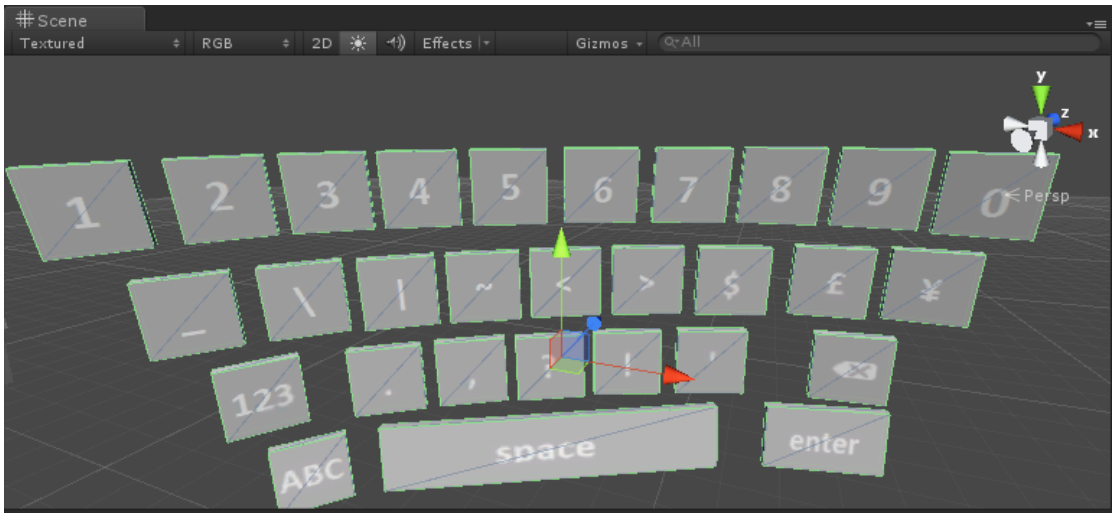


Figure 25: The Folder, Picture and Preview Box Models of the Explorer Scene

For the needs of the Email and Text processor scene, we modeled a 3D virtual keyboard. The keyboard layout is based on a standard mobile layout and supports Latin characters, both

capitals and lower case, and some basic symbols. We modeled each key one by one using Unity's primitive cube objects, scaled and positioned at the desired position and size. Also, each texture and material assigned to the keys are custom made, using Photoshop. The main color material is in the color of gray and the letter font is in white. The keyboard model consists of 3 separate models, each one representing a different layout; the Characters layout and 2 Numbers and Symbols layouts. Below, we present the keyboard model.





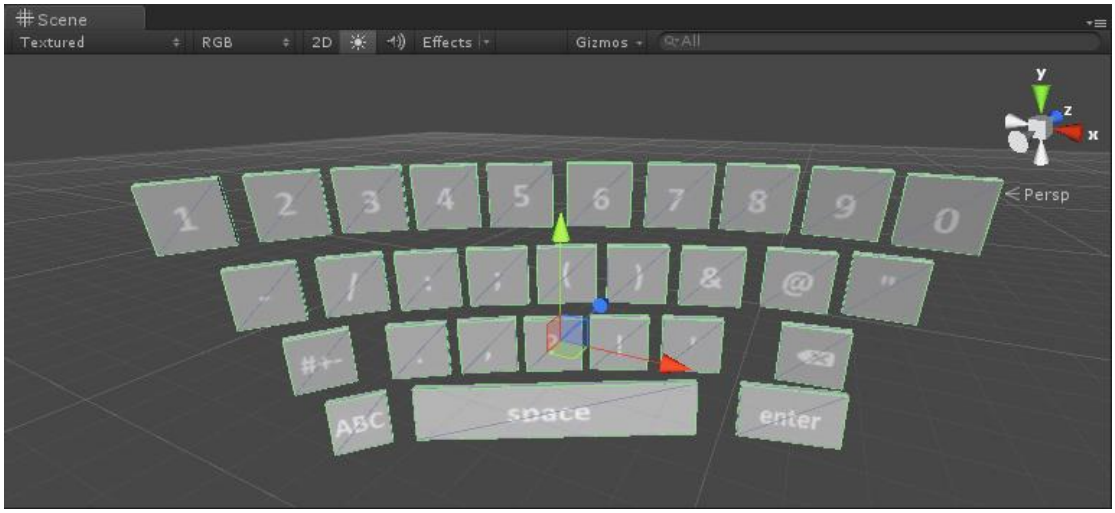
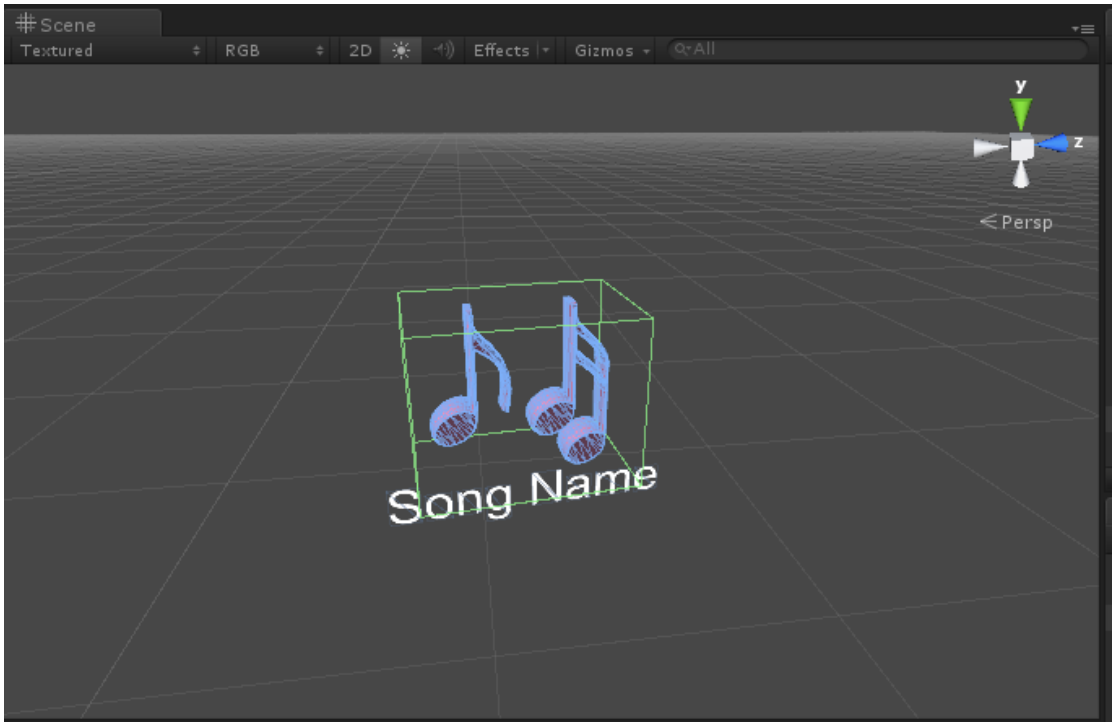


Figure 26: The Keyboard Models. All Three Types are shown. Char Layout, and both Number and Symbol Layouts

As for the Music Player application, models for the music player, the song files and folders, and the speakers were created. The music player, represents a 3D version of a classic 2D music player, with a Play, Pause, and Stop buttons as well as Volume Buttons. The music player is accompanied by 4 speakers. A pair of custom made speakers made of Unity's primitive game objects, such as cubes and cylinders, scaled and positioned according to our needs and a pair of speakers from the model's repository [19].



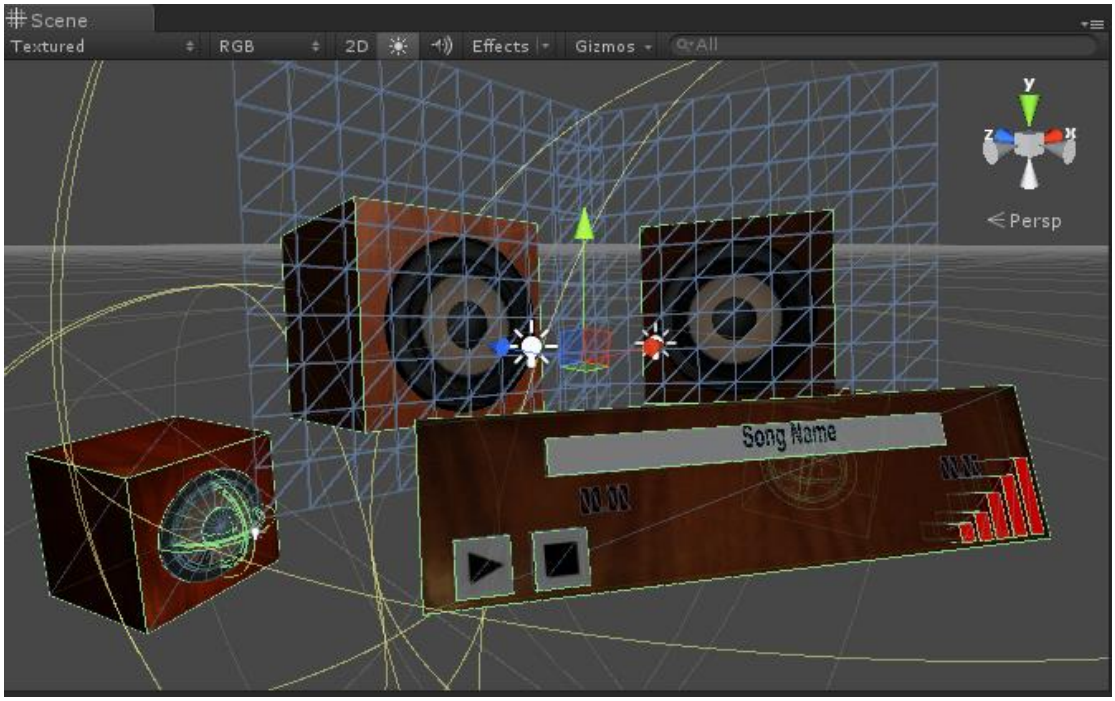


Figure 27: The Song Model, Music Player Model and the Speakers

For the tiles game, the basic object that was created was a model that consists of blocks. We thought of breaking the picture into twenty five blocks making it more challenging and interesting. The model of a picture is therefore divided into twenty five sub blocks; twenty five identical primitive cube objects scaled and positioned in order to depict the picture that the user selects to play with. As for the materials and textures of each block, they are assigned through code and not from the modeling view of Unity3D. Just for reference, since coding and functionality will be explained in detail in the next chapter, empty materials are added at this point to each tile and during runtime and based on the picture that is loaded, the picture itself is the texture of each block but with a specific offset. For example the top left block's texture depicts the first upper left area of the picture if imagined it is divided into twenty five identical areas. Another model of this scene, is a transparent gameobject, exactly identical with the one described above, that will work like a guidance tool in order to guide the user at where he is supposed to place the blocks during the game. This model has the exact scale as the previous one and is also divided into twenty five blocks.

Finally, the most fascinating and complex scene, regarding its rendered components is the action game. This application of the MUI has its own new environment, different from all the previous ones. We have created a whole new virtual world, depicting a valley. From scratch, we have created a whole valley surrounded from mountains, with the help of the Unity's terrain attributes. As a terrain we refer to the ground of an outdoor scene. Unity has a

toolkit that allows one to regulate the dimensions of the terrain, adjust changes in the height of the terrain, and apply trees, grass, wind zones and much more.

In the beginning, a terrain is nothing more than a large, flat plane as shown below.

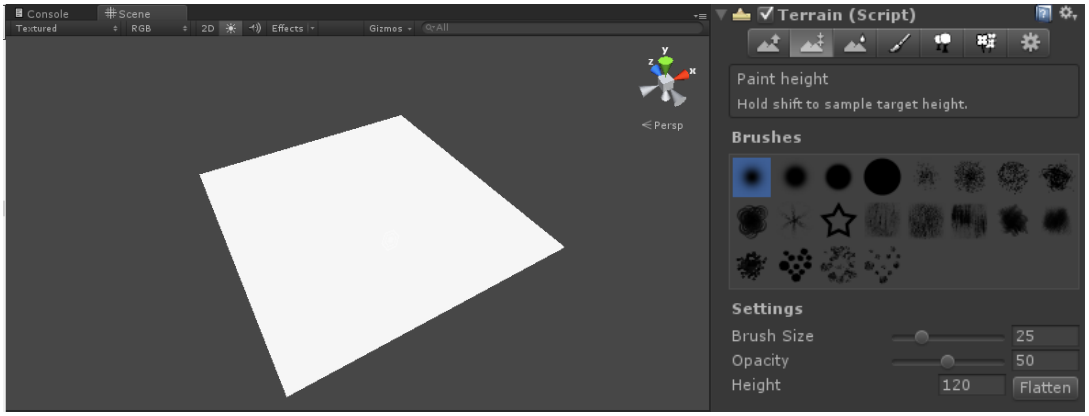


Figure 28: A New Terrain Gameobject and Unity's Terrain Toolkit

Since the airplane will fly towards one direction only, we set a very big depth to the terrain. There are tools that allow us to adjust the height levels of the terrain, allowing us to create the mountains and in general the different height areas of the valley. Using the so called "brushes" in Unity's terrain toolkit, the terrain is firstly transformed as shown below.

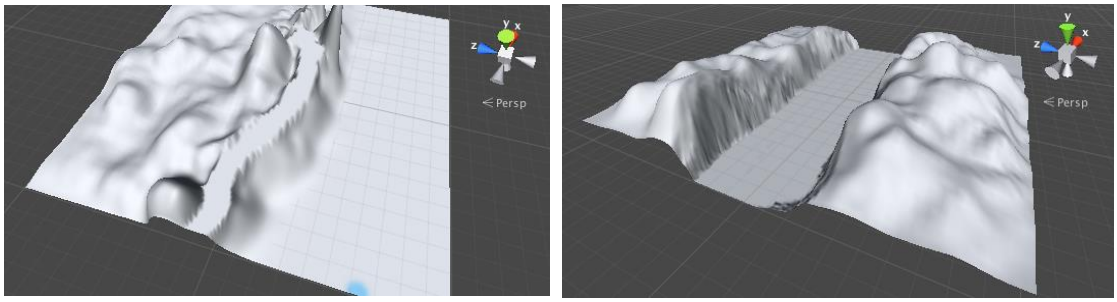


Figure 29 : Using Unity's Terrain Toolkit. The First Attempt of Creating the Environment

Having created the final model of the terrain, it's time to decorate it with some colors and in general all the textures needed. Ground textures, grass, dirt are all applied from the Unity's terrain toolkit.

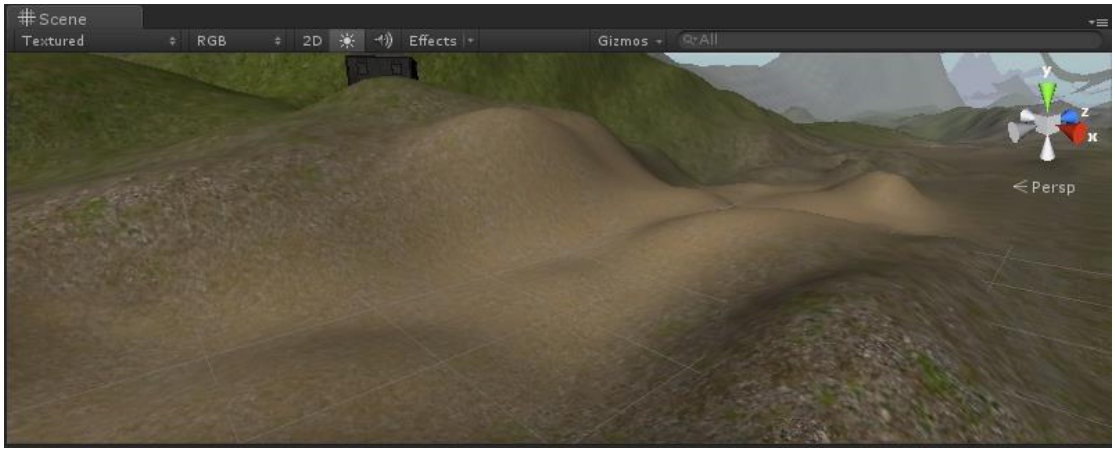


Figure 30: Applying Ground Textures to the Terrain

The next step was to import some models from online repositories. We search online for house and trees models that suited our needs and placed them on the terrain

Figure 31: House and Tree Models Imported into the Scene

Finally, pipe models, coins and the airplane that the user would navigate through the world were imported. For the pipe models, custom made materials and textures were used and assigned and a coin object was placed in between the vertical pipes so that the user could collect. The pipe models are loaded on runtime of the game because of high memory and rendering costs. After having placed the pipes, the airplane is instantiated at the beginning of the pipes row in order to be controlled and navigate through them.

Figure 32: A Random View of The Final Action Game Scene After Having Imported All the Models

Apparently, all these models were created in separate Unity project folders and scenes and organized in a way that would allow easy export and import into our final master scene.

4.2 Setting up the Virtual Scene

Having created all the models for our interface, it is about time we explained how they were imported and positioned in our master scene as well as how they interact with each other and the virtual environemnt. Since the user interacts with most of the created objects, we must explain how this interaction was achieved. In the previous sub-chapter we did not mention anything about the collision shape of the 3D objects.

In general, the collision shape of 3D objects can be automatically assigned using the mesh component collider. Each gameobject that participates in raycast hits, or in other words that the user can interact with by looking at it, have custom made colliders. According to each object's geometry, primitive type colliders are assigned. By using custom colliders rather than the mesh colliders, the colliders that wrap exactly the object, the dimensions of the colliders could change and small divergences of gaze data would still allow correct interaction.

Below, an example of the creation of a box collider is presented. The figure represents the folder model inside a green box. The green scaled primitive cube surrounding the geometry of the folder is the collider that was created for the folder object. As we can see, the collider surrounding the object does not fit exactly the gameobject it surrounds. Depending on the gameobject, each collider extends the bounds of the objet by ten or fifteen percent on each side; on the top, left, right and bottom.

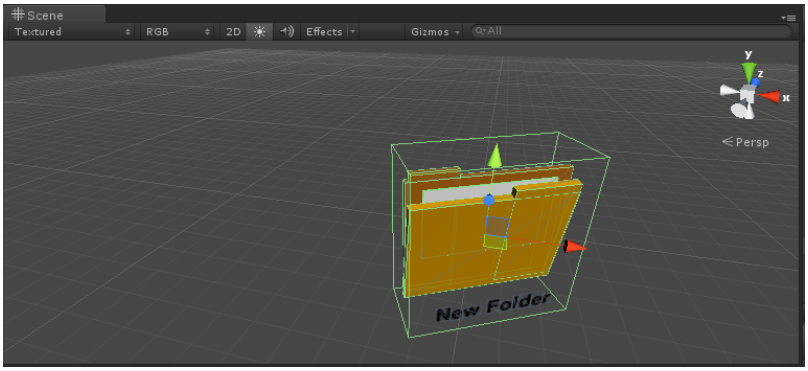


Figure 33 : The Green Box is the Folder's Collider Component

The imported models were placed inside a new master scene. First of all, the scene was divided into six cone zones respectively for each application. The scene behind each transparent dark gray layer belongs to each individual application.

The Pictures Explorer scene and the Music Player scene have the same basic layout. They are both divided into left and right panels. The left half area consists of the models that represent the directory folder that the user accesses and on the right half is the area of slide preview of the selected photos or the music player respectively.

Pictures Explorer & Music Player Scene

An example of the Pictures explorer scene is shown below.

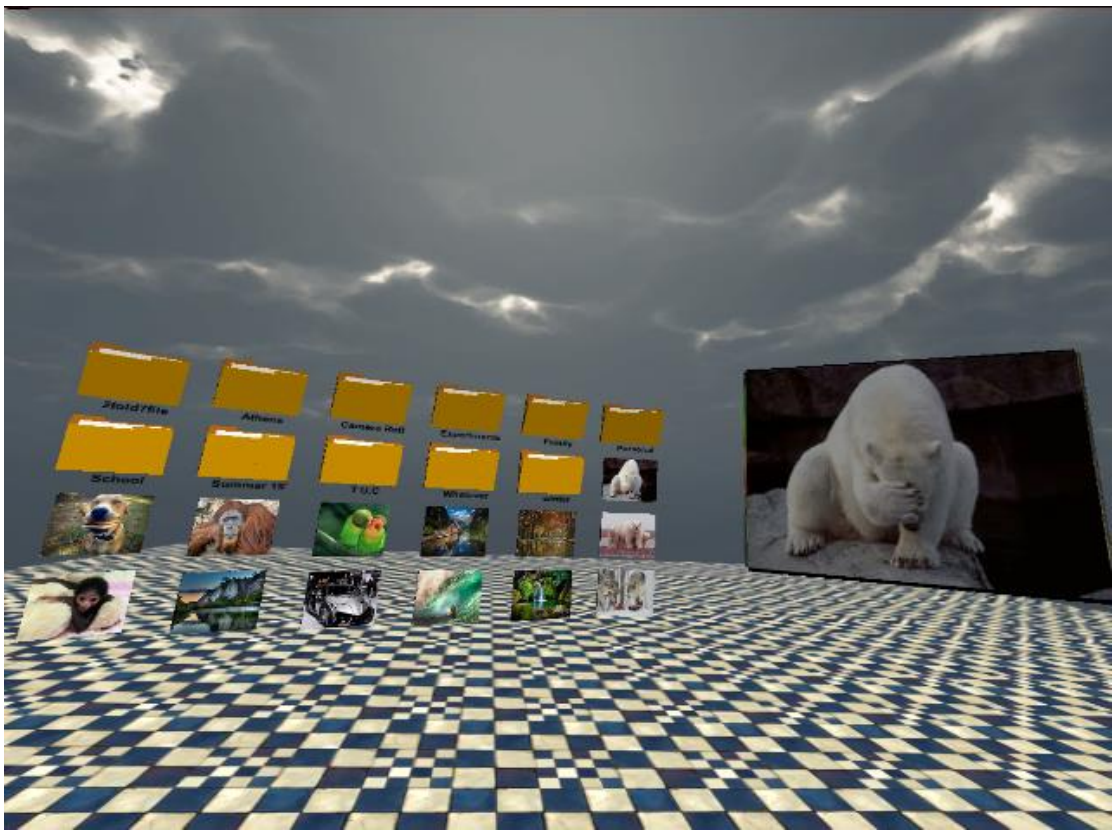


Figure 34: A View of the Pictures Explorer Scene

As we can see, on the left half of the view, there are two different types of models, the folders models and the pictures models. Depending on the number of folders and content of each current directory, more or less models are presented. This means that there is no standard visualization of the scene. The only thing that is exactly the same at every run time is the number of models that consist a row of the visualized directory. Just for reference, the maximum number of models, both folders and files, are six per row.

On the right half view of the scene, we can see the picture preview model. This object is placed in a static position that never changes but only rotates around itself on user selection. By default the textures placed on the model are the first four pictures of the current directory the user is navigating through and change every time a user clicks a picture model, changes directory or selects the preview model. By changing, we mean that the models' surface-textures change dynamically to depict the next photo-model the user selects or, if he keeps interacting with the model, respectively navigate through the entire directory. If the user navigates through the entire directory then the preview repeats itself.

The root folder is the Window's Picture Directory folder. Once the user opens an inner directory then the GoBack model appears dynamically, on a static always position, allowing him to go back to the previous directory. The Back game object disappears when he reaches the Root Directory again since there is no parent folder to access and visualize. Each model is positioned through code and depending on the folder that is depicted, changes.

Following is an example of a random view of the Music scene.

Figure 35: A View of the Music Player Scene

As for the Music Player Scene, on the left half view we can see the Windows Music directory content. The folders and subfolders are represented with the folder model and each song represented with the song model. Like in the Picture Scene, the display changes depending on the computer's directory. On the right, we can find the music player itself being accompanied with the 4 speakers.

On the other hand, the Email Composer and the Text Editor have similar layouts. On the left half of the view we can find the keyboard model and on the right the editable fields.

Email Composer & Text Editor Scene

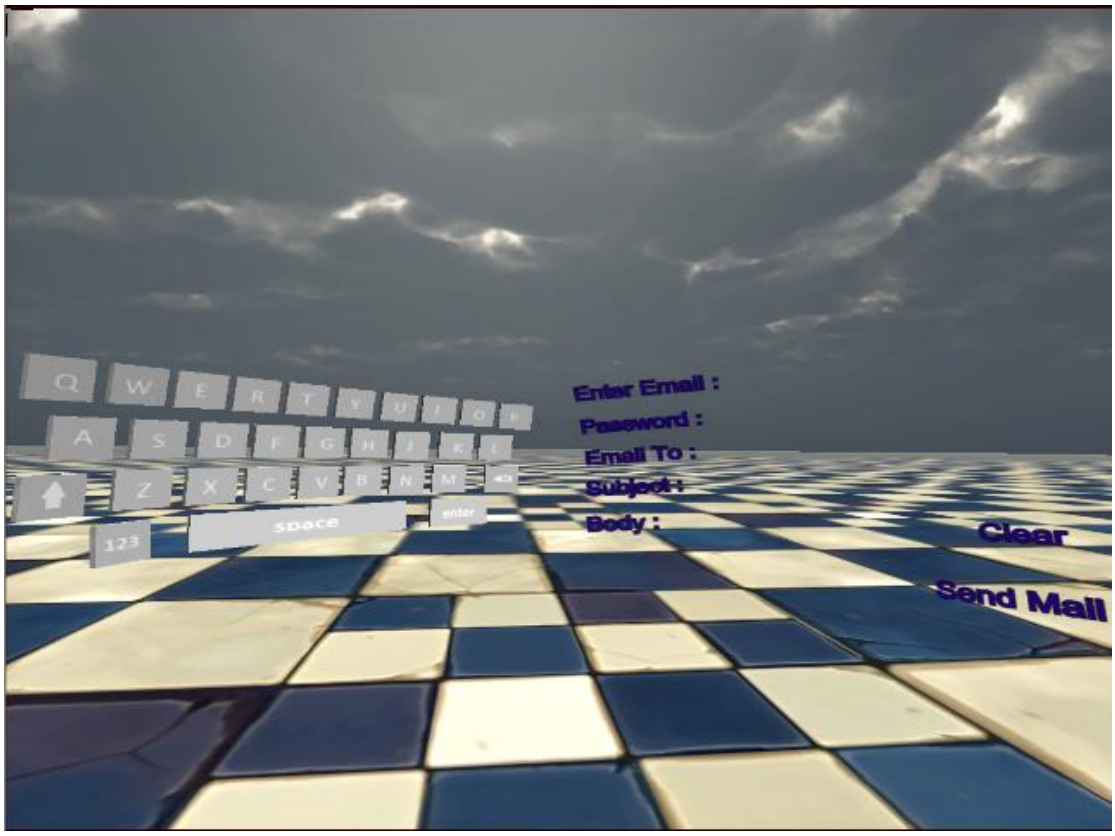


Figure 36: A View of the Email Composer Scene

As mentioned above, on the left half of the scene we can see the keyboard model. It is always placed on a static position and different layouts are loaded depending on the user's needs and interaction. By default, the character layout is displayed and when a transition to the numbers' or symbols' layout is needed, the new layout is placed at the same position of the previous.

Concerning the Email Composer scene, on the other half of the display is the 3D email form that the user is called to fill in order to compose the email. The Form is placed in such a way in order for the user to have good vision of it while typing. The form consists of 5 fields, the

sender's email, password, receiver's email, subject and body field of the mail as also a Clear and Send selection that, respectively, clears the form or sends the email

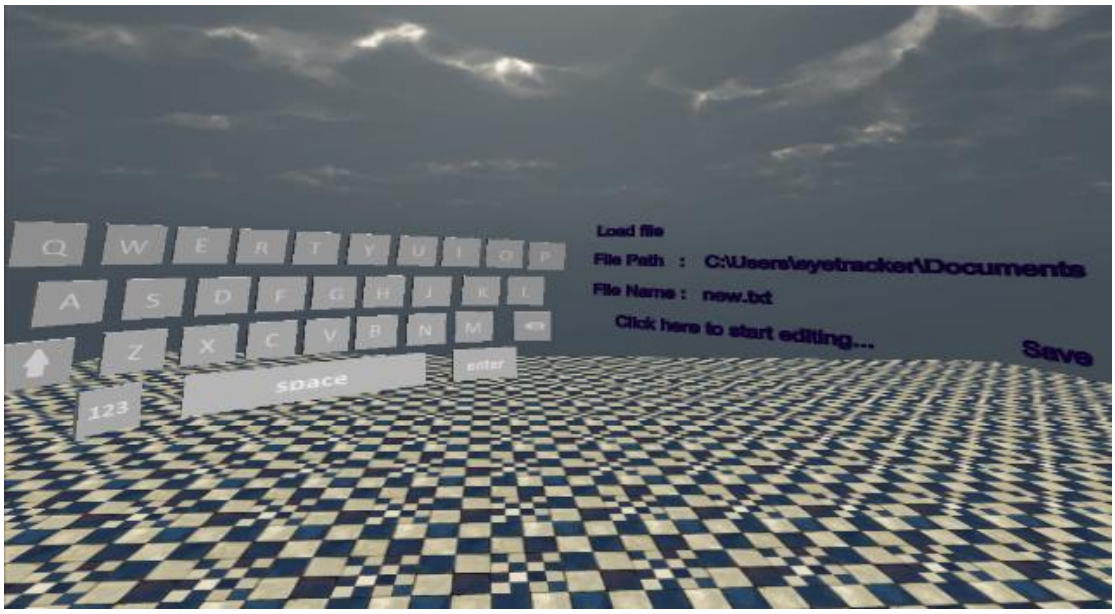


Figure 37: A View of the Text Editor Scene when the User Created a New Document File

As for the Text Editor, the right half of the scene depicts the 3D text models and the actions that are allowed. By default, the user is allowed to create a new documents file with the name "new.txt", edit its body field and save it at the default Windows Document Directory. If he selects the load file field, the root Documents directory of the computer is represented and can select the model of the desired document to be loaded. Once he selects a file, the document models are destroyed and the fields are loaded with the respective data of his selection. Below the process of editing an existing file is shown.

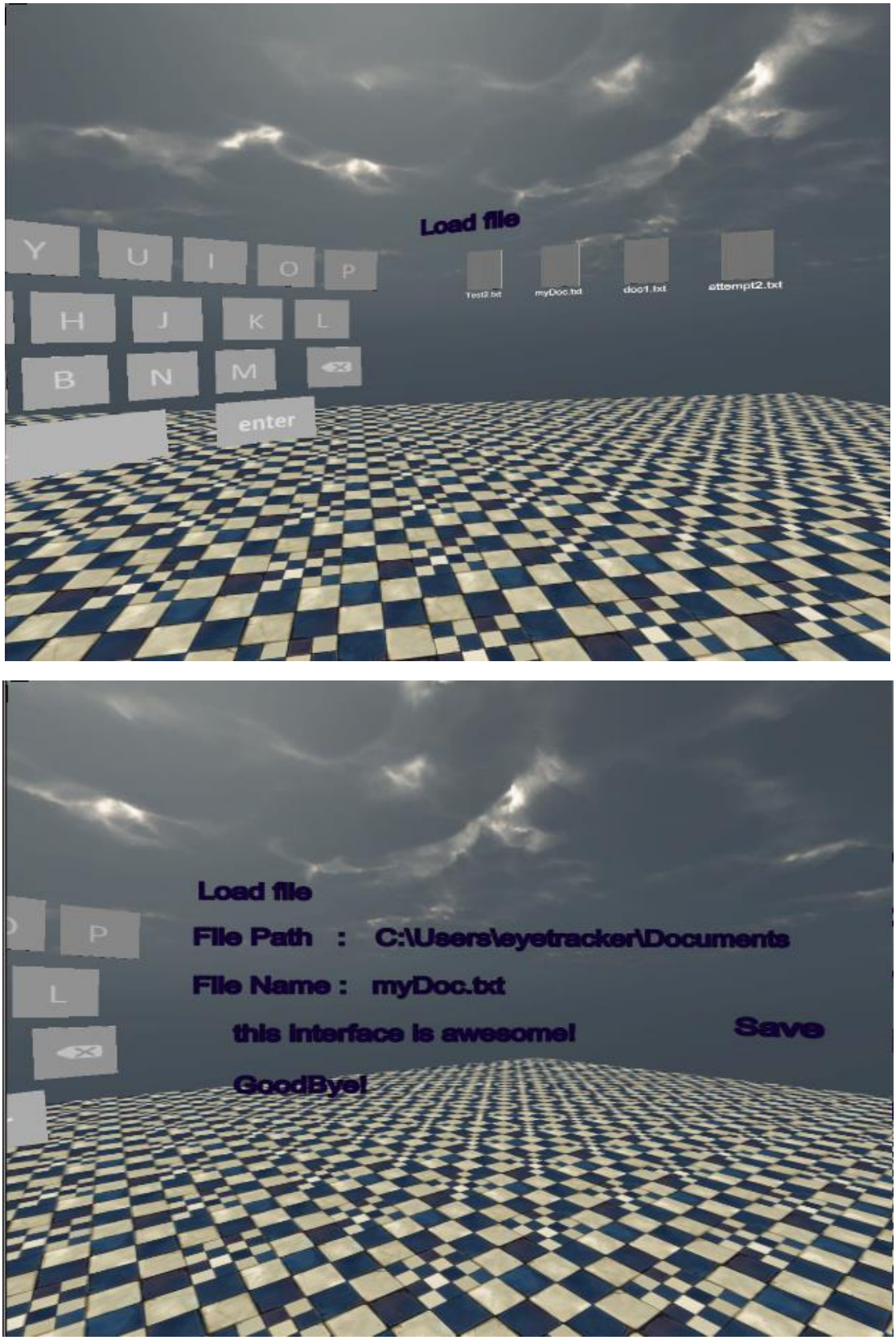


Figure 38: (a) A View of the Scene when the User Selects to Load Document File from the Windows Directory (b) The User Selects to Edit the File myDoc.txt. It's Content and Path are Displayed and the User is Allowed to Edit, Save or Discard the Changes he makes

The two Game scenes have completely individual layouts. Only the tile game will be described below. For more details on the action game please refer to the previous chapter.

Tiles Game

Figure 39: A View of the Tiles Game Scene Before the User has Chosen a Image to Start Playing With.

For the tiles game only the Windows root Pictures folder is allowed to be accessed and therefore only the pictures it contains, no folders or subfolders, will be visualized. Once loaded, each picture in the root directory is fragmented and applied as a texture, on the 5x5 block models. The folder inspector displays 5 images in each row. Once the user selects an image, then it moves to the front, while the rest are placed deeper in the background and in a fixed time they disappear. After three seconds, the tiles are randomized and then the 5x5 model is brought up and placed in a fixed position in order to guide the user to where he is allowed to place each block in order to reform the initial image. Once a tile is selected the allowed positions to place the block when hovered change to a transparent color of green.

Figure 40: A View of the Tile Game Scene While the User has Selected to play the Puzzle with the Polar Bear Image and Has Placed a Few Blocks at the Correct position

Chapter 5 – Implementation (Scripting)

In this chapter, the implementation and development of the application and its components will be described. More specifically, the way in which the 3D Virtual Scene was configured in order to achieve stereoscopy, the main classes and controllers of the application and how user's inputs were handled.

5.1 Stereoscopic View - Setup

The HMD used in this thesis uses the partial overlap method to achieve stereoscopy. However, the HMD does not automatically adjust the input image signal to match that requirement thus the computer's output image signal must be formatted based on the HMD's specifications.

The ideal solution was to use dual monitors and output two video signals from Unity3D, one for each monitor and therefore for each camera/screen of the HMD. Since the HMD requires SXGA resolution to work our solution should meet this specification. Based on the SXGA format, each camera of the HMD has resolution of 1280 x 1024 pixels, amounting to a total of 2560 x 1024 pixels for the display of the HMD screens (the resulted resolution of 1280 x 1024 pixels for each separated view was not adequate). To fix this, the viewport of the application was expanded using the graphics card's option to select two displays with total resolution of 2560 x 1024 pixels.

In order to output two video signals from Unity3D, one for the left screen (left eye) and one for the right screen (right eye) of the HMD, two cameras and two controllers were needed. We tried to implement real world dimensions as for the virtual "head format" to suit the specifications of the HMD, thus we translated the positions of each part of the head to Unity3D world units. Depicting the headset in Unity3D, the "Head" model consists of a head and two cameras. Both cameras were centered at the width of the head but positioned at the 2/3 point of the heads height (if we look at it from bottom to up) and in the 1/3 of its depth. The distance between the two cameras-eyes based on the average pupil distance measurement, is at 64mm or 0.064 Unity3D world units. Since that is the average measurement of the inter-pupillary distance (IPD), we allowed through scripting each user to manually set his own IPD in order to achieve better stereoscopic effects. By pressing the 'o' keyboard key the IPD increases and the 'p' keyboard key it decreases. We have to be careful here because by increasing the IPD the user might feel uncomfortable and dizzy [14, 16]. Thus, during the experiments we were responsible for setting the IPD. Below we present the code that set the IPD of the individual, and in simple words, the distance between the two eye-cameras.

```
void Update () {
          setIPD_ =0.01f;
      if (Input.GetKey("p")){
         Vector3 prevRight = CamRightEye.transform.position;
         Vector3 prevLeft = CamLeftEye.transform.position;
         CamRightEye.transform.position= new Vector3(prevRight.x + setIPD_, prevRight.y,
prevRight.z);
         CamLeftEye.transform.position= new Vector3(prevLeft.x setIPD_, prevLeft.y,
prevLeft.z);
         Debug.Log(CamRightEye.transform.position.x);
      }else if (Input.GetKey("o")){
         Vector3 prevRight = CamRightEye.transform.position;
         Vector3 prevLeft = CamLeftEye.transform.position;
         CamRightEye.transform.position= new Vector3(prevRight.x - setIPD_, prevRight.y,
prevRight.z);
         CamLeftEye.transform.position= new Vector3(prevLeft.x + setIPD_, prevLeft.y,
prevLeft.z);
      }
   }
```

So far, a full overlap stereoscopic effect has not been achieved yet since both eyes are still viewing the exact same video signal. The FoV needs to be adjusted. Based on the HMD's specifications, the FoV of each camera-eye was set to 90 ° and the cameras were rotated by 13 ° left and right respectively.

The figure below depicts how the cameras position, rotation and FoV are set.

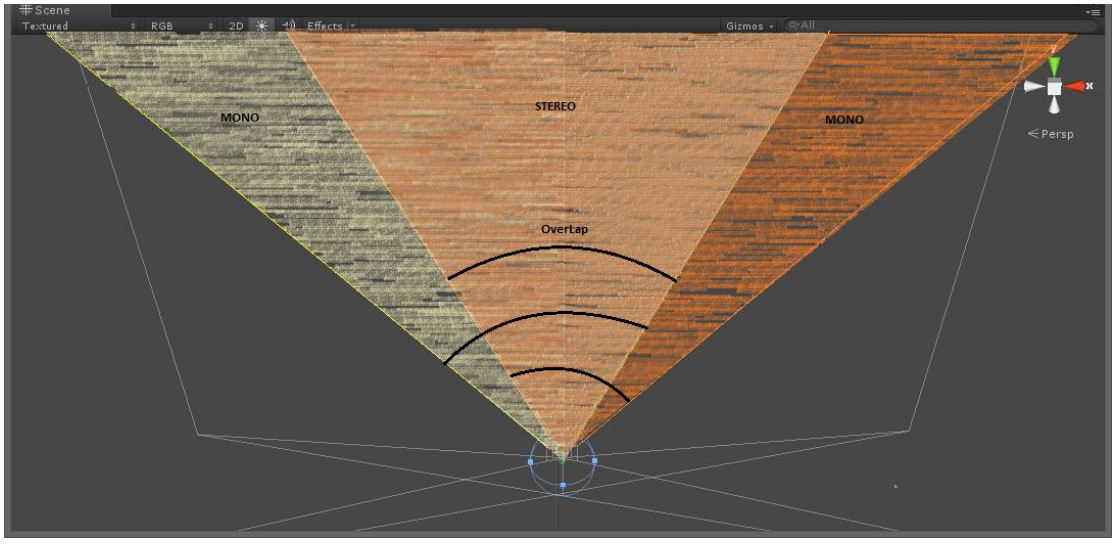


Figure 41: Stereoscopic Cameras' Field of View. The Left and Right Cameras are Rotated by 13º with a Field of View 90º each. The Field of View of the Left and Right Cameras are in a Yellowish and Orange Shade Respectively. The Stereo View of the Scene is in a Pink Shade.

Finally, in order to display the scene to the user, the eye cameras' video signals must be projected on two planes that depict the HMD screens. A plane is a flat model on which materials and textures can be applied on. In our case, we added two components on each plane. A render texture that allows a video signal to be set as a texture on a material and the material to apply the render texture on. So, to be more specific, the video signal of the left camera-eye was assigned to the left render texture and then that texture assigned as the material of the left plane. Respectively, the same steps were applied for the right plane. Both textures' size was set to 1024 x 1024 pixels. Basically, it is exactly like the technique a projector uses, but in our case the surface that the image, video signal, is projected are the two planes.

5.2 Handling User Input

Navigation through the interface is limited and specified by rotating around a main pivot. There are three different type of inputs according to the purpose of this thesis; gaze data inputs, head data inputs and a single button input. Initially, two user inputs would allow interaction with the application, eye gaze data and head orientation data. The eye gaze data would not only simulate the traditional mouse movement and move the mouse cursor respectively based on where the user looks but would also simulate the mouse click by detecting blinks.

In the very beginning, once the first out of the six applications was completed, we tried to apply the above interaction technique; controlling the whole application only with gaze data. With our preliminary approach this interaction lead to a common issue that occurs while using eye tracking to control an interface known as the Midas' touch problem. Certain eye movements are involuntary and accidental interface activation is frequent [5]. Due to the way the ViewPoint Software dealt with blinks, it was very difficult to apply settings to accurately detect an intentional blink with a duration longer than the normal autonomous blink As the eye lid comes down during a blink, the ellipsis that the ViewPoint software tries to fit to the pupil in order to "read" the gaze becomes increasingly flat before it disappears. This flattening, allows the software to detect blinks since it crosses below a set threshold. The normal semi-autonomous blinking, in other words the non-intentional blinking, can be detected almost accurately since the ellipsis is below threshold for less than 350 milliseconds, in average, and thus it "finds" the pupil almost immediately. In 99% of our cases, wanting to detect blinks that were larger or equal to one second, the minimum estimated time for both our application and more important the user feeling comfortable with keep his one eye closed in order to click on an object, was never detected.

This happened because despite the elliptical fit being flattened since it couldn't find a pupil, it would randomly detect the pupil at the user's eyelashes until the blink would be considered as a click-blink. After many tests and hours of research, we figured out that this was impossible and if it were possible, precise settings of the elliptical fit's threshold would

be necessary for each individual user. But, even if we found the exact settings for a single individual, other factors such as the infrared camera's reflections or external environmental light would lead to false blink detections. Since false blink detection would lead to random clicks on objects and therefore non-intended interactions with the interface we decided to comment the code out of the project and instead replace it with a single button input or more specific the "space" key of the computer's keyboard. However, if a different eye tracker or HMD is used it is possible that the disabled code may work with minor changes in the blink detection algorithm.

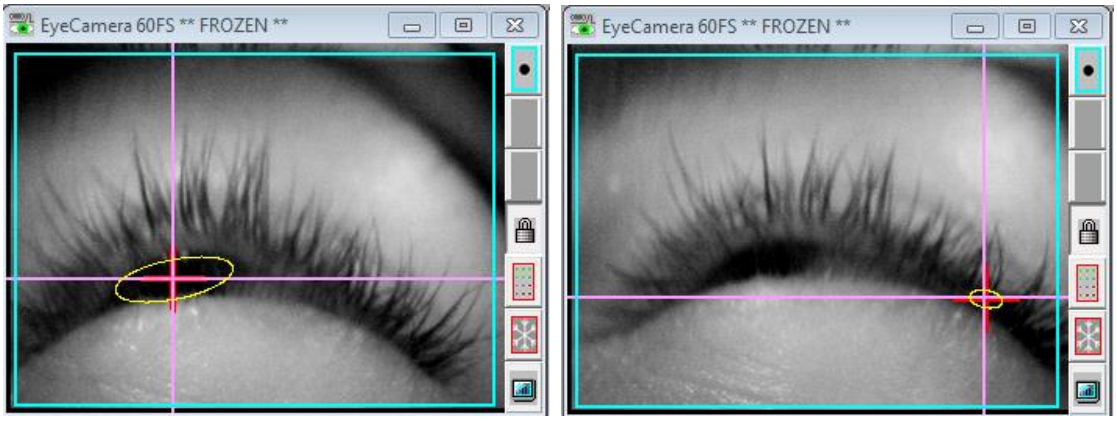


Figure 42: ViewPoint Software Detecting a Blink Once the Eye Lid Closes (Left) but unfortunately it Almost Immediately identifies the eye lashes as a pupil. (Right)

The simplest of both is head trackers data input. While immersed in the interface, the user may rotate around the scene using his head and by sitting on a swivel chair to be more comfortable. The rotation is in both horizontal (yaw) axis and vertical (pitch) is 360°. The user input was captured through the head-tracking device IntertiaCube3.

Both input types will be explained in detail in the following chapters.

5.3 Communication with the Eye - Tracking Device (.dll)

As mentioned in previous chapters, the data from the infrared cameras of the Eye-tracking device is done through the SDK that ViewPoint EyeTracker® of the Arrington Research Company provides for third parties applications. More specific, the SDK comes with a dynamic link library named VPX_InterApp.dll which allows any third party application to interact with the eye –tracking device. The dynamic linked library must be saved in the same folder as the Unity executable file in order to be accessed by the application.

In order to interact with the eye-tracking device, our application needed to register with the dynamic link library. After the registration, the application obtains a unique message identifier used by ViewPoint for inter-process communication. Since the source code of the

dynamic linked library is already precompiled, and written in C++, we had to create a new class MyVPX in C# that will bind the library with our application.

C# allows to make calls to native code from managed applications, in our case Unity3D, through the DLLImport attribute. The DLLImport attribute tells the compiler to declare a function residing in the VPX_InterAPP.dll. So, in our case the code below "reads" the needed functions from the dynamic linked library. Below are the functions in the library that are called inside Unity3D.

```
[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_GetGazePoint(VPX_RealPoint *gp );

[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_GetGazePoint2(int eye,
VPX_ReaPoint *gp );

[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_GetGazePointSmoothed2(int eye,
VPX_RealPoint *gp );

[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_GetDataQuality2(int eyn, int
*quality);

[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_InsertCallback (Func <int, int,
int, int, int> callbackfunc);

[DllImport(vpx_dllPath)]
unsafe public static extern int VPX_RemoveCallback (Func< int, int,
int, int, int> callbackfunc);

[DllImport(vpx_dllPath)]
public static extern VPX_EyeEventType VPX_GetBlinkEvent2(int eyn );
```

As we can see above, there is an unsafe command in front of each function declaration. C# allows using pointer variables in a function code block when it is marked by the unsafe modifier. The unsafe code or the unmanaged code is a code block that uses a pointer variable. However, unsafe code cannot be compiled by Unity 3D. This was solved by adding two files in our Unity3D project folder, a smcs.rsp file and a gmcs.rsp, containing only the command shown below.

```
-unsafe
```

Eventually after the successful registration with the Eye-tracking software, myVPX class defines a callback function which is managed by the library VPX_InterApp.dll. Unfortunately, inside the callback functions limited coding can be used. Since it interacts with native code written in other programming language, C++, only common types such as integers or characters can be defines and used. Otherwise the application will crash. For example, c# tolerates strings in a different way C++ does, causing inter-process communication to fail

since these languages understand string data types in different ways. Below is the function that establishes Unity and ViewPoint inter-process communication.

```
VPX_InsertCallback (theCallBackFunction2);
```

The *theCallBackFunction* function is responsible for the data exchange between the library and the UI's executable file and will be described later on. When a connection is established, we can see from the ViewPoint software that in the Status Window the attribute of DLL Sharing has been changed by 1; this shows the number of third party applications that are registered.

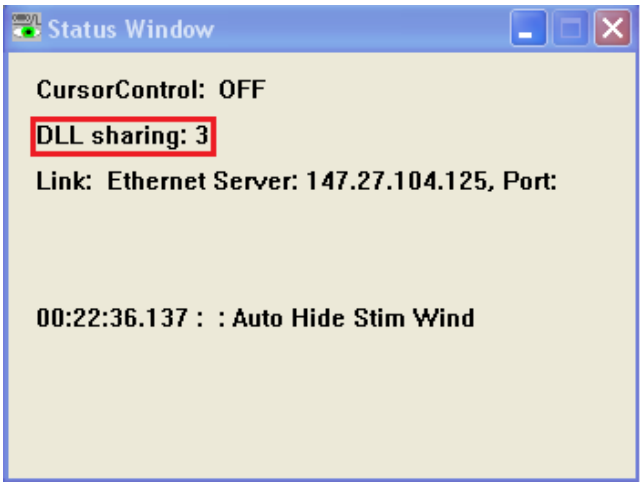


Figure 434: The Status Window of the Viewpoint Software Shows the Total Number of Registrations from Third Party Applications

Every new "fresh" data arrives in the ViewPoint application from the frame grabber and the application sends it to all the other programs that are registered. Then the library VPX_InterApp.dll calls every function that was defined as callback for each application, passing the "fresh" data. As "fresh" data we mean the message which can inform the application that new data is available about a change on eye fixations or an infrared camera, and in general useful data about the eye. Every time "fresh" data is available quality checks are made, using the VPX_GetDataQuality2 () function of the VPX_InterApp.dll and depending on the resulted quality code gaze data are fetched using the VPX_GetGazePoint2 () function.

Quality Code	**Information**
5	Pupil scan threshold failed.
4	Pupil could not be fit with an ellipse.
3	Pupil was bad because it exceeded criteria limits.
2	Wanted glint, but it was bad, using the good pupil.
1	Wanted only the pupil and got a good one.
0	Glint and pupil are good.

Table 1: Quality Codes and the Respectively Information about the New Data Received

Depending on quality code, data is either fetched or discarded. Quality codes “0”, “1” and “2” are considered as good data and fetched. On the other hand codes “3”,”4”,”5” are discarded. Possible reasons for these errors are either the user blinked or an impropriate eye movement was made or the infrared camera was instantly shaken due to sharp spin of the HMD.

```
unsafe int theCallBackFunction2(int msg, int subMsg, int param1, int
param2){
      VPX_RealPoint gp_0;
      //VPX_EyeEventType blnk;
      int qualityData;
      //Debug.Log ("Quality Data "+qualityData);
      //VPX_GetDataQuality2(1,&qualityData);
      //if (qualityData == 3 && VPX_GeBlinkEvent2(1) ==
      //VPX_EyeEventType.VPX_EVENT_Blink_Continue) {
            //Debug.Log("blinkingA...");
            //isBlinking=1;
            //endblink=0;
      //}

      VPX_GetDataQuality2(0,&qualityData);
      if (qualityData == 1 || qualityData == 0||qualityData == 2)
      {//pupil(1) or Pupil & glint(0) are good.GET data
            //VPX_GetGazePoint2(0,&gp_0);
            VPX_GetGazePointSmoothed2 (0, &gp_0);
            //VPX_GetBlinkEvent2 (0);
            //isBlinking=0;
            trackerData.x = gp_0.x;
            trackerData.y = gp_0.y;
            //endblink=1;
      }
      return 0;

}
```

In the end, when we exit the application, we must end the binding that was made with the dynamic linked library of the ViewPoint software. This action is mandatory because every time we would close our application, it would crash because ViewPoint would keep on sending data, or at least try, to a receiver that would not exist anymore.

```
void OnApplicationQuit(){
      int removed=VPX_RemoveCallback (theCallBackFunction2);
      Debug.Log ("exit  "+removed);
}
```

5.4 Communication with the Head - Tracking Device (.dll)

In a similar way, data is passed from the head tracking device through the InterCube3 software's SDK using a link with trackers dynamic linked library, isense.dll; the DLL must be saved inside the System32 Windows directory. Two are the main classes that are responsible for the communication between the two applications, StereoCamLook and HeadTracker scripts. Below we will explain how connection is established and how data is retrieved from the device. *TheHeadTracker* class is responsible to establish the communication between the two applications and retrieving the data from the tracker and the StereoCamLook class is responsible for passing the data as the Euler angles of the "head" model in our application.

The dynamic linked library is accessed with C#'s DLLImport attribute, exactly as described previously about communication with the eye-tracking software. Below, the function declaration is shown:

```
public const string isense_dllPath = "C:\\Windows\\Sytem32\\
isense.dll";

[DllImport(isense_dllPath)]
unsafe public static extern int ISD_OpenTracker(
      IntPtr hParent,
      int commPort,
      bool infoScreen,
      bool verbose
   );

[DllImport(isense_dllPath)]
unsafe public static extern bool ISD_CloseTracker(int handle);

[DllImport(isense_dllPath)]
unsafe public static extern float ISD_GetTime();

[DllImport(isense_dllPath)]
unsafe public static extern bool ISD_GetTrackerConfig(
      int handle,
      ref ISD_TRACKER_INFO_TYPE Tracker,
      bool verbose
   );

[DllImport(isense_dllPath)]
unsafe public static extern int ISD_GetTrackingData(
      int handle,
      ref ISD_TRACKING_DATA_TYPE Data
   );

[DllImport("isense.dll")]
public static extern bool ISD_GetStationConfig(
      int handle,
      ref ISD_STATION_INFO_TYPE Station,
      int stationID,
      bool verbose
   );

[DllImport("isense.dll")]
public static extern bool ISD_ResetHeading(
      int handle,
      int stationID
   );
```

Using the *ISD_OpenTracker* function, the application seeks for the tracker that is connected to the computer and if a tracker is detected then a timer is set to count the connection duration and with *ISD_ResetHeading* synchronization to the tracker is done. After successful connection, head tracking data are passed from the device to our application by calling the *ISD_GetTraackingData* function once per frame. Once the user decides to exit the application, *ISD_CloseTracker* is called to terminate smoothly the connection without crashing.

The data are passed from the one class to the other using the PlayerPrefs attribute that is allowed in Unity3D:

```
PlayerPrefs.SetFloat ("StereoEuler_y",data.Euler[0]);
PlayerPrefs.SetFloat ("StereoEuler_x",-data.Euler[1]);
PlayerPrefs.SetFloat ("StereoEuler_z",-data.Euler[2]);
```

Respectively, the yaw, pitch and roll orientation data that are received from the tracker, are then used by the *StereoCamLook* Class as inputs in order to adjust the head models' orientation, using the PlayerPrefs' Get method.

```
float eulerX, eulerY, eulerZ;

eulerX = PlayerPrefs.GetFloat ("StereoEuler_x");
eulerY = PlayerPrefs.GetFloat ("StereoEuler_y");
eulerZ = PlayerPrefs.GetFloat ("StereoEuler_z");

myHead.transform.eulerAngles = new Vector3 (eulerX, eulerY,eulerZ);
```

In the end, when we exit the application, the *ISD_CloseTracker* function is responsible to terminate the connection.

```
void OnApplicationQuit(){
      ISD_CloseTracker (handle);
}
```

5.6 Deprojection of eye's gaze data

As projection we refer to the transformation from the 3D world coordinate system to the 2D screen coordinates. On the other hand, deprojection refers to the transformation of a 2D screen coordinate into a 3D world space origin and direction. In our situation we are interested in deprojection since they eye gaze input should be used to detect objects in the scene. The process of deprojection will be explained below in detail with some basic source code samples.

The data received form the ViewPoint application is raw data of the user's gaze in 2D coordinates normalized between values zero and one, in both x and y axis. Since each point in a 3D scene is described with a three-dimensional coordinate system with origin and direction, this data is not useful as it cannot be used to determine which object in the 3D world space a participant is looking at. Therefore, this screen point must be transformed in a Vector3, with origin and direction, components of a ray. The ray or even better RayCast, known as the procedure of casting a ray, is casted from a specific origin and has a direction, therefore has meaning in a 3D world, and can determine what objects in the world the user is aiming at with the mouse, and in our situation with by gaze data.

The eye's gaze data from the ViewPoint software are normalized between zero and one. Thus, these 2D coordinates must be converted into 2D screen coordinates. Since the total resolution, as mentioned in previous chapters, of our screen is 2560 x 1024 pixels every gaze data value has to be multiplied and scaled based on this custom resolution. Also, since our initial thought was to use the right eye only as a blinking input, in order to apply clicks, only the left eye's data are used and therefore, only the left half of the screen, 0 to 1280 pixels, horizontally will be deprojected into the scene.

```
function translateToPixels(){

        if (gaze_x>=0 && gaze_y>=0){
            cross_x = gaze_x * 1280.0;
            cross_y = (1.0-gaze_y)* 1024.0;
        }
}
```

On the above cited source code, *gaze_x and gaze_y* are the data that are received from the eye tracking device. We only accept positive values since negative values of gaze data are data with noise and therefore should be discarded. Also, the screen coordinate system values on the vertical axis increase from bottom to the top however the gaze data vertical values increase from top to the bottom thus we have to inverse the values before the data are scaled based on the resolution.

Having transformed the 2D gaze data into 2D screen coordinates the next step is to cast a ray with origin and direction into the scene. The ray must be casted in order to determine what the user is looking in the 3D world. The ray is casted every frame having as the origin the left eye-camera and direction of the z axis of the 3D scene and an infinite depth. If the ray cast doesn't hit anything then no action is taken but if it hits a gameobject then we have all the information of that gameobject; the gameobject must have a collider component, else the ray doesn't detect it. In our project, the hit information that we consider is the gameobject's name or tag; as a tag we refer to number of objects that are labeled as a group. A respective sample code is the following:

```
var hitInfo : RaycastHit = new RaycastHit();
var ray : Ray ;
Debug.DrawRay(ray.origin, ray.direction * 10000, Color.red);
      if (Physics.SphereCast(ray.origin,0.1f , ray.direction,
hitInfo,Mathf.Infinity)){
            if (hitInfo.transform.tag == "something"){
                  if (Input.GetKeyUp(KeyCode.Space)){

                        /*
                         * Do something
                         */
                        }
            }
}
```

Depending on the phase that the application is during a raycast hit, the main controller class of the interface decides what actions, initializations, scripts or models must be loaded.

5.7 Logging of Events

The application was recording every event that was happening during the runtime of the application. As an event was considered every interaction or selection using the users gaze and the single button as inputs. Each log entry reported the specific gameobject that the user was looking at and 'clicked' on along with the exact time it happened. The time was measured in seconds after the start of each experiment and was reset every time the application was run. Also, data such as the gaze coordinate and screen deprojection of the eye data were held but proved meaningless for the statistical analysis that will be explained on chapter 6. All this data was organized in a database using the SQLite Database Engine.

Chapter 6 – User Study

6.1 Material and Methods

This user study was designed to evaluate the proposed eye-tracked interaction and the 3DUI paradigm. We aim to explore the effect of eye-tracking and in what rate users actually prefer the traditional methods in order to interact with a 3D MUI. The experimental methodology and procedure will be descried in detail in the following sections.

6.1.1 Apparatus

A total of 7 post-graduates from Technical University of Crete, 5 males and 2 females with mean age 24.3, participated in the experiment. All participants had normal or corrected to normal vision and no reported neuromotor or stereovision impairment. The test VE was set up in a dedicated experimental space on campus, which was darkened to remove any periphery disturbance during the exposure.

The 3D MUI was displayed on an NVisor ™ SX111 HMD, having a stereo SXGA resolution and a FoV of 102 degrees horizontal by 64 degrees vertical. Participants panned around the VE using an InterSense ™ InertiaCube3TM 3DoF head tracker attached to the HMD. Eye-tracking data was recorded using a twin-CCD binocular eye-tracker by Arrington Research ™, also attached to the HMD updating at a frequency of 30Hz.

6.1.2 Visual Content and Experimental Procedure

We conducted a pilot study in order to identify which application required the greatest gaze-tracking accuracy. The study indicated that the Mail Composer was the most suitable application to assess task accuracy requiring complex eye movements such as typing on a virtual keyboard.

The experiment was set to be completed in three stages by each participant. The first stage was a training session. During this stage, the participants were able to familiarize themselves with the device and understand how gaze control works and appropriate adjustments were made individually for each one. The second stage, which is the main phase of the experiment, the participant was asked to perform a specific task and data was recorded. The third and final stage took effect when participants finish with the task and were asked to complete a questionnaire about their experience. Below, the three stages of the experiment will be explained in detail.

Phase 1

During the first stage, appropriate adjustments were conducted accordingly for each individual. Initially, the participant was asked to fill in his personal information in order to save in the database and the procedure was explained. Ensuring that they have understood the procedure, the next step was to wear the HMD and adjust it in order for them too feel comfortable and achieve the best stereoscopic view. This was achieved by moving slightly the position of the HMD on his head and by matching his IPD. After feeling comfortable and having a good sense of the stereoscopic view without side-effects we set the infrared cameras of the HMD in a static position in order to start the calibration of the eye's gaze. Participants were instructed how to conduct the calibration process described in Chapter 3.2.3. For two subjects, the calibration process was conducted more than once. Having everything set up, the training VE was loaded and the users freely navigated through it as much time as they wanted.

Phase 2

After ensuring that the participant felt comfortable and seeing that he could actually control the interface through his gaze, the main scene of the experiment was loaded. At this point, we remind the basic task that he has to complete; the task was divided into two parts. Firstly, Each user was asked to compose a dictated email by moving their eyes to select letters on the virtual keyboard and by pressing a switch, the "space" key from a keyboard was used, he could select the letter or symbol we wished to write. Then, the same email was written using the traditional keyboard while still being immersed in the scene; thus the keyboard was occluded by the HMD and the participant had no vision of it. Each gaze input or keyboard press respectively, accuracy and task completion times were inserted in the database for later on analysis. All but one participants (that was excluded from the analysis due to insufficient eye tracking data) successfully executed both tasks by wearing the eye-tracking HMD.

Phase 3

Following the experiments, participants were asked to fill in a questionnaire about their experience. They were asked to rate their experience when using the 3D MUI on a 1-7 Likert Scale, by answering four questions, commonly used in qualitative assessments of 3D UIs [13]:

(i) The 3D UI is as comfortable as the traditional mouse-keyboard paradigm.
(ii) I felt more tired when using the 3D UI than with the traditional mouse-keyboard paradigm.
(iii) The 3D UI was more interesting when using it compared to traditional mouse-keyboard paradigm.
(iv) I prefer the 3D UI more than the traditional mouse-keyboard paradigm.

6.1.3 Simulator Sickness

A potential side effect of all HMDs is simulator sickness. As simulator sickness we refer to fatigue, headaches, dizziness, visual discomfort and nausea that appear while someone is immersed in a VE. Also, some other side effects that were met during the experiments was eyestrain and changes in balance. Usually, these effects are caused due to system latency [14, 15], limitations to Field of View [14], to image scale factor [15], etc. In our situation, the hugest factor that causes most of the side effects is the IPD. Improper adjustments of the IPD perceives dissimilar imagery from their eyes that causes dizziness and fatigue.

Nevertheless, the experiments reported here were conducted without any participant interrupting the procedure because of simulator sickness. Only fatigue, as expected, was an issue since all the users never had past experience with eye-tracking and gaze control systems.

6.2 Results and Discussion

An independent-samples t-test was conducted to compare type rate with the eye-tracker versus occluded keyboard input. There was a significant difference in the scores for the eye-tracker (M=257.6s, SD=5570.3) and keyboard (M=123.3s, SD=7169.0) conditions; t(6) = 2:91, p < 0:05. These results suggest that participant's type about 2 times slower on average with the eye-tracker interface.

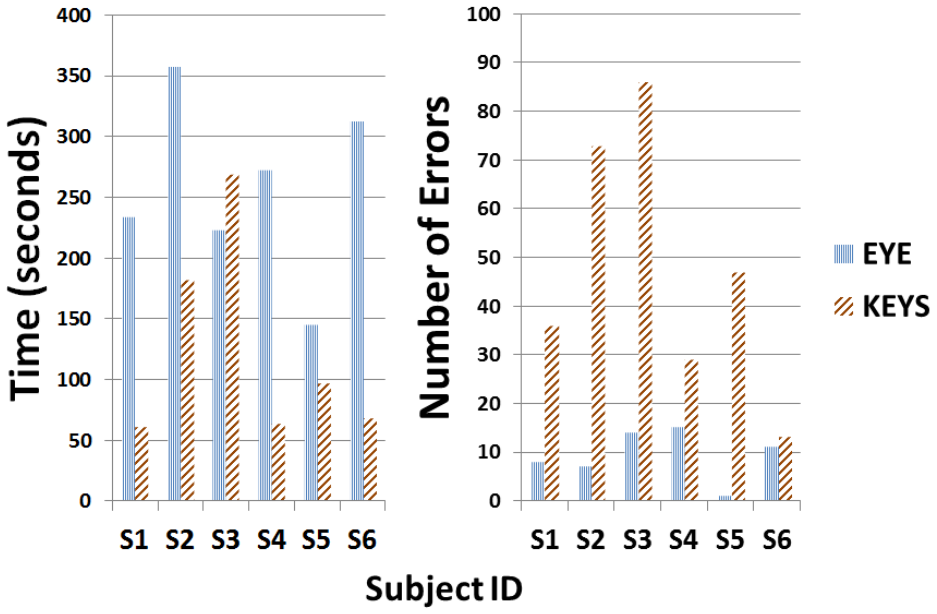


Table 2: (Left) Comparison of Task Completion Times for the Eye Tracker Input Versus Blind Keyboard Input for all Subjects. (Right) Comparison of Type Errors for the Eye Tracker Input Versus Blind Keyboard Input for all Subjects

However, analyzing the type error rate for both interfaces (9: 3% for the eye-tracker vs. 54: 19% for the occluded keyboard) indicated that despite the fact that users type faster when utilizing the out-of-view keyboard, they make many more errors. The results of the qualitative analysis and thorough discussion with the test subjects indicated that the proposed interaction method is much more enjoyable than a standard keyboard. Fatigue levels where found to be the same for both interfaces, with the exception of one subject. Usage of a 3D UI keyboard does not require good typing skills. After interaction with the rest of the applications, participants stated that they would certainly opt for the 3D UI since it feels more futuristic and therefore more exciting for the users.

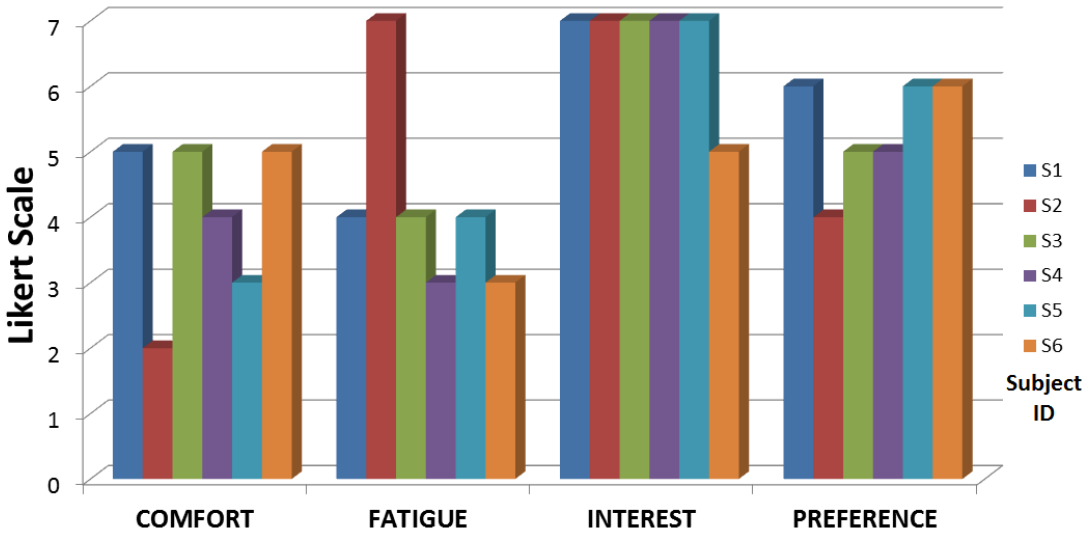


Table 3: Responses on the Four Questions of the Qualitative Questionnaire for the 3D UI Keyboard.

Chapter 7 – Conclusions

The purpose of this thesis was to employ eye tracking technology as a new interaction technique in VR and Immersive applications. Eye movements seem to be an ideal means to be used for interface control while immersed in a virtual environment. Moving the eyes over long distances on the screen required less effort when compared to other control methods [5]. We have showed that the user can move a cursor by simply gazing at the intended location. However, many eye movements are involuntary and in combination with the technology being used to detect gaze changes may lead to a certain ‘’Midas Touch’’ [5] problem. Such problems can be dealt with by combining eye movements with other input devices; a single button input or even speech commands.

Our contention is that eye-movement tracking represents a promising technique that can offer useful advantages for user interfaces in VR. A first intuitive application for such a method is interfacing with a computer for disabled users not able to move their hands to control a mouse or keyboard. Eye tracking techniques are expected to become more popular as this technology becomes increasingly more affordable and easier to use. The future seems promising for eye tracking and HCI.

Chapter 8 – Reference & Bibliography

[1] D. A. Bowman, J. Chen, C. A. Wingrave, J. F. Lucas, A. Ray, N. F.Polys, Q. Li, Y. Haciahmetoglu, J.-S. Kim, S. Kim, et al. New directions in 3d user interfaces. IJVR, 5(2):3–14, 2006.

[2] D. A. Bowman, E. Kruijff, J. J. LaViola Jr, and I. Poupyrev. 3D user interfaces: theory and practice. Addison-Wesley, 2004.

[3] T. F. B. game. Flappy bird, Feb. 2015.

[4] T. E. Hutchinson, K. P. White Jr, W. N. Martin, K. C. Reichert, and L. A. Frey. Human-computer interaction using eye-gaze input. Systems, Man and Cybernetics, IEEE Transactions on, 19(6):1527–1534, 1989.

[5] R. J. Jacob and K. S. Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. Mind, Oxford Journal, 2(3):4, 2003.

[7] D. Mardanbegi and D. W. Hansen. Mobile gaze-based screen interaction in 3d environments. In Proceedings of the 1st conference on novel gaze-controlled applications, page 2. ACM, 2011.

[8] Oculus. Oculus developer guide, Feb. 2015.

[9] Eye Tracking in HCI and Usability Research Alex Poole (Lancaster University, UK) and Linden J. Ball (Lancaster University, UK)

[10] SMI. Sensomotoric instruments announces eye-tracking upgrade for oculus rift, Feb. 2015.

[11] A. Sutcliffe. Multimedia user interface design. Human-Computer Interaction: Design Issues, Solutions, and Applications, page 65, 2009.

[12] M. Yamada and T. Fukuda. Eye word processor (ewp) and peripheral controller for the als patient. IEEE Proceedings on Physical Science, Measurement and Instrumentation, Management and Education-Reviews, 134(4):328–330, 1987.

[13] B. Yoo, J.-J. Han, C. Choi, K. Yi, S. Suh, D. Park, and C. Kim. 3d user interface combining gaze and hand gestures for large-scale display. In CHI'10 Extended Abstracts on Human Factors in Computing Systems, pages 3709–3714. ACM, 2010.

[14] DiZio, P. & Lackner, J.R. (1997). Circumventing side effects of immersive virtual environments. In M.J. Smith, G. Salvendy, & R.J. Koubek (Eds.), Advances in Human Factors/Ergonomics. Vol. 21: Design of Computing Systems (pp. 893-897). Amsterdam: Elsevier.

[15] Cobb, S., Nichols, S., Ramsey, A., & Wilson, J. (1999). Virtual reality-induced symptoms and effects (VRISE). Presence: Teleoperators & Virtual Environments, 8, 169-186.

[16] Draper, M. H., Viirre, E. S., Furness, T. A. & Gawron, V. J. (2001). Effects of image scale and system time delay on simulator sickness within head-coupled virtual environments. Human Factors, 43(1), 129-146.

[17] Yarbus, A. L., Haigh, B., and Rigss, L. A. (1967). Eye movements and vision, volume 2. Plenum press New York.

Web Links:

(Unity3D)

http://unity3d.com/

(Eye Tracking)

http://en.wikipedia.org/wiki/Eye_tracking/

(Model Repositories)

https://www.assetstore.unity3d.com/en/#!/home

http://tf3dm.com/3d-models/unity

http://www.turbosquid.com/

(ViewPoint EyeTracker ® by Arrington Research, Inc)

http://ArringtonResearch.com/

(Eye Movements)

http://www.dartmouth.edu/~rswenson/NeuroSci/chapter_8D.html

https://en.wikipedia.org/wiki/Eye_movement

https:// lecerveau.mcgill.ca/flash/capsules/articles_pdf/type_eye_movements.pdf

Appendix A: Publication

Binocular Eye-Tracking for the Control of a 3D Immersive Multimedia User Interface

Nikolaos Sidorakis* George Alex Koulieris† Katerina Mania‡

Technical University of Crete
Chania, Greece

ABSTRACT

In this paper, we present an innovative approach to design a gaze-controlled Multimedia User Interface for modern, immersive headsets. The wide-spread availability of consumer grade Virtual Reality Head Mounted Displays such as the Oculus Rift™ transformed VR to a commodity available for everyday use. However, Virtual Environments require new paradigms of User Interfaces, since standard 2D interfaces are designed to be viewed from a static vantage point only, e.g. the computer screen. Additionally, traditional input methods such as the keyboard and mouse are hard to manipulate when the user wears a Head Mounted Display. We present a 3D Multimedia User Interface based on eye-tracking and develop six applications which cover commonly operated actions of everyday computing such as mail composing and multimedia viewing. We perform a user study to evaluate our system by acquiring both quantitative and qualitative data. The study indicated that users make less type errors while operating the eye-controlled interface compared to using the standard keyboard during immersive viewing. Subjects stated that they enjoyed the eye-tracking 3D interface more than the keyboard/mouse combination.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality;

1 INTRODUCTION

Virtual Reality (VR) is soon to become ubiquitous. The wide-spread availability of consumer grade VR Head Mounted Displays (HMDs) such as the Oculus Rift™ transformed VR to a commodity available for everyday use. VR applications are now abundantly designed for recreation, work and communication. However, interacting with VR setups requires new paradigms of User Interfaces (UIs), since traditional 2D UIs are designed to be viewed from a static vantage point only, e.g. the computer screen [2]. Adding to this, traditional input methods such as the keyboard and mouse are hard to manipulate when the user wears a HMD. Using a keyboard and a mouse while immersed in a VR HMD is an erroneous extension of the desktop paradigm to VR, constituting a fundamental challenge that needs to be addressed [2].

Recently, various companies (e.g. SensoMotoric Instruments™, [10]) announced an eye-tracking add-on to the Oculus Rift Development Kit 2 (DK2) HMD. Novel, immersive 3D UI paradigms embedded in a VR setup that is controlled via eye-tracking can now be designed, implemented and evaluated. Gaze-based interaction is intuitive and natural, providing a completely immersive experience to the users. Tasks can be performed directly into the 3D spatial context without having to search for an out-of-view keyboard/mouse. Furthermore, people with physical disabilities, already depending on technology for recreation and basic communication, can now benefit even more from VR.

In this paper, we present an innovative approach to design a gaze-controlled Multimedia User Interface (MUI) [11] for a modern eye-tracking capable HMD. User fixations control the MUI. An on-screen cursor's orientation and position is directly manipulated by the gaze data. New types of immersive applications can be developed by employing this interaction paradigm. In our prototype implementation, we have developed six applications which cover commonly operated actions of everyday computing such as mail composing, photo viewing, music playing and gaming. Our approach is applicable to most 3D accelerated devices having a standard High-Definition Multimedia Interface (HDMI) port to drive a HMD (mobiles, tablets, laptops and desktops). We evaluate our system by conducting both a quantitative and a qualitative study.

*e-mail: nik.sidorakis@gmail.com
†e-mail: gkoulieris@isc.tuc.gr
‡e-mail: k.mania@ced.tuc.gr

2 RELATED WORK

2.1 3D User Interfaces

UI design for 2D applications is based on fundamental principles and best practices formed after many years of research. However, user interaction in a 3D spatial context introduces constraints due to the multiple degrees of motion freedom, requiring novel interaction metaphors such as "fly" and "zoom". These metaphors are not applicable in a standard 2D interface [1]. Bowman et al. [2] report an overview of metaphors, conventions and best practices for 3D UI design. 3D UIs are an integral part of Virtual Environments (VEs) in many interactive systems. In this work, we propose an eye-tracking paradigm for the control of modern 3D UIs deployed on commercial HMDs.

2.2 Eye-tracking as an Input Device

Eye tracking has been used in the past as an input device to interact with a 2D UI [4, 5, 12], but not for an immersive 3D MUI. By moving their eyes, users can manipulate an on-screen cursor to point virtual items on the interface and then activate them via prolonged fixations or blinking. Previous research investigated eye-tracker based interfaces for disabled users [4, 5, 12]. Physically disabled users can benefit the most from VR technology since they usually greatly depend on computer aid for recreation or basic communication. However, most computer interfaces for the disabled are neither inexpensive nor easily accessible systems. Eye-tracking has also been used as a context-sensitive help system when a user had difficulty to comprehend a text during reading [9]. A head-mounted eye tracker has been used for interaction with real world planar digital displays [7]. A common issue when using an eye-tracking interface as an input device is known as the Midas' touch problem. Certain eye movements are involuntary and accidental interface activation is frequent [5]. Speech recognition has been used in the past to signify an event [6], however, it required accurate synchronization of gaze and speech data streams in order to be reliable. In our eye-tracking interface we deal with the Midas' touch issue

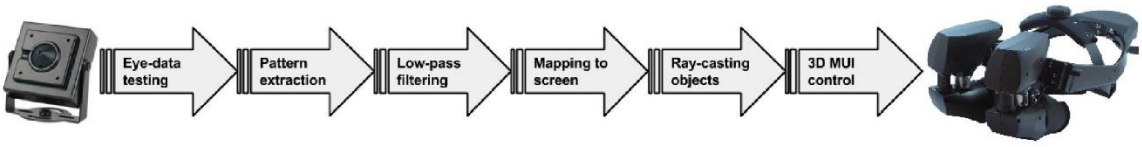


Figure 1: A schematic diagram of the proposed system, introducing principal software components.

by employing an additional mechanical input (switch) to signify a selection in the immersive environment.

Figure 2: The application selector of the proposed 3D MUI. The user is placed at the center.

3 An Eye-Tracking 3D MUI

3.1 Design

In this section, the 3D user interface design and implementation will be presented. The 3D MUI elements are considered to be placed at static locations in the virtual space while a moving observer may interact with them from various locations. For our prototype implementation, the designed applications are centered around a main pivot point while the observer turns around and selects one of the applications (Figure 2). The project is implemented in Unity3D™ and deployed on a stereo NVisor™ SX111 HMD by setting up a virtual stereo camera rig. The rig simulates a virtual head model having two eyes and a neck, similar to the Oculus Rift head tracker implementation, however, without the positional 6DoF tracker [8]. The Inter-Pupillary Distance (IPD) can be individually adjusted for each user.

3.2 Implementation

Our system is comprised of six principal software components (Figure 1). The first, is a raw eye data calibration component performing clear pupil and glint tests, signal smoothing and filtering to eliminate noise. The second component is an eye scan pattern extraction system indicating the direction of the eye movement, fixations and blinks. The third, identifies that a clear glint signal was located and performs a low pass filtering to avoid flicker. The fourth component maps movements from eye space to screen coordinates. The fifth performs ray-casting over the 3D menus to identify fixated items. Finally, the 3D MUI receives data and executes control algorithms for cursor motion manipulation and menu item highlighting. Supplemental software components include a head tracker manager that provides 3DoF data of head movement and a SQLite database used for event logging and statistics.

Figure 3: The eye-controlled Photo Gallery.

3.3 Developed Applications

Six applications based on the eye-tracking 3D UI paradigm have been implemented. User gaze substitutes the mouse pointer while the mechanical switch acts as a selector for the UI. A Photo Gallery allows the user to browse the pictures folder and manipulate photos (Figure 3). The application searches through an image folder and visualizes sub-folders and files on a virtual 3D slide-show supporting .jpg, .gif, .tiff, and .bmp formats. A 3D Music Player explores the user's music folder and exposes virtual 3D geometry for audio and playback control (Figure 4). The virtual speakers visualize the music and vibrate according to the music tempo. A 3D Email Composer consists of a 3D custom-made keyboard and a 3D email form (Figure 5). The keyboard is based on a standard mobile device keyboard layout supporting Latin characters and a set of symbols. In order to compose an email, the user fills his email and password, the receiver's email, the subject and the main body of the mail. A Word Processor allows the user to create, edit and save a .txt file. The scene consists of a 3D paper model representing a notepad and a Latin 3D keyboard. Finally, two immersive mini games were implemented. A puzzle game in which a user-selected picture is fragmented in tiles (Figure 6). The tile layout is then randomized and the user has to re-arrange the tiles to form the original picture and solve the puzzle. Finally, the provided action game is a 3D rendition of the classic flappy bird game including an airplane [3] (Figure 7).

4 User Study

4.1 Materials & Methods

We conducted a user study in order to evaluate the proposed eye-tracking interaction and 3D UI paradigm. We conducted a pilot study in order to identify which application required the greatest gaze-tracking accuracy. The study indicated that the Mail Composer required complex eye movements such as typing on a virtual keyboard and was selected to assess our method. A total of 7 people (2 female, mean age 24.3) participated in the experiment.

Apparatus The 3D UI was displayed on a NVisor™ SX111 HMD, having a stereo SXGA resolution and a Field-of-View (FoV)

Figure 4: The eye-controlled Music Player.

Figure 5: The eye-controlled Mail Composer.

of 102 degrees horizontal by 64 degrees vertical. Participants panned around the virtual environment using an InterSense™ InertiaCube3™ 3DoF head tracker attached to the HMD. Eye-tracking data was recorded using a twin-CCD binocular eye-tracker by Arrington Research™, also attached to the HMD updating at a frequency of 30Hz.

Procedure Participants sat on a swivel chair and were familiarized with the setup in a training session. Before acquiring data, every participant performed the standard eye tracker calibration procedure provided by Arrington Research™. Each individual calibration took approximately 20 seconds during which the subjects gazed at the center of 12 target squares displayed at different locations on the HMD. Following this, participants composed a dictated email by moving their eyes to select letters on a virtual keyboard and pressing a switch to select the letter they wished to write. Then the same email was written on a physical keyboard while still wearing the HMD. The keyboard was occluded by the HMD. Task accuracy and task completion times were recorded. All but one participants (that was excluded from the analysis due to insufficient eye tracking data) successfully executed both tasks by wearing the eye-tracking capable HMD.

After the experiment ended, participants were asked to rate their experience when using the 3D MUI on a 1-7 Likert Scale, by answering four questions, commonly used in qualitative assessments of 3D User Interfaces [13]: (i) The 3D UI is as comfortable as the traditional mouse-keyboard paradigm. (ii) I felt more tired when using the 3D UI than with the traditional mouse-keyboard paradigm. (iii) The 3D UI was more interesting when using it compared to traditional mouse-keyboard paradigm. (iv) I prefer the 3D UI more than the traditional mouse-keyboard paradigm.

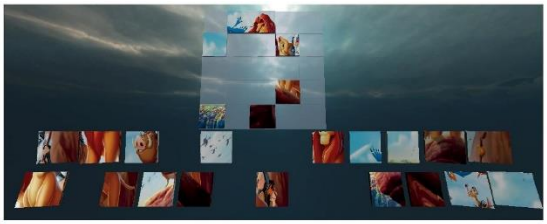


Figure 6: The eye-controlled Puzzle game.

Figure 7: The eye-controlled airplane in the Flappy airplane game.

4.2 Data Analysis & Discussion

An independent-samples t-test was conducted to compare type rate with the eye-tracker versus occluded keyboard input. There was a significant difference in the scores for the eye-tracker (M=257.6s, SD=5570.3) and keyboard (M=123.3s, SD=7169.0) conditions; $t(6) = 2.91$, $p < 0.05$. These results suggest that participants type about 2 times slower on average (Figure 8) with the eye-tracker interface. However, analysing the type error rate for both interfaces (9.3% for the eye-tracker vs 54.19% for the occluded keyboard) indicated that despite the fact that users type faster when utilizing the out-of-view keyboard, they make many more errors (Figure 8).

The results of the qualitative analysis (Figure 9) and thorough discussion with the test subjects indicated that the proposed interaction method is much more enjoyable than a standard keyboard. Fatigue levels where found to be the same for both interfaces, with the exception of one subject (Figure 9). Usage of a 3D UI keyboard does not require good typing skills. After interaction with the rest of the applications, participants stated that they would certainly opt for the 3D UI since it feels more futuristic and therefore more exciting for the users.

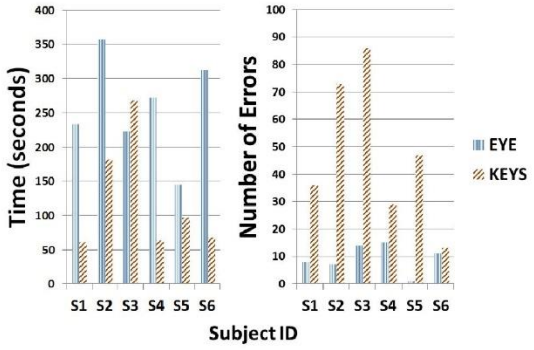


Figure 8: Left: Comparison of task completion times for the eye-tracker input versus blind keyboard input for all subjects. Right: Comparison of type errors for the eye-tracker input versus blind keyboard input for all subjects.

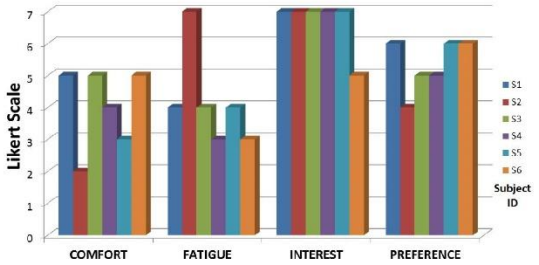


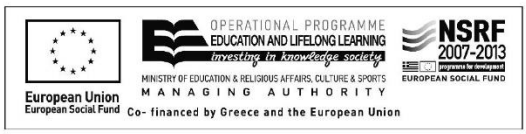
Figure 9: Responses on the four questions of the qualitative questionnaire for the 3D UI keyboard.

5 Conclusion

We presented an innovative gaze-controlled MUI for an eye-tracking capable headset, suitable for modern consumer-grade HMDs. We developed six applications which cover commonly operated actions of everyday computing such as mail composing, music playing and photo viewing. We performed a user study by acquiring both quantitative and qualitative data. The type error rate was lower when utilizing the proposed 3D UI in comparison to an occluded keyboard. The qualitative study indicated that users enjoy considerably more the proposed 3D UI over the traditional input methods such as a keyboard and mouse. A middle-ware API will soon be provided for eye-tracking and blink handling that can be extended to additional MUIs designated for the Oculus Rift™ and Samsung Gear VR™.

Acknowledgements

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II: Investing in knowledge society through the European Social Fund.



References

[1] D. A. Bowman, J. Chen, C. A. Wingrave, J. F. Lucas, A. Ray, N. F. Polys, Q. Li, Y. Haciahmetoglu, J.-S. Kim, S. Kim, et al. New directions in 3d user interfaces. *IJVR*, 5(2):3–14, 2006.

[2] D. A. Bowman, E. Kruijff, J. J. LaViola Jr, and I. Poupyrev. *3D user interfaces: theory and practice*. Addison-Wesley, 2004.

[3] T. F. B. game. Flappy bird, Feb. 2015.

[4] T. E. Hutchinson, K. P. White Jr, W. N. Martin, K. C. Reichert, and L. A. Frey. Human-computer interaction using eye-gaze input. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1527–1534, 1989.

[5] R. J. Jacob and K. S. Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind, Oxford Journal*, 2(3):4, 2003.

[6] M. Kaur, M. Tremaine, N. Huang, J. Wilder, Z. Gacovski, F. Flippo, and C. S. Mantravadi. Where is it? event synchronization in gaze-speech input systems. In *Proceedings of the 5th international conference on Multimodal interfaces*, pages 151–158. ACM, 2003.

[7] D. Mardanbegi and D. W. Hansen. Mobile gaze-based screen interaction in 3d environments. In *Proceedings of the 1st conference on novel gaze-controlled applications*, page 2. ACM, 2011.

[8] Oculus. Oculus developer guide, Feb. 2015.

[9] J. L. Sibert, M. Gokturk, and R. A. Lavine. The reading assistant: eye gaze triggered auditory prompting for reading remediation. In *Proceedings of the 13th annual ACM Symposium on User Interface Software and Technology (SUIT)*, pages 101–107. ACM, 2000.

[10] SMI. Sensomotoric instruments announces eye-tracking upgrade for oculus rift, Feb. 2015.

[11] A. Sutcliffe. Multimedia user interface design. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, page 65, 2009.

[12] M. Yamada and T. Fukuda. Eye word processor (ewp) and peripheral controller for the als patient. *IEEE Proceedings on Physical Science, Measurement and Instrumentation, Management and Education-Reviews*, 134(4):328–330, 1987.

[13] B. Yoo, J.-J. Han, C. Choi, K. Yi, S. Suh, D. Park, and C. Kim. 3d user interface combining gaze and hand gestures for large-scale display. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3709–3714. ACM, 2010.

Appendix B: User Study Questionnaire

Employing binocular eye tracking for the control of a custom made immersive multimedia user interface

Φόρμα Συγκατάθεσης

Σας ζητείται να συμπληρώσετε ένα ερωτηματολόγιο το οποίο θα βοηθήσει την έρευνα και το πειραματικό στάδιο μιας προπτυχιακής εργασίας που πραγματοποιείται στο Πολυτεχνείο Κρήτης, στο τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών. Υπεύθυνοι αυτού του πειράματος είναι ο προπτυχιακός φοιτητής Σιδωράκης Νικόλαος, ο υποψήφιος διδάκτορας Κουλιέρης Γεώργιος Αλέξανδρος και η αναπληρώτρια καθηγήτρια Μανιά Κατερίνα. Η έρευνα πραγματοποιείται για την επαλήθευση μιας τρισδιάστατης διεπαφής χρήστη. Αναμένεται να απασχοληθείτε το πολύ 10 λεπτά. Θα χρησιμοποιήσουμε τα δεδομένα σας ανώνυμα μαζί με πολλά άλλα άλλων συμμετεχόντων.

Κατανοείτε τη φόρμα συγκατάθεσης?	Ναι	Όχι
Δίνετε τα δεδομένα σας για περαιτέρω ανάλυση στα πλαίσια της παρούσας εργασίας?	Ναι	Όχι
Επιβεβαιώνετε ότι είστε τουλάχιστον 18 χρονών?	Ναι	Όχι

Ονοματεπώνυμο	
E-mail	
Ηλικιακή ομάδα	18-22 23-27 28-32 33-37 38-42 > 43
Φύλο	Αρσενικό Θηλυκό
Ημερομηνία	31 / 1 / 2015
Αν σπουδάζετε τώρα παρακαλώ επιλέξτε ποια είναι η βαθμίδα σας	Προπτυχιακός Μεταπτυχιακός Διδακτορικός

Υπογραφή

Ερωτηματολόγιο

Η τρισδιάστατη διεπαφή ήταν το ίδιο άνετη στη χρήση σε σχέση με τον παραδοσιακό τρόπο αλληλεπίδρασης

Διαφωνώ απόλυτα			**Αδιάφορο**		**Συμφωνώ απόλυτα**	
1	**2**	**3**	**4**	**5**	**6**	**7**

Ένιωσα περισσότερη κούραση κατά τη χρήση της τρισδιάστατης διεπαφής σε σχέση με τον παραδοσιακό τρόπο αλληλεπίδρασης

Διαφωνώ απόλυτα			**Αδιάφορο**		**Συμφωνώ απόλυτα**	
1	**2**	**3**	**4**	**5**	**6**	**7**

Η τρισδιάστατη διεπαφή ήταν πιο ενδιαφέρουσα σε σχέση με τον παραδοσιακό τρόπο αλληλεπίδρασης

Διαφωνώ απόλυτα			**Αδιάφορο**		**Συμφωνώ απόλυτα**	
1	**2**	**3**	**4**	**5**	**6**	**7**

Προτιμώ περισσότερο την τρισδιάστατη διεπαφή σε σχέση με τον παραδοσιακό τρόπο αλληλεπίδρασης

Διαφωνώ απόλυτα			**Αδιάφορο**		**Συμφωνώ απόλυτα**	
1	**2**	**3**	**4**	**5**	**6**	**7**