



Electronics and Computer Engineering Dept.

Technical University of Crete

Development of an Eye- Tracked 3D/ Stereoscopic Interactive and Photorealistic Platform for Spatial Cognition Experiments Using the Unreal Game Engine

Paraskeva Christos

A thesis submitted in fulfillment of the requirements 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 – TUC/MUSIC

Abstract

A remarkable interest around spatial navigation motivated research investigating under which circumstances spatial navigation is optimal, often in hazardous environments such as navigating a building on fire. The relationship between gender and spatial abilities and how spatial performance is affected by gender in spatial tasks has been extensively explored. Numerous studies have been carried out in Real and Virtual Environments (VE) investigating gender differences in a variety of navigational tasks. This thesis presents a 3D interactive experimental framework exploring gender differences in spatial navigation, memory performance and spatial awareness in a complex Immersive Virtual Environment (IVE). The immersive simulation implemented in this thesis consisted of a radiosity-rendered space divided in four zones including a kitchen area, a dining area, an office area and a lounge area. The experimental framework as well as the system logging user actions was implemented with the Unreal Development Kit. The space was populated with objects consistent as well as inconsistent with each zone's context. The simulation was then displayed on a stereo head tracked Head Mounted Display with binocular eye tracking capabilities which also recorded eye gaze information. After being exposed to the VE, participants completed an object-based memory recognition task. Participants also reported one of two states of awareness following each recognition response which reflected either the recollection of contextual detail or informed guesses. A clear gender difference was found with female participants correctly identifying objects in their correct location more often than the male participants.

Declaration

Declaration

The work in this thesis is original and no portion of the work referred to here has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning.

Signed:

Date:

Paraskeva Christos

Acknowledgements

Το Λακωνίζειν εστί φιλοσοφείν...

ΆΞΙΟΙ όσοι φθάνουν στο τέλος της διαδρομής των στόχων τους και επιτέλους αντικρίζουν την πολυπόθητη Ιθάκη. Βέβαια όπως όλοι ξέρουμε σημασία δεν έχει ο προορισμός αλλά το ταξίδι μέσα από τις συμπληγάδες της γνώσης και τις σειρήνες που πάντα βρίσκοντε εμπόδιο. Θέλω και εγώ με τη σειρά μου να ευχαριστώ τους δικούς μου δασκάλους για το ευζήν και για όλα τα εφόδια και γνώσεις που μου έχουν δώσει για να ευδοκιμήσω στην ζωή μου. Εντούτοις θέλω να τους υπενθυμίσω ότι η λέξη καθηγητής παράγεται από το ρήμα της αρχαίας *καθηγοῦμαι* (< κατά *+ήγοῦμαι*), που σημαίνει προπορεύομαι και δείχνω το δρόμο, καθοδηγώ και που σε καμία περίπτωση δεν σημαίνει επιβάλλω, παρακινώ είτε άμεσα αλλά κυρίως έμμεσα. Ο νοών νοεΐτω. Ευχαριστώ όλους όσους στάθηκαν δίπλα μου όλα αυτά τα χρόνια και όσους ακόμη και στο ελάχιστο με έχουν βοηθήσει. Ιδιαίτερα και περισσότερο απο όλους λέω ένα μεγάλο ευχαριστώ σε όσους έχουμε διαφωνήσει έντονα (ναι παύλο κυρίως εσένα έχω στο μυαλο) γιατί με έκαναν να δω από άλλη οπτική γωνιά τις απόψεις μου. Τελειώντας με τις επόμενες δύο προτάσεις σε κεφαλαία γράμματα. Δεν εύχομε σε κανένα καλή τύχη γιατί τύχη είναι ο χαρακτήρας μας συνάμα με τις επιλογές μας. Κλείνοντας με την ίδια λέξη που άρχισα , τη λέξη αξία που στις μέρες μας πλέον έχει καταντίσει ο αποδιοπομπεός τράγος του κάθε θεατρίνου και εύχομε σε όλους μας να είμαστε σε κάθε τομέα της ζωής μας πάντα ΆΞΙΟΙ.

Publications

- Paraskeva, C., Koulieris, G.A., Coxon, M., Mania, K.. (Accepted, 2012) Gender Differences in Spatial Awareness in Immersive Virtual Environments: A Preliminary Investigation, *ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry*, Singapore.
- Paraskeva, C., Koulieris, G.A., Coxon, M., Mania, K. (in preparation). Gender Differences in Spatial Awareness in Immersive Virtual Environments, *Human Computer Studies*, Elsevier.

Table of Contents

Abstract	2
Declaration	3
Acknowledgements	4
Publications	5
Table of Contents	6
List of Figures	9
List of Tables	13
1 Chapter 1 – Introduction	14
1.1 Contribution	15
1.2 Thesis Outline	16
2 Chapter 2 – Background	17
2.1 Computer Graphics Rendering	17
2.1.1 The Physical Behavior of the Light.....	18
2.1.2 Computer Graphics Illumination Models.....	20
2.1.3 Ray Tracing.....	22
2.1.4 Radiosity	23
2.1.5 Texturing – Importance of texture maps.....	26
2.2 Virtual Reality Technology and Game Engines	30
2.2.1 Unity 3D	31
2.2.2 Torque 3D	31
2.2.3 Unreal Engine 3 – Unreal Development Kit (UDK).....	32
2.2.4 Unreal Engine 3 – Overview	32
2.3 Immersive, Virtual Reality Systems	32
2.3.1 Stereoscopy.....	33
2.3.2 Head Mounted Displays Device.....	35
2.4 Eye-tracking	36
2.4.1 Eye-tacking methods	37
2.4.2 Vision types.....	39
2.4.3 Eye dominance.....	39
2.5 Memory Schemata and Awareness States	40
2.5.1 Memory and Perception	40
2.5.2 The Remember/Know Paradigm	41
2.5.3 Memory Schemata	42

<Table of Contents

2.5.4	Goal	43
3	Chapter 3 – Software Architecture and Development Framework	44
3.1	Game Engine – Unreal Development Kit (UDK)	44
3.1.1	Unreal Editor	44
3.1.2	Sound engine.....	48
3.1.3	DLL Files	48
3.1.4	Input Manager.....	49
3.1.5	Lighting and Rendering Engine.....	49
3.1.6	Unreal Lightmass	50
3.1.7	UnrealScript	51
3.2	Flash Applications as User Interfaces (UI).....	55
3.2.1	Authoring environment for interactive content.....	56
3.2.2	ActionScript 2.0	56
3.2.3	Connection of the User Interface (UI) with the application	57
3.3	SQL database system as data storage.....	58
3.3.1	SQLite Database Engine	59
3.3.2	SQL Database Schema.....	59
3.3.3	Connection of the Database Engine with the application.....	60
3.4	Eye-tracking device’s software	61
3.4.1	Overview of the software	61
3.4.2	Binocular vs. Monocular Method	64
4	Chapter 4 – UI Implementation	65
4.1	Creating the 3D Virtual Scenes	65
4.1.1	Creating the Visual Content	65
4.1.2	Creating texture maps	68
4.1.3	Setting up the virtual scene in UDK	69
4.2	Application menu as Flash UIs.....	75
5	Chapter 5 – Implementation	77
5.1	UnrealScript Classes	77
5.2	Handling User Input	80
5.3	Logging of events and participants’ actions mechanism	80
5.4	Stereoscopic view setup.....	86
5.5	Deprojection of eye’s gaze data	89
5.6	Communication with the Eye-tracking Device	92
6	Chapter 6 – Experiments	96
6.1	Materials and Methods.....	96

<Table of Contents

6.1.1	Participants and Apparatus.....	96
6.1.2	Visual Content.....	96
6.1.3	Experimental Procedure.....	98
6.1.4	Simulator Sickness.....	106
6.2	Statistical Analysis.....	106
6.2.1	Analysis of Variance.....	106
6.3	Results and Discussion.....	107
	Memory recognition performance:.....	107
7	Chapter 7 – Conclusions.....	111
7.1	Main contributions.....	112
8	References – Bibliography.....	113
	APPENDIX A.....	118
	APPENDIX B.....	121
	APPENDIX C.....	123

List of Figures

Figure 1: The goal of realistic image synthesis: an example from photography.	17
Figure 2: The visible portion of the electromagnetic spectrum.	19
Figure 3: Light transmitted through a material.	19
Figure 4: Light absorbed by a material.....	20
Figure 5: Light refracted through a material.	20
Figure 6: Light reflected off a material in different ways. From left to right, specular, diffuse, mixed, retro-reflection and finally gloss [Katedros 2004].....	20
Figure 7: Graphical Depiction of the rendering equation (Yee 2000).	22
Figure 8: Ray-tracing.	23
Figure 9: Radiosity (McNamara 2000).....	24
Figure 10: Relationship between two patches [Katedros 2004].	25
Figure 11: The hemicube (Langbein 2004).	25
Figure 12: The difference in image quality between ray tracing (middle) and radiosity (right hand image).....	26
Figure 13: 1) 3D model without textures, 2) 3D model with textures.	26
Figure 14: Examples of multitexturing. 1) Untextured sphere, 2) Texture and bump maps, 3) Texture map only, 4) Opacity and texture maps.	27
Figure 15: Taurus pt 92 textured 3d model. Rendered in marmoset engine(real time game engine).	28
Figure 16: Diffuse, normal and specular map of the above 3d model.....	29
Figure 17: Mesh without any texture (left image). Reflection image projected onto the object (right image).	29
Figure 18: Mesh with diffuse map only (left image). Opacity texture applied on the mesh (right image).	30
Figure 19: Normal mapping used to re-detail simplified meshes.....	30
Figure 20: Architecture of a game engine.	31
Figure 21: Classic Virtual reality HMD.....	33
Figure 22: To view the stereoscopic image on the left a pair of shutter glasses as showed on the second image is needed.	33
Figure 23: Polarized 3D image (on the left). Polarized 3D Glasses (on the right).....	34
Figure 24: Anaglyph 3D photograph (on the left). Anaglyph red/cyan 3D Glasses (on the right).	34
Figure 25: To the left Binocular Image with Full Overlap, to the right Binocular Image with Partial Overlap.	35
Figure 26: To the left Full Overlap view, to the right Partial Overlap view.	35
Figure 27: nVisor SX111 Head Mounted Display (on the left), head-tacker InertiaCube3 (on the right).....	36
Figure 28: A participant wearing the head mounted display model nVisor SX111 during the experiment.	36
Figure 29: A subject wearing a scleral search coil in conventional manner (A), the wire exiting directly (white arrow). When wearing a bandage lens on top of the coil, the wire is applied to	

List of Figures

the sclera and exits the eye near the medial canthus (B, gray arrow). The bandage lens (C, centre) is 5 mm larger in diameter than a scleral search coil (C, right).	37
Figure 30: A subject wearing a headset attached by two optical sensors, those within the red circles above.	38
Figure 31: A woman wearing EOG goggles.....	38
Figure 32: Eye dominance test sketch.....	40
Figure 33: The Unreal Editor with a virtual scene loaded.	44
Figure 34: Static Mesh Editor for an imported object.	45
Figure 35: Properties of a Static Mesh Actor instance.	46
Figure 36: Light Actor properties.	46
Figure 37: Unreal Kismet set up for testing the experiment scene.	47
Figure 38: The Material Editor with a sample material.	48
Figure 39: UDK's Sound Editor and a sound imported into a scene.	48
Figure 40: Class Hierarchy Diagram for 3 user-created classes: VEGame , VEPawn and VEPlayerController.....	53
Figure 41: Flash Authoring environment with a Flash User Interface loaded.	56
Figure 42: Initiation of a Flash User Interface Application through Unreal Kismet.	58
Figure 43: Using SQLite shell to make a database.	59
Figure 44: Exploring the database schema "VREXPERIMENT_FINAL", using the SQLite Administrator tool.	60
Figure 45: Game engine using a dynamic link library in order to access the database file.	61
Figure 46: Start-up arrangement of the user windows.....	62
Figure 47: EyeCamera window.	62
Figure 48: EyeSpace window on the left before calibration. EyeSpace window to the right after calibration of a participant's eye.	63
Figure 49: PenPlot and GazeSpace windows on the left and right images respectively.....	63
Figure 50: Control Window in tab of the EyeA (right eye) where can be adjusted various parameters for the "Pupil Location" method.....	64
Figure 51: A 3D object representation of a lounge table. The 3d object was created in 3ds Max and its geometry will be exported to UDK.	66
Figure 52: The UVW mapping of an object in 3ds Max. This UV channel is used by UDK in order to apply a material correctly onto the object.....	66
Figure 53: The UVW unwrapping of an object in 3ds Max. This UV channel is used by UDK in order to realistically illuminate an object, giving more shading detail in the bigger polygons of the object in the channel.	67
Figure 54: The final scene with all the 3D objects created in 3ds Max. The final scene does not include items that were placed in each zone because their choice was made much later.	67
Figure 55: The image represents the normal map applied on the cylinder model as showing in the figure below (second image).	68
Figure 56: First picture shows the original diffuse map (image), second one is after applying a normal map on it.	68
Figure 57: The image represents the normal map applied on the cylinder model as showing in the figure below (second image).	69
Figure 58: First picture shows the original diffuse map (image), second one is after applying a specular map on it.	69

List of Figures

Figure 59: A manually blocking volume was created and applied on the wine bottle 3D model. One thing to pay attention is that in the properties of the blocking volume is the option in the red rectangular. Collision type is set to "COLLIDE_BlockAll" which means that nothing can go through that volume.	70
Figure 60: On the left is a grey matte material, with no specular color. On the right is the same grey material with white specular color.	71
Figure 61: A material for the surface of a carpet with a normal map defined. As can be seen from the sample sphere with the material applied on it, the surface of the carpet is relief.	71
Figure 62: A carpet surface object with the normal mapped material applied on it. Although the carpet's geometry is simple, with only 43 triangles, the lighting effects give the illusion of a wool (and complex) surface.	72
Figure 63: A translucent material representing the glass in a window.....	72
Figure 64: An unlit scene with the modeled 3D objects in UDK. The created materials are applied to the respective objects.	73
Figure 65: Default Midday lighting configuration.....	74
Figure 66: Screenshot of final scene after the light computation using unreal light mass.....	74
Figure 67: The start Flash menu that was displayed when the experiment application was started.	75
Figure 68: The start menu Flash UI is loaded and displayed immediately after the virtual scene becomes visible.	76
Figure 69: Logging table showing sample participant's records from the database file.	81
Figure 70: ParticipantInfo table showing sample participant's records from the database file.	82
Figure 71: The four LOS Triggers positioning in the center of each zone.	84
Figure 72: The trigger's "LOS" generated event evokes the Activated method in the Zones_Sequence class.....	85
Figure 73: Simultaneous activation of two LOS Triggers.....	86
Figure 74: The partial overlap view in the HMD.	86
Figure 75: The two output images displayed each on left and right screen respectively in the HMD.	88
Figure 76: The final scene that user views within the HMD.....	89
Figure 77: Transformation of the user's viewpoint from a 2D vector into 3D world-space origin and direction.	89
Figure 78: In the image above the green plane represents the world, the red frustum represents the world view frustum, the blue spot represents the 2D mouse position and the blue line represents the trace. The flat top of the frustum represents the screen (in this case, viewing the world over the top). The canvas deprojection function converts a 2D position into the world.	91
Figure 80: The status windows of Viewpoint application. In the red rectangular is the total number of registrations from third party applications.	94
Figure 81: The experimental scene (side-view).....	98
Figure 82: The sample image projected on the HMD.	99
Figure 83: Calibration for eye-gaze on left dominant eye (left picture) and right dominant eye (right picture).....	99
Figure 84: The training Virtual Environment.	100
Figure 85: First page of the leaflet containing the Lounge Area.	102

List of Figures

Figure 86: Second page of the leaflet containing the Office Area103
Figure 87: Third page of the leaflet containing the Kitchen Area.....104
Figure 88: Fourth page of the leaflet containing the Dining Area.105

List of Tables

Table 1: Objects existing in each zone of the house.	65
Table 2: Quality code and the respectively information about the new data received.	94
Table 3: Consistent objects existing in each zone of the house.	97
Table 4: Inconsistent objects existing in each zone of the house.....	97
Table 5: Number of correct responses and standard deviations.	107
Table 6: Proportion of correct responses and standard deviations.	107
Table 7: Mean confidence rating and standard deviation as a function of Gender (Males, Females) and context consistency (consistent, inconsistent).	108
Table 8: Total Number of Fixations.....	109
Table 9: Total Time Spent Fixating (seconds).	110
Table 10: Mean Time Spent Fixating (seconds).	110

1 Chapter 1 – Introduction

During the last decade, a remarkable interest around spatial navigation motivated research investigating under which circumstances spatial navigation is optimal, often in hazardous environments such as navigating a building on fire. Furthermore, the relationship between gender and spatial abilities and how spatial performance is affected by gender in spatial tasks has been extensively explored. Numerous studies have been carried out in real and Virtual Environments (VE) investigating gender differences in a variety of navigational tasks such as wayfinding, navigation using digital or hand-held paper map, distance estimation, position recognition, object-location/route learning etc. As several studies showed that spatial performance in a Virtual Environment exploited “real life’s” abilities [Laria et al., Lovden et al., Moffat et al.], such research is attracting interest in the research world.

The term Virtual Environment (VE) is generally understood to mean an environment that is described in three dimensions be presented on a computer display. They are perhaps most commonly encountered in computer games, but are also found in research, simulation, training and design— particularly architectural design. The term Immersive Virtual Environment (IVE) in this thesis is referred to VEs that are displayed using equipment that produces an ego-centric view, allowing the view position and direction to be changed by moving the head and body in a natural way [Sutherland 1965]. Today, this may be achieved through the use of an at least three degrees-of-freedom spatial tracker and a stereoscopic head-mounted display (HMD). When using an IVE one can attain a sense that one is actually present within the virtual environment that is displayed, and ‘presence’ has in fact been identified as a key feature for their general use [Held and Durlach 1992], or even their defining factor in terms of the human experience [Steuer 1992].

Thus IVEs is now becoming an increasingly popular alternative approach for research exploration of gender differences in spatial navigation. Robust gender differences in training effectiveness of VEs have been revealed in recent literature [Ross et al. 2006]. Males showcase performance superiority while conducting spatial tasks such as navigating in virtual mazes and way finding [Lovden et al. 2007] in a novel environment using different types of cues or maps. Such superiority is reduced or follows dissimilar patterns according to the task requirements. Female participants responded faster in a 2D matrix navigation task than males when landmark instructions were provided; however, when the same participants participated in a recognition task, male participants recognized key elements involved in a previously viewed video of a real-world driving scene more accurately than female participants [Kim et al. 2007]. Previous research has also revealed that there are no gender differences in the use of different spatial strategies based on either geometric or landmark information when navigating through virtual mazes such as water or radial arm mazes [Lauren et al. 2005]. In spite of the premise that men have more experience with video games than women, it seems that video game experience does not predict the success of spatial tasks [Chai et al. 2009]. On the other hand, interesting findings from earlier research [Desney et al. 2003] present specific benefits for females in spatial navigation with wider fields of view (FOV) of large displays. A recent study proposed a cognitive map model integrated by two parallel map components that

are constructed from two distinct classes of cues: directional cues and positional cues respectively [Jacobs et al. 2003]. In a target location experiment men were overall more accurate in estimating the target location. It was then concluded that gender differences influence navigation performance in humans [Chai et al. 2009]. In addition, similar research investigated the effect of directional cues such as sky and slant but also positional such as trees on a hidden target memory test with cue removal. Men and women performance was impaired when directional and positional cues were removed [Chai et al. 2009]. Those findings supported previous reports that gender differences in spatial memory arise from the dissociation between a preferential reliance on directional cues in males and on position cues in females [Noah et al. 1998, Mueller et al. 2008].

Spatial awareness and memory is crucial for human performance efficiency of any task that entails perception of space. Awareness states accompany the retrieval of spaces after exposure. Memory of spaces may also be influenced by the context of the environment. The main premise of this work is that memory performance, as explored in previous research, is an imperfect reflection of the cognitive activity underlying memory recognition and could be complemented by self-report of how memory recollections are induced rather than just what is remembered.

This thesis focuses upon exploring the effect of gender (male vs female) on object-location recognition memory and its associated awareness states while immersed in a synthetic simulation of a complex scene displayed on a Head Mounted Display (HMD). Simultaneous eye-tracking during exposure to the VE offers additional information of attention and the eye's gaze patterns. This information is at a high enough resolution to be useful in determining and log the different attentional patterns of participants. Those data could be correlated to memory performance and spatial awareness states data.

The system in this thesis can also be utilized for any spatial cognition experiment besides investigating gender effects after minor modifications. Therefore, it could be exploited as a generic platform for spatial cognition experiments employing eye tracking data while allowing users to navigate IVEs in real-time. The system includes not only the visual front-end visible to the user but also a database system to handle eye tracking and other user-related information.

1.1 Contribution

As safety and cost issues in relation to training in real world task situations are becoming increasingly complex, simulators have proven to be the most successful arena for production of synthetic environments. Within simulators, such as flight simulators, flight crew can train to deal with emergency situations, gain familiarity with new aircraft types and learn airfield specific procedures. It is argued that training in a simulator with maximum fidelity would result in transfer equivalent to real-world training, since the two environments would be indistinguishable. However, there is always a trade-off between visual/interaction fidelity and computational complexity. The key in this trade-off is to maintain users' suspension of

disbelief and spatial awareness in the VE, while keeping the rendering computation as simple as possible, so as to be performed by computers in real-time.

The contribution of this thesis is the innovative application designed to render an interactive IVE on a Head Mounted Display in combination with the Eye-tracking technology enabling the conduct of experiments examining gender differences in spatial navigation, memory performance and spatial awareness employing fundamental memory protocols derived from cognitive psychology. Gender differences were explored by responses to a questionnaire, as well as examining the eye's gaze data during the navigation in the IVE. The experimental study presented here investigates the effect of gender on both the accuracy and the phenomenological aspects of object memories acquired in an IVE.

1.2 Thesis Outline

This thesis is divided into a number of chapters, which will be outlined below.

Chapter 2 – Background: This chapter introduces a set of fundamental terms in computer graphics starting with defining light and its properties, light energy, photometry and radiometry. Subsequently, computer graphics illumination models are analyzed. Furthermore, this chapter illustrates the complex variety of tools and equipment required to create, view and interact with immersive VEs. It provides background information regarding the key technologies used in order to implement the experimental framework and experimental protocol put forward and an overview of the technologies necessary to display and interact with immersive VEs. Moreover, in this chapter, previous research utilizing Eye-tracking methods are analyzed including the one used in this research project.

Chapter 3 – Software Architecture and Development Framework: In this chapter, the technical requirements of the Eye-tracked stereoscopic 3D interactive system are introduced. The architecture of the application developed for the experiments is presented, along with the inherent architecture of the Unreal Development Kit (UDK) used to develop it. In addition, an overview of the Eye-tacked software and its capabilities is introduced.

Chapter 4 – User Interface Implementation: In this chapter, the implementation of the User Interfaces (UI) as presented to the users wearing the HMD is described. The steps taken to create the 3D scene, the textures applied on each 3D object and the final adjustments as lighting, within UDK in order to make the whole scene photorealistic. Moreover, the authoring of a Flash application as a main application menu in order to control the different stages of the experiment and embed it in the complete system is presented.

Chapter 5 – Implementation: Chapter 5 describes in detail the implementation of the Interactive computer graphics framework. The technical issues that occurred are explained, as well as the decisions taken to address them. More specifically, there are examples and source code samples demonstrated, in relation to how the application met the requirements which were requisite in order to conduct eye-tacking experiments.

Chapter 6 – Experiments: This chapter is concerned with the experimental methods employed when the actual experiments were conducted with the Eye-tracking device. The experimental procedure is presented, as well as the results of the experiments.

Chapter 7 – Conclusions: In the final chapter, the conclusions of this thesis are presented as well as hints about future work.

2 Chapter 2 – Background

A Virtual Environment (VE) is a computer simulated scene which can be interactively manipulated by users. Typical scenes used in such environments generally comprise of geometric 3D models, shades, textures, images and lights which are converted into final images through the rendering process. The rendering process must be conducted in real time in order to provide scenes which are updated in a reacting to user interaction. An immersive virtual environment (IVE) perpetually surrounds the user within the VE. The first and second section of this chapter present a set of fundamental terms of computer graphics. The third section describes immersive, Virtual Reality systems along with stereo rendering and the devices used in such environments. The fourth section presents the most current methods used for Eye-Tracking. Finally, the fifth section provides all the background information needed regarding memory schemata and awareness states employed in this thesis.

2.1 Computer Graphics Rendering

As described in Chapter 1 there is a great need for realistic rendering of computer graphical images in real time. The term ‘realistic’ is used broadly to refer to an image that captures and displays the effects of light interacting with physical objects occurring in real environments looking perceptually plausible to the human eye, e.g. as a painting, a photograph or a computer generated image (Figure 1).



Figure 1: The goal of realistic image synthesis: an example from photography.

There are no previously agreed-upon standards for measuring the actual realism of computer-generated images. In many cases, physical accuracy may be used as the standard to be achieved. Perceptual criteria are significant rather than physics based simulations to the

point of undetermined ‘looks good’ evaluations. Ferwerda (2003) proposes three levels of realism requiring consideration when evaluating computer graphical images. These are physical realism, in which the synthetic scene is an accurate point-by-point representation of the spectral radiance values of the real scene; photorealism, in which the synthetic scene produces the same visual response as the real scene even if the physical energy depicted from the image is different compared to the real scene; and finally functional realism, in which the same information is transmitted in real and synthetic scenes while users perform visual tasks targeting transfer of training in the real world [Ferwerda 2003]. Physical accuracy of light and geometry does not guarantee that the displayed images will seem real. The challenge is to devise real time rendering methods which will produce perceptually accurate synthetic scenes in real – time. This chapter describes how different rendering models produce their realistic graphical images.

2.1.1 The Physical Behavior of the Light

Light is one form of *electromagnetic radiation*, a mode of propagation of energy through space that includes radio waves, radiant heat, gamma rays and X-rays. One way in which the nature of electromagnetic radiation can be pictured is as a pattern of waves propagated through an imaginary medium. The term ‘visible light’ is used to describe the subset of the spectrum of electromagnetic energy to which the human eye is sensitive. This subset, usually referred to as the *visual range* or the *visual band*, consists of electromagnetic energy with wavelengths in the range of 380 to 780 nanometers, although the human eye has very low sensitivity to a wider range of wavelengths, including the infrared and ultraviolet ranges. The range of visible light is shown in Figure 2. As shown, the wavelength at which the human eye is most sensitive is 555 nm.

In the field of computer graphics three types of light interaction are primarily considered: *absorption, reflection and transmission*. In the case of absorption, an incident photon is removed from the simulation with no further contribution to the illumination within the environment. Reflection considers incident light that is propagated from a surface back into the scene and transmission describes light that travels through the material upon which it is incident and can then return to the environment, often from another surface of the same physical object. Both reflection and transmission can be subdivided into three main types:

Specular: When the incident light is propagated without scattering as if reflected from a mirror or transmitted through glass.

Diffuse: When incident light is scattered in all directions.

Glossy: This is a weighted combination of diffuse and specular.

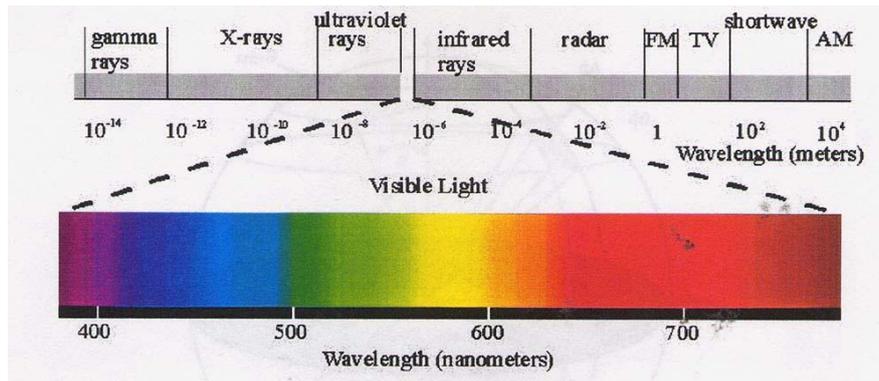


Figure 2: The visible portion of the electromagnetic spectrum.

Most materials do not fall exactly into one of the material categories described above but instead exhibit a combination of specular and diffuse characteristics.

In order to create shaded images of three dimensional objects, we should analyze in detail how the light energy interacts with a surface. Such processes may include emission, transmission, absorption, refraction, interference and reflection of light [Palmer 1999].

- **Emission** is when light is emitted from an object or surface, for example the sun or man-made sources, such as candles or light bulbs. Emitted light is composed of photons generated by the matter emitting the light; it is therefore an intrinsic source of light.
- **Transmission** describes a particular frequency of light that travels through a material returning into the environment unchanged as shown in Figure 3. As a result, the material will be transparent to that frequency of light.

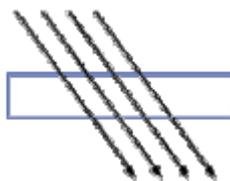


Figure 3: Light transmitted through a material.

- **Absorption** describes light as it passes through matter resulting in a decrease in its intensity as shown in Figure 4, i.e. some of the light has been absorbed by the object. An incident photon can be completely removed from the simulation with no further contribution to the illumination within the environment if the absorption is great enough.

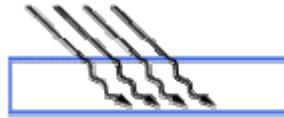


Figure 4: Light absorbed by a material.

- **Refraction** describes the bending of a light ray when it crosses the boundary between two different materials as shown in Figure 5. This change in direction is due to a change in speed. Light travels fastest in empty space and slows down upon entering matter. The refractive index of a substance is the ratio of the speed of light in space (or in air) to its speed in the substance. This ratio is always greater than one.

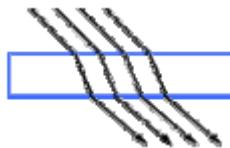


Figure 5: Light refracted through a material.

- **Interference** is an effect that occurs when two waves of equal frequency are superimposed. This often happens when light rays from a single source travel by different paths to the same point. If, at the point of meeting, the two waves are in phase (the crest of one coincides with the crest of the other), they will combine to form a new wave of the same frequency. However, the amplitude of this new wave is the sum of the amplitudes of the original waves. The process of forming this new wave is called *constructive interference* [NightLase 2004].
- **Reflection** considers incident light that is propagated from a surface back into the scene. Reflection depends on the smoothness of the material's surface relative to the wavelength of *the radiation* (ME 2004). A rough surface will affect both the relative direction and the phase coherency of the reflected wave. Thus, this characteristic determines both the amount of radiation that is reflected back to the first medium and the purity of the information that is preserved in the reflected wave. A reflected wave that maintains the geometrical organization of the incident radiation and produces a mirror image of the wave is called a *specular reflection*, as can be seen in Figure 6.

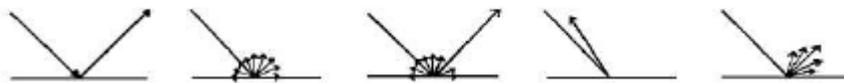


Figure 6: Light reflected off a material in different ways. From left to right, specular, diffuse, mixed, retro-reflection and finally gloss [Katedros 2004].

2.1.2 Computer Graphics Illumination Models

An illumination model computes the color at a point in terms of light directly emitted by the light source(s). A *local illumination model* calculates the distribution of light that comes directly from the light source(s). A *global illumination* model additionally calculates reflected light from all the surfaces in a scene which could receive light indirectly via interreflections from

other surfaces. Global illumination models include, therefore, all the light interaction in a scene, allowing for soft shadows and color bleeding that contribute towards a more photorealistic image. The rendering equation expresses the light being transferred from one point to another (Kajiya 1986). Most illumination computations are approximate solutions of the rendering equation:

$$I(x,y) = g(x,y) [\epsilon(x,y) + \int_S p(x,y,z) I(y,z) dz]$$

where

x,y,z are points in the environment,

$I(x,y)$ is related to the intensity passing from y to x ,

$g(x,y)$ is a 'geometry' term that is 0 when x,y are occluded from each other and 1 otherwise,

$p(x,y,z)$ is related to the intensity of light reflected from z to x from the surface at y , the integral is over all points on all surfaces S .

$\epsilon(x,y)$ is related to the intensity of light that is emitted from y to x .

Thus, the rendering equation states that the light from y that reaches x consists of light emitted by y itself and light scattered by y to x from all other surfaces which themselves emit light and recursively scatter light from other surfaces. The distinction between *view-dependent* rendering algorithms and *view-independent* algorithms is a significant one. *View-dependent* algorithms discretise the view plane to determine points at which to evaluate the illumination equation, given the viewer's direction, such as ray-tracing [Glassner 2000]. *View-independent* algorithms discretise the environment and process it in order to provide enough information to evaluate the illumination equation at any point and from any viewing direction, such as radiosity.

A global illumination model adds to the local illumination model, the light that is reflected from other non-light surfaces to the current surface. A global illumination model is physically correct and produces realistic images resulting in effects such as color bleeding and soft shadows. When measured data is used for the geometry and surface properties of objects in a scene, the image produced should then be theoretically indistinguishable from reality. However, global illumination algorithms are also more computationally expensive.

Global illumination algorithms produce solutions of the rendering equation proposed by Kajiya (1990):

$$L_{out} = L_E + \int L_{in} f_r \cos(\theta) d\omega$$

where L_{out} is the radiance leaving a surface, L_E is the radiance emitted by the surface, L_{in} is the radiance of an incoming light ray arriving at the surface from light sources and other surfaces, f_r is the bi-directional reflection distribution function of the surface, θ is the angle between the surface normal and the incoming light ray and $d\omega$ is the differential solid angle around the incoming light ray.

The rendering equation is graphically depicted in Figure 7. In this figure L_{In} is an example of a direct light source, such as the sun or a light bulb, L'_{In} is an example of an indirect light source i.e. light that is being reflected off another surface, R, to surface S. The light seen by the eye, L_{Out} , is simply the integral of the indirect and direct light sources modulated by the reflectance function of the surface over the hemisphere Ω .

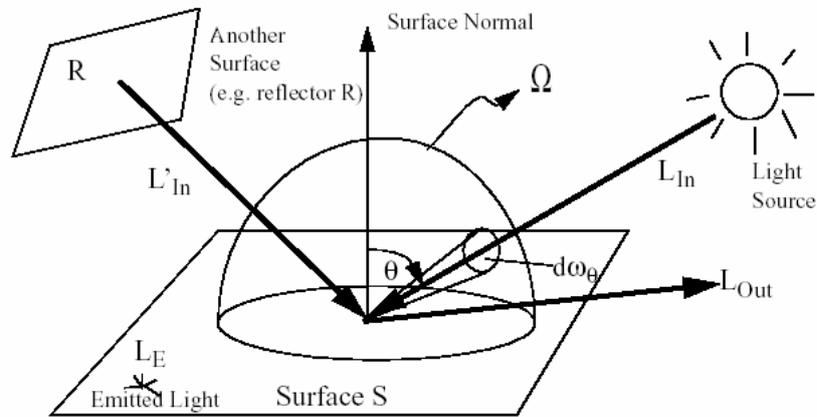


Figure 7: Graphical Depiction of the rendering equation (Yee 2000).

The problem of global illumination can be seen when you have to solve the rendering equation for each and every point in the environment. In all but the simplest case, there is no closed form solution for such an equation so it must be solved using numerical techniques and therefore this implies that there can only be an approximation of the solution [Lischinski 2004]. For this reason most global illumination computations are approximate solutions to the rendering equation.

The two major types of graphics systems that use the global illumination model are *radiosity* and *ray tracing*. For the processes of this work, *Lightmass* was used, which is a global illumination algorithm that combines the *radiosity* and *ray-tracing* algorithm techniques and is developed and supported by the Unreal Development Kit (UDK).

2.1.3 Ray Tracing

Ray tracing is a significant global illumination algorithm which calculates specular reflections (view dependent) and results in a rendered image. Rays of light are traced from the eye through the centre of each pixel of the image plane into the scene, these are called *primary rays*. When each of these rays hits a surface it spawns two child rays, one for the reflected light and one for the refracted light. This process continues recursively for each child ray until no object is hit, or the recursion reaches some specified maximum depth. Rays are also traced to each light source from the point of intersection. These are called *shadow rays* and they account for direct illumination of the surface, as shown in Figure 8. If a shadow ray hits an object before intersecting with the light source(s), then the point under consideration is in shadow. Otherwise, there must be clear path from the point of intersection of the primary ray to the light source and thus a local illumination model can be applied to calculate the contribution of the light source(s) to that surface point.

The simple ray tracing method outlined above has several problems. Due to the recursion involved and the possibly large number of rays that may be cast, the procedure is inherently expensive. Diffuse interaction is not modeled, nor is specular interaction, other than that by perfect mirrors and filters. Surfaces receiving no direct illumination appear black. In order to overcome this, an indirect illumination term, referred to as *ambient light*, is accounted for by a constant ambient term, which is usually assigned an arbitrary value [Glassner 2000]. Shadows are hard-edged and the method is very prone to aliasing. The result of ray tracing is a single image rendered for a particular position of the viewing plane, resulting in a view –dependent technique.

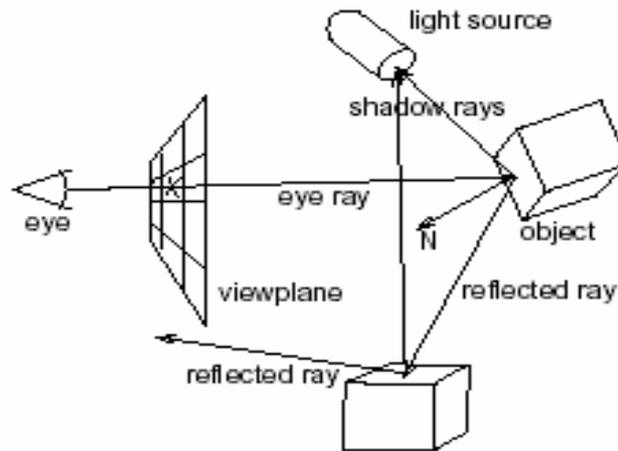


Figure 8: Ray-tracing.

2.1.4 Radiosity

Radiosity calculates diffuse reflections in a scene and results in a finally divided geometrical mesh and it is the main photorealistic rendering method utilized in the immersive environment of this thesis. The scenes produced in this work have been rendered using radiosity calculations. The *radiosity* method of computer image generation has its basis in the field of thermal heat transfer [Goral et al. 1984]. The heat transfer theory describes radiation as the transfer of energy from a surface when that surface has been thermally excited. This encompasses both surfaces that are basic emitters of energy, as with light sources and surfaces that receive energy from other surfaces and thus have energy to transfer. The thermal radiation theory can be used to describe the transfer of many kinds of energy between surfaces, including light energy.

As in thermal heat transfer, the basic radiosity method for computer image generation makes the assumption that surfaces are diffuse emitters and reflectors of energy, emitting and reflecting energy uniformly over their entire area. Thus, the radiosity of a surface is the rate at which energy leaves that surface (energy per unit time per unit area). This includes the energy emitted by the surface as well as the energy reflected from other surfaces in the scene. Light sources in the scene are treated as objects that have self emittance.

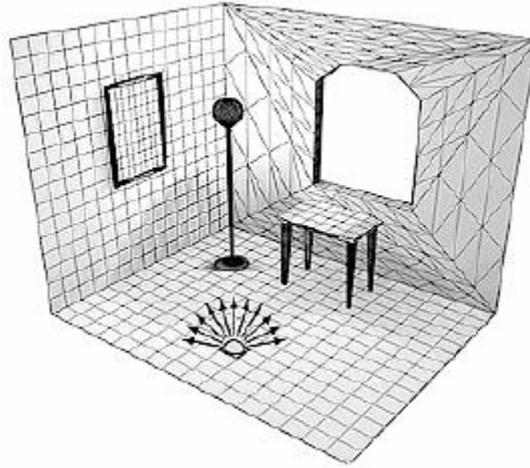


Figure 9: Radiosity (McNamara 2000).

The environment is divided into surface patches, Figure 9, each with a specified reflectivity and between each pair of patches there is a *form factor* that represents the proportion of light leaving one patch (*patch i*) that will arrive at the other (*patch j*) [Siegel and Howell 1992].

Thus the radiosity equation is:

$$B_i = E_i + r_i \sum_j F_{ij} B_j$$

Where:

B_i = Radiosity of *patch i*

E_i = Emissivity of *patch i*

r_i = Reflectivity of *patch i*

B_j = Radiosity of *patch j*

F_{ij} = Form factor of *patch j* relative to *patch i*

Where the form factor, F_{ij} , is the fraction of energy transferred from *patch i* to *patch j* and the reciprocity relationship [Siegel and Howell 1992] states:

$$A_j F_{ji} = A_i F_{ij}$$

Where A_j and A_i are the areas of patch j and i respectively, as shown in Figure 10.

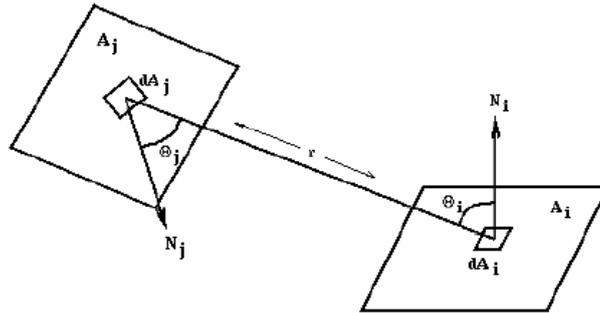


Figure 10: Relationship between two patches [Katedros 2004].

As the environment is closed, the emittance functions, reflectivity values and form factors form a system of simultaneous equations that can be solved to find the *radiosity* of each patch. The radiosity is then interpolated across each of the patches and finally the image can then be rendered.

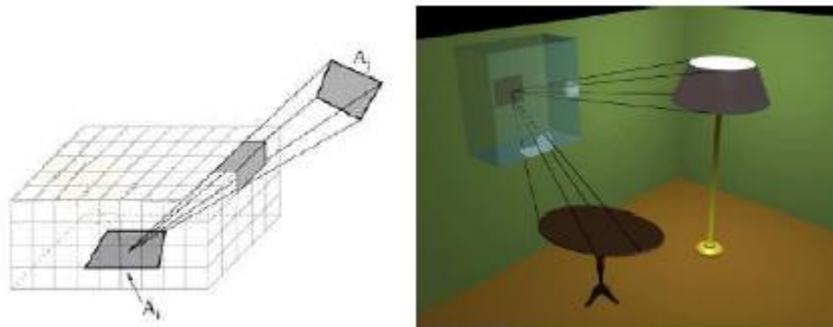


Figure 11: The hemicube (Langbein 2004).

Radiosity assumes that an equilibrium solution can be reached; that all of the energy in an environment is accounted for, through absorption and reflection. It should be noted that because of the assumption of only perfectly diffuse surfaces, the basic radiosity method is **viewpoint independent**, i.e. the solution will be the same regardless of the viewpoint of the image. The diffuse transfer of light energy between surfaces is unaffected by the position of the camera. This means that as long as the relative position of all objects and light sources remains unchanged, the radiosity values need not be recomputed for each frame. This has made the radiosity method particularly popular in architectural simulation, targeting high-quality walkthroughs of static environments. Figure 12 demonstrates the difference in image quality that can be achieved with radiosity compared to ray tracing.



Figure 12: The difference in image quality between ray tracing (middle) and radiosity (right hand image).

Progressive refinement radiosity [Cohen et al. 1988] works by not attempting to solve the entire system simultaneously. Instead, the method proceeds in a number of passes and the result converges towards the correct solution. At each pass, the patch with the greatest unshot radiosity is selected and this energy is propagated to all other patches in the environment. This is repeated until the total unshot radiosity falls below some threshold. Progressive refinement radiosity generally yields a good approximation to the full solution in far less time and with lesser storage requirements, as the form factors do not all need to be stored throughout. Many other extensions to radiosity have been developed and a very comprehensive bibliography of these techniques can be found in [Ashdown 2004].

2.1.5 Texturing – Importance of texture maps

Texture mapping is a method for adding detail, surface texture (a bitmap or raster image), or color to a computer-generated graphic or 3D model. Texture mapping is used for creating 3d objects for objects, avatars, rooms for virtual worlds such as IMVU (Instant Messaging Virtual Universe) which is an online social entertainment website and Secondlife PC game. For example in IMVU a mesh is produced by a developer and if it is left as 'derivable' then other creators can apply their own textures to that object. This leads to different texture maps of the same mesh being produced. The textures can be as simple or complex as the developer wishes. The size of texture map is dependent on the developer but is recommended to be having pixel width/height of a combination from 32, 64, 128, 256 and 512.

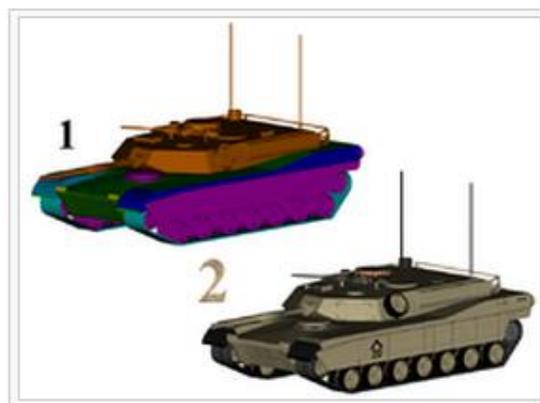


Figure 13: 1) 3D model without textures, 2) 3D model with textures.

A **texture map** is applied (mapped) to the surface of a shape or polygon [Jon Radoff 2008]. This process is akin to applying patterned paper to a plain white box. Every vertex in a polygon is assigned a texture coordinate (which in the 2d case is also known as a UV coordinate) either via explicit assignment or by procedural definition. Image sampling locations are then interpolated across the face of a polygon to produce a visual result that seems to have more richness than could otherwise be achieved with a limited number of polygons. Multitexturing is the use of more than one texture at a time on a polygon. For instance, a light map texture may be used to light a surface as an alternative to recalculating that lighting every time the surface is rendered. Another multitexture technique is bump mapping, which allows a texture to directly control the facing direction of a surface for the purposes of its lighting calculations; it can give a very good appearance of a complex surface, such as tree bark or rough concrete that takes on lighting detail in addition to the usual detailed coloring. Bump mapping has become popular in recent video games as graphics hardware has become powerful enough to accommodate it in real-time.

The way the resulting pixels on the screen are calculated from the texels (texture pixels) is governed by texture filtering. The fastest method is to use the nearest-neighbour interpolation, but bilinear interpolation or trilinear interpolations between mipmaps are two commonly used alternatives which reduce aliasing or jaggies. In the event of a texture coordinate being outside the texture, it is either clamped or wrapped.

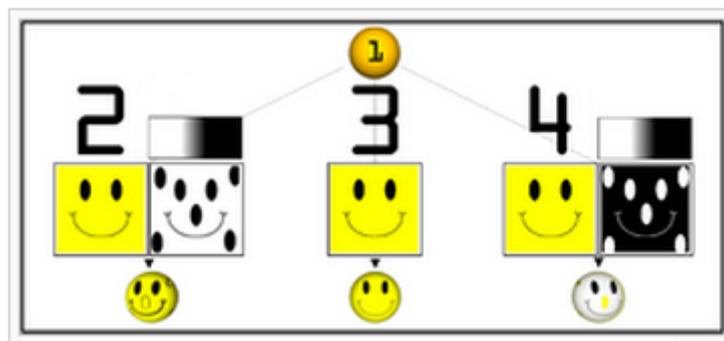


Figure 14: Examples of multitexturing.

1) Untextured sphere, 2) Texture and bump maps, 3) Texture map only, 4) Opacity and texture maps.

The essential map types are described below:

Color (or Diffuse) Maps: As the name would imply, the first and most obvious use for a texture map is to add color or texture to the surface of a model. This could be as simple as applying a wood grain texture to a table surface, or as complex as a color map for an entire game character (including armor and accessories). However, the term texture map, as it's often used is a bit of a misnomer—surface maps play a huge role in computer graphics beyond just color and texture. In a production setting, a character or environment's color map is usually just one of three maps that will be used for almost every single 3D model.

Specular Map: (Also known as a gloss map). A specular map tells the software which parts of a model should be shiny or glossy, and also the magnitude of the glossiness. Specular maps are named for the fact that shiny surfaces, like metals, ceramics, and some plastics show a strong specular highlight (a direct reflection from a strong light source). Specular highlights are the white reflection on the rim of a coffee mug. Another common example of specular reflection is the tiny white glimmer in someone's eye, just above the pupil.

A specular map is typically a grayscale image, and is absolutely essential for surfaces that aren't uniformly glossy. An armored vehicle, for example, requires a specular map in order for scratches, dents, and imperfections in the armor to come across convincingly. Similarly, a game character made of multiple materials would need a specular map to convey the different levels of glossiness between the character's skin, metal belt buckle, and clothing material.

Bump, Displacement, or Normal Map: A bit more complex than either of the two previous examples, bump maps are a form of texture map that can help give a more realistic indication of bumps or depressions on the surface of a model.

To increase the impression of realism, a bump or normal map would be added to more accurately recreate the coarse, grainy surface of, for instance, a brick, and heighten the illusion that the cracks between bricks are actually receding in space. Of course, it would be possible to achieve the same effect by modeling each and every brick by hand, but a normal mapped plane is much more computationally efficient. Normal mapping is a significant process incorporated in the development of modern computer games.

Bump, displacement, and normal maps are a discussion in their own right, and are absolutely essential for achieving photo-realism in a render.



Figure 15: Taurus pt 92 textured 3d model. Rendered in marmoset engine(real time game engine).

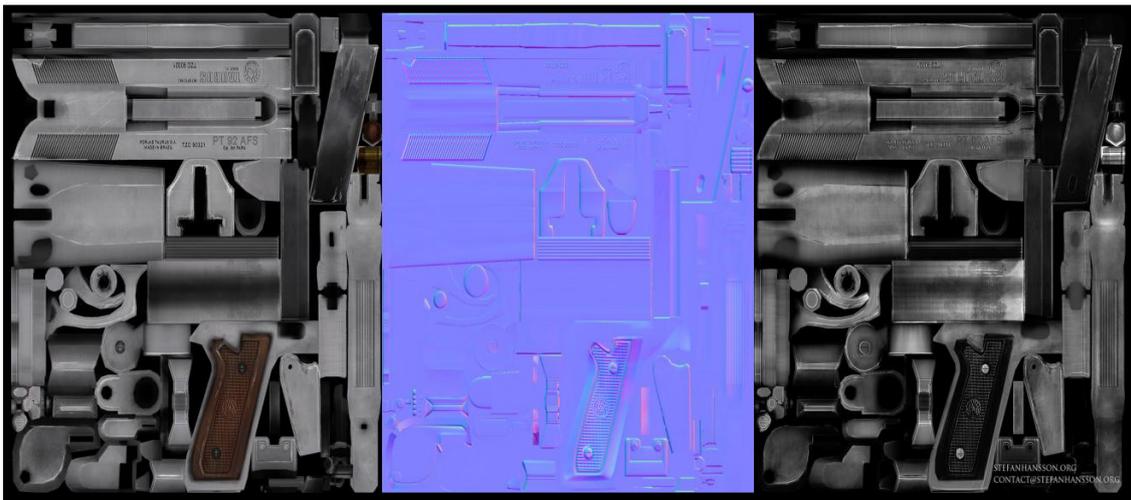


Figure 16: Diffuse, normal and specular map of the above 3d model.

Aside from these three map types, there are one or two others you'll see relatively often:

Reflection Map: It the software which portions of the 3D model should be reflective. If a model's entire surface is reflective or if the level of reflectivity is uniform a reflection map is usually omitted. Reflection maps are grayscale images, with black indicating 0% reflectivity and pure white indicating a 100% reflective surface.

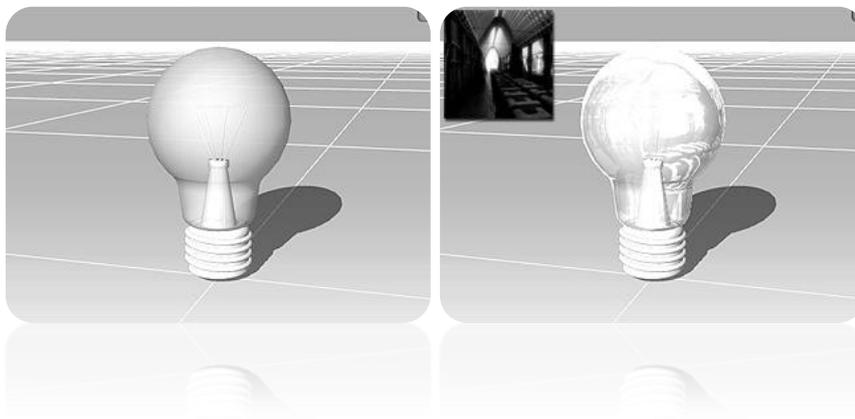


Figure 17: Mesh without any texture (left image). Reflection image projected onto the object (right image).

Transparency (or Opacity) Map: Exactly like a reflection map, except it tells the software which portions of the model should be transparent. A common use for a transparency map would be a surface that would otherwise be very difficult, or too computationally expensive to duplicate, like a chain link fence. Using a transparency, instead of modeling the links individually can be quite convincing as long as the model doesn't feature too close to the foreground, and uses far fewer polygons.

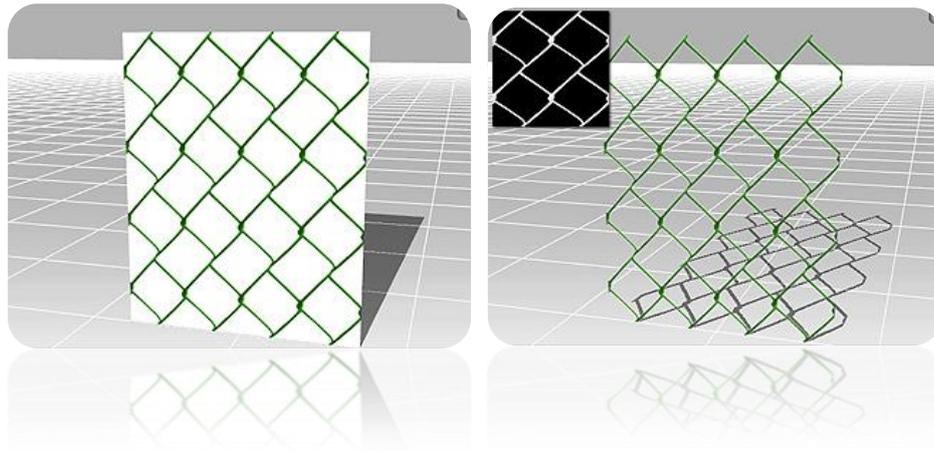


Figure 18: Mesh with diffuse map only (left image). Opacity texture applied on the mesh (right image).

Texture maps are crucial for the design of the virtual scene. They facilitate the reduction of the polygonal complexity of the 3d models used, which in any other way would hinder rendering performance. In particular, the normal maps of some low polygon count models used in our scene were acquired from their high polygon versions to keep all the fine details of the models, thus maintaining an acceptable lighting quality with low computational cost. An example of this method can be seen on the following picture. Another use of texture maps, are opacity maps which allow for the otherwise opaque window in our scene to become transparent.

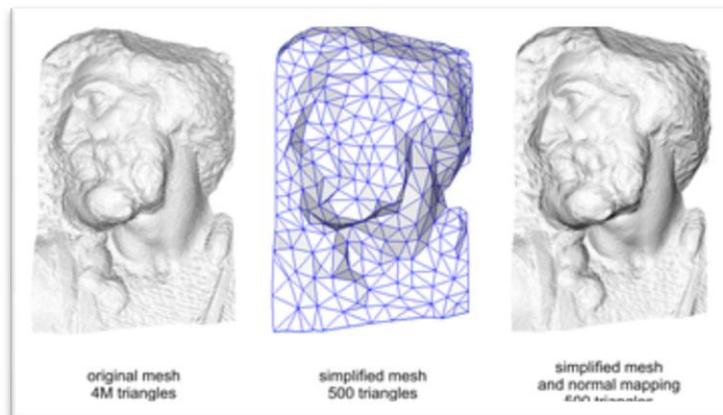


Figure 19: Normal mapping used to re-detail simplified meshes.

2.2 Virtual Reality Technology and Game Engines

There are several game engines offering a wide range of tools to create interactive photorealistic environments, to be used for the creation of a computer game. For the processes of this work, although the final result would not be a computer game, since it would not be recreational, a game engine could offer the required tools to create an interactive photorealistic environment.

A game engine provides the framework and the Application User Interface (API) for the developer to use and communicate with the hardware. It consists of separate autonomous system, each handling a specific process, e.g. graphics system, sound system, physics system, etc. Figure 20 shows the common architecture of game engines.

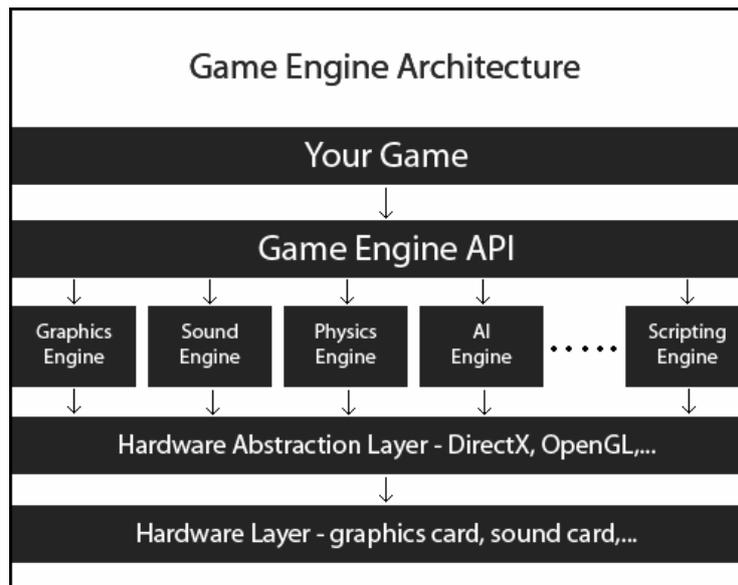


Figure 20: Architecture of a game engine.

For this work, a game engine was decided to be used in order to create interactive realistic virtual scenes. In this chapter, the game engines that were considered to be used will be presented, as well as UDK, which is the game engine that was preferred.

2.2.1 Unity 3D

Unity 3D is a fully featured application suite, providing tools to create 3D games or other applications, e.g. architectural visualization. It provides support for editing object geometry, surfaces, lights and sounds. It uses the Ageia physics engine, provided by nVidia. A light mapping system, called Beast, is included.

In terms of programming, Unity 3D supports 3 programming languages: JavaScript, C# and Boo, which is a python variation. All 3 languages are fast and can be interconnected. The game’s logic runs in the open-source platform “Mono”, offering speed and flexibility. For the development process, a debugger is also included, allowing pausing the game at any time and resuming it step-by-step.

Unity 3D is widely used, with a large community offering help. It is free for non-commercial use and is targeted to all platforms, such as PC, MAC, Android, iOS and web.

2.2.2 Torque 3D

Torque 3D is a game engine with a lot of features for creating networking games. It includes advanced rendering technology, a Graphical User Interface (GUI) building tool and a World Editor, providing an entire suit of WYSIWYG (What-You-See-Is-What-You-Get) tools to create the game or simulation application.

The main disadvantage of Torque 3D is that it is not free, but it needs to be licensed for \$100. The programming language used is “TorqueScript”, which resembles C/C++. It is targeted for both Windows and MacOS platforms, as well as the web.

2.2.3 Unreal Engine 3 – Unreal Development Kit (UDK)

Unreal Development Kit (UDK) is one of the leading game engines currently. It became free on November 2009 for non-commercial use and is used from the world's largest development studios. The UDK community includes thousands of people from around the world, providing help and advice.

UDK's core is written in C++, making it very fast. It offers the ability to use both "UnrealScript", UDK's object-oriented scripting language, and C/C++ programming languages. It provides many different tools for the creation and the rendering of a virtual scene.

2.2.4 Unreal Engine 3 – Overview

The main elements of Unreal Engine 3 which were central to the development of the experimental framework presented in this project report are the following:

- The Unreal Lightmass component of UDK which is a global illumination solver. Provides high-quality static lighting with next-generation effects, such as soft shadows with accurate penumbras, diffuse, specular inter-reflection and color bleeding;
- The ability to add fracture effects to static meshes to simulate destructible environments;
- Soft body dynamics (physics);
- Large crowd simulation;
- **Free license** for non-commercial use.

The selection of Unreal game engine 3 was based on the above specifications and the sophistication of lighting simulation of the Unreal Lightmass feature. The next chapter is describing the characteristics of Unreal Lightmass in detail.

2.3 Immersive, Virtual Reality Systems

Immersive systems are high tech, three dimension display systems that allow users to be "immersed" into a displayed image. In an immersive environment, images are often displayed in stereoscopic 3D. Tracking systems can also be utilized, enabling a user to move all around these 3D images and even interact with them. The result is an experience that very much looks and feels like it is "real."

Staffan Björk and Jussi Holopainen, in Patterns in Game Design 2004, divide immersion into four categories: sensory-motoric immersion, cognitive immersion, emotional immersion and **spatial immersion**. The last one tends to be the most suitable for the purposes of this project. Spatial immersion occurs when a player feels the simulated world is perceptually convincing. The player feels that he or she is really "there" and that a simulated world looks and feels "real".



Figure 21: Classic Virtual reality HMD.

2.3.1 Stereoscopy

Stereoscopy (also called **stereoscopies** or **3D imaging**) 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.

There are two categories of 3D viewer technology, active and passive. Some of the most well known stereoscopic vision methods are shown below.

Active Shutter Systems: A Shutter system works by openly presenting the image intended for the left eye while blocking the right eye's view, then presenting the right-eye image while blocking the left eye, and repeating this so rapidly that the interruptions do not interfere with the perceived fusion of the two images into a single 3D image. It generally uses liquid crystal shutter glasses. Each eye's glass contains a liquid crystal layer which has the property of becoming dark when voltage is applied, being otherwise transparent. The glasses are controlled by a timing signal that allows the glasses to alternately darken over one eye, and then the other, in synchronization with the refresh rate of the screen.



Figure 22: To view the stereoscopic image on the left a pair of shutter glasses as showed on the second image is needed.

Passive Polarization Systems: To present stereoscopic pictures, two images are projected superimposed onto the same screen through polarizing filters or presented on a display with polarized filters. For projection, a silver screen is used so that polarization is preserved. The viewer wears low-cost eyeglasses which also contain a pair of opposite polarizing filters. As

each filter only passes light which is similarly polarized and blocks the opposite polarized light, each eye only sees one of the images, and the effect is achieved.

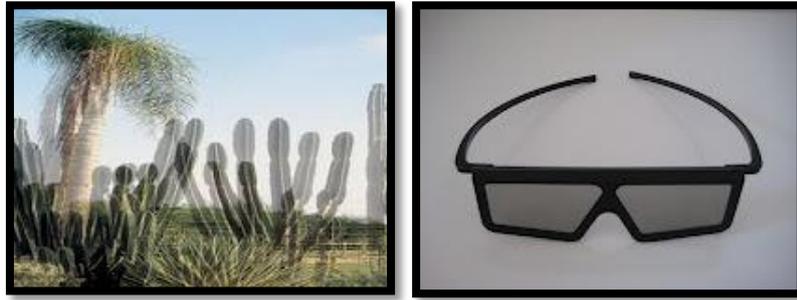


Figure 23: Polarized 3D image (on the left). Polarized 3D Glasses (on the right).

Color Anaglyph Systems: Anaglyph 3D is the name given to the stereoscopic 3D effect achieved by means of encoding each eye's image using filters of different (usually chromatically opposite) colors, typically red and cyan. Anaglyph 3D images contain two differently filtered colored images, one for each eye. When viewed through the "color coded" "anaglyph glasses", each of the two images reaches one eye, revealing an integrated stereoscopic image. The visual cortex of the brain fuses this into perception of a three dimensional scene or composition.



Figure 24: Anaglyph 3D photograph (on the left). Anaglyph red/cyan 3D Glasses (on the right).

Full or Partial Overlap Systems: Traditional stereoscopic photography consists of creating a 3D illusion starting from a pair of 2D images, a stereogram. These systems to enhance depth perception in the brain are providing to the eyes of the viewer with two different images, representing two perspectives of the same object, with a minor deviation exactly equal to the perspectives that both eyes naturally receive in binocular vision. The viewer for such systems is commonly a HMD thus this technology is mostly used for research purposes.

In full overlap each eye is viewing exactly the same identical image. Therefore in partial overlap systems each eye is viewing a percentage of the image in common and the other is unique for both eyes. In addition the angle of screen that each eye is viewing is rotated horizontal by some degrees. This technique achieves to enlarge the field of view of the user.

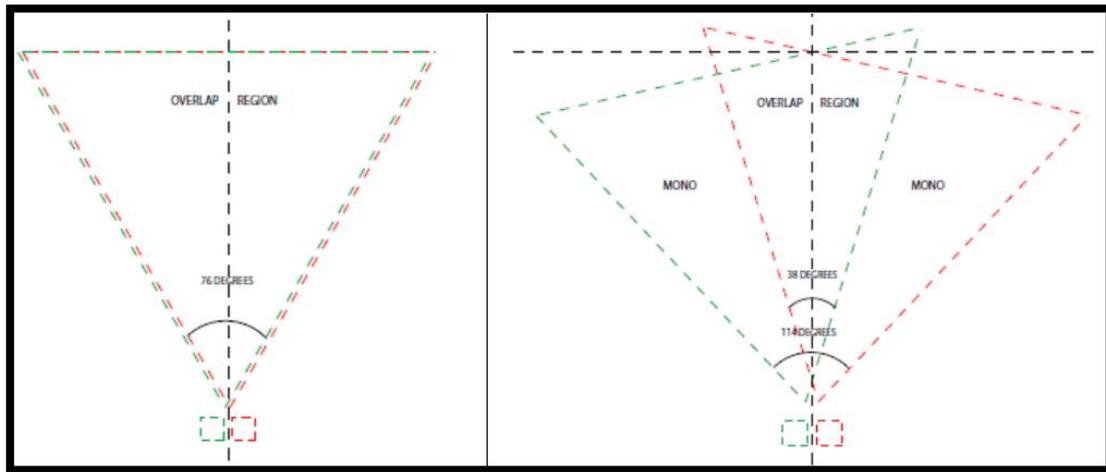


Figure 25: To the left Binocular Image with Full Overlap, to the right Binocular Image with Partial Overlap.

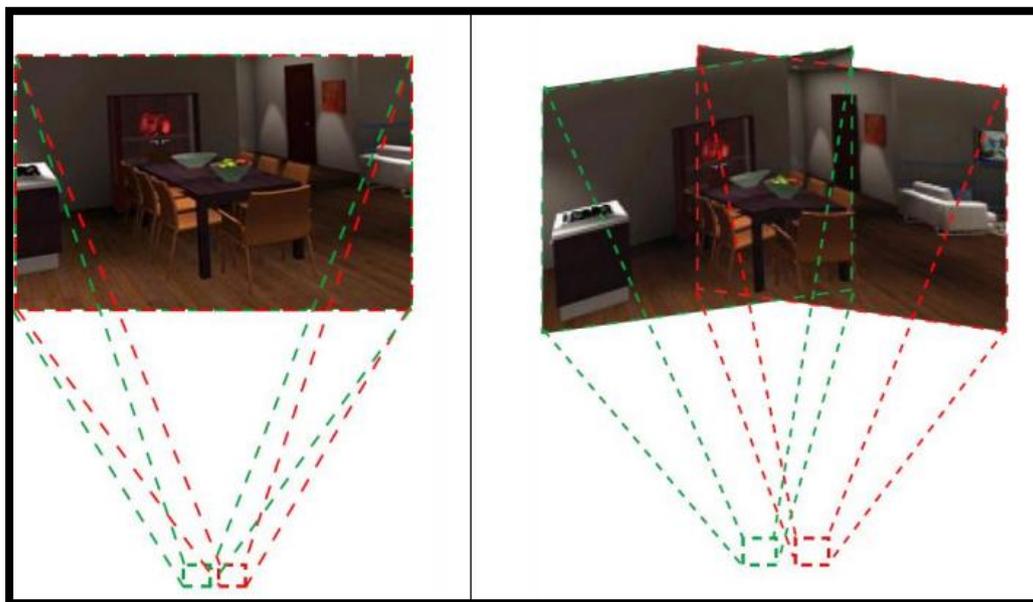


Figure 26: To the left Full Overlap view, to the right Partial Overlap view.

2.3.2 Head Mounted Displays Device

A head-mounted display (HMD), is a display device, worn on the head or as part of a helmet, that has a small display optic in front of one (monocular HMD) or each eye (binocular HMD). A typical HMD has either one or two small displays with lenses and semi-transparent mirrors embedded in a helmet, eye-glasses (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.



Figure 27: nVisor SX111 Head Mounted Display (on the left), head-tacker InertiaCube3 (on the right).

The Head Mounted Display (HMD) used in these experiments is the NVIS SX111 (Figure 28). One of its significant characteristics is the wide field-of-view of a total 102° for both eyes. A 3-degrees-of freedom (rotational) head-tacker was attached to this HMD acquiring the user's head rotational direction (Figure 28). The HMD makes use of partial overlap stereoscopic method in order to produce stereoscopic vision. This method will be discussed in the Implementation - Chapter 5 .



Figure 28: A participant wearing the head mounted display model nVisor SX111 during the experiment.

2.4 Eye-tracking

Eye tracking is the process of measuring either the point of gaze ("where we are looking") or the motion of an eye relative to the head. An **eye tracker** is a device for measuring eye positions and eye movement. The HMD employed in this research included embedded binocular eye tracking by Arrington Research. Eye trackers are also 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 electrooculogram.

2.4.1 Eye-tacking methods

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.

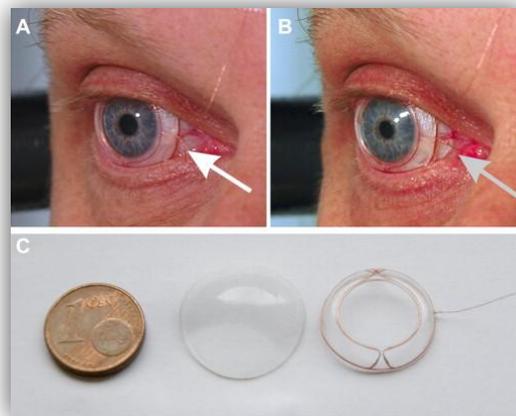


Figure 29: A subject wearing a scleral search coil in conventional manner (A), the wire exiting directly (white arrow). When wearing a bandage lens on top of the coil, the wire is applied to the sclera and exits the eye near the medial canthus (B, gray arrow). The bandage lens (C, centre) is 5 mm larger in diameter than a scleral search coil (C, right).

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.



Figure 30: A subject wearing a headset attached by two optical sensors, those within the red circles above.

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 centre 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 discretisation 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.

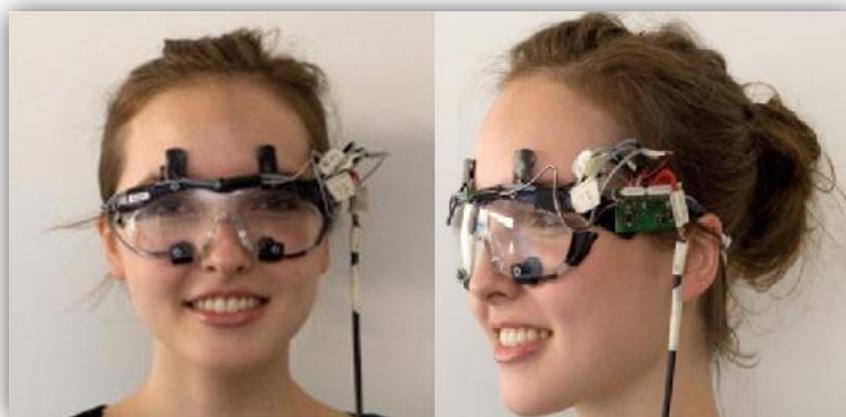


Figure 31: A woman wearing EOG goggles.

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

2.4.2 Vision types

There are two kinds of vision in almost all species in the planet, e.g. Binocular and Monocular vision.

Binocular vision is vision in which both eyes are used together. Having two eyes confer has at least four advantages over having one. First, it gives a creature a spare eye in case one is damaged. Second, it gives a wider field of view. For example, humans have a maximum horizontal field of view of approximately 200 degrees with two eyes, approximately 120 degrees of which makes up the binocular field of view (seen by both eyes) flanked by two unocular fields (seen by only one eye) of approximately 40 degrees. Third, it gives binocular summation in which the ability to detect faint objects is enhanced. Fourth, it can give stereopsis in which parallax provided by the two eyes' different positions on the head give precise depth perception. Such binocular vision is usually accompanied by singleness of vision or binocular fusion, in which a single image is seen despite each eye having its own image of any object. Stereopsis is the impression of depth that is perceived when a scene is viewed with both eyes by someone with normal binocular vision. Binocular viewing of a scene creates two slightly different images of the scene in the two eyes due to the eyes' different positions on the head. These differences, referred to as binocular disparity, provide information that the brain can use to calculate depth in the visual scene, providing a major means of depth perception. The term stereopsis is often used as short hand for 'binocular vision', 'binocular depth perception' or 'stereoscopic depth perception', though strictly speaking, the impression of depth associated with stereopsis can also be obtained under other conditions, such as when an observer views a scene with only one eye while moving. Observer motion creates differences in the single retinal image over time similar to binocular disparity; this is referred to as motion parallax. Importantly, stereopsis is not usually present when viewing a scene with one eye, when viewing a picture of a scene with either eyes, or when someone with abnormal binocular vision (strabismus) views a scene with both eyes. This is despite the fact that in all these three cases humans can still perceive depth relations.

Monocular vision is vision in which each eye is used separately. By using the eyes in this way, as opposed by binocular vision, the field of view is increased, while depth perception is limited. The eyes are usually positioned on opposite sides of the animal's head giving it the ability to see two objects at once. Monocular vision implies that only one eye is receiving optical information, the other one is closed.

2.4.3 Eye dominance

Eye dominance or eyedness, is the tendency to prefer visual input from one eye to the other. It is somewhat analogous to the laterality of right or left handedness; however, the side of the dominant eye and the dominant hand do not always match. This is because both hemispheres control both eyes, but each one takes charge of a different half of the field of view, and therefore a different half of both retinas. Therefore there is no direct analogy between "handedness" and "eyedness" as lateral phenomena.

There exist some very simple tests to determine which eye is dominant such as the Miles, Porta, Ring and others. The Miles test was used to determine participant's eye dominance in this thesis experiments. For this test each participant extends both arms in front

of his body and places the hands together so as to make a small triangle between his thumbs and forefingers. With both of his eyes open, the participant looks through the triangle and focus on a specific small object. He then moves his hands slowly to his face, trying to keep sight of the object through the opening; the opening will naturally lay over his dominant eye.



Figure 32: Eye dominance test sketch.

2.5 Memory Schemata and Awareness States

While previous background knowledge in this chapter introduces the basic principles of computer graphics and provides technical information relating to the complexities of IVE generation, this section provides background information regarding the memory schema and awareness states theory employed in this project. The effect of gender on spatial abilities and spatial performance inspired numerous studies carried out in real and Virtual Environments (VE). VEs are now becoming an increasingly popular alternative approach for research exploration of gender differences in spatial navigation. Moreover, simulation fidelity is based on simulation of spatial awareness as in the real world. This project presents a 3D interactive experimental framework exploring the effect of gender differences in spatial navigation, memory performance and spatial awareness in a complex Immersive Virtual Environment (IVE).

2.5.1 Memory and Perception

Human Memory is a system for storing and retrieving information acquired through our senses [Baddeley 1997, Riesberg 1997]. The briefest memory store lasts for only a fraction of a second. Such sensory memories are perhaps best considered as an integral part of the process of perceiving. Both vision and hearing, for instance, appear to have a temporary storage stage, which could be termed short-term auditory or visual memory and that could last for a few seconds. In addition to these, though, humans clearly retain long-term memory for sights and sounds. Similar systems exist in the case of other senses such as smell, taste and touch. In this section, the memory awareness methodology and memory schema theories employed in this project are analyzed.

2.5.2 The Remember/Know Paradigm

In this section, the main methodology employed in this thesis is going to be analyzed. This methodology forms the core of the experimental design presented in Chapter 4.

In the process of acquiring a new knowledge domain, visual or non-visual, information retained is open to a number of different states. Accurate recognition memory can be supported by: a specific recollection of a mental image or prior experience (remembering); reliance on a general sense of knowing with little or no recollection of the source of this sense (knowing); strong familiarity rather than a un-informed guess (familiar); and guesses. ‘Remembering’ has been further defined as ‘personal experiences of the past’ that are recreated mentally [Gardiner and Richardson-Klavehn 1997]. Meanwhile ‘knowing’ refers to ‘other experiences of the past but without the sense of reliving it mentally’. Tulving [Tul92] provided the first demonstration that these responses can be made in a memory test, item by item out of a set of memory recall questions, to report awareness states as well. He reported illustrative experiments in which participants were instructed to report their states of awareness at the time they recalled or recognized words they had previously encountered in a study list. If they remembered what they experienced at the time they encountered the word, they made a ‘remember’ response. If they were aware they had encountered the word in the study list but did not remember anything they experienced at that time, they expressed a ‘know’ response. The results indicated that participants could quite easily distinguish between experiences of remembering and knowing. These distinctions provide researchers a unique window into the different subjective experiences an individual has of their memories

Measures of the accuracy of memory can therefore be enhanced by self-report of states of awareness such as ‘remember’, ‘know’, ‘familiar’ and ‘guess’ during recognition [Conway et al. 1997; Brandt et al. 2006]. Object recognition studies in VE simulations have demonstrated that low interaction fidelity interfaces, such as the use of a mouse compared to head tracking, as well as low visual fidelity, such as flat-shaded rendering compared to radiosity rendering, resulted in a higher proportion of correct memories that are associated with those vivid visual experiences of a ‘remember’ awareness state [Mania et al. 2003; 2006; 2010]. As a result of these studies, a tentative claim was made that those immersive environments that are distinctive because of their variation from ‘real’ representing low interaction or visual fidelity recruit more attentional resources. This additional attentional processing may bring about a change in participants’ subjective experiences of ‘remembering’ when they later recall the environment, leading to more vivid mental experiences. The present research builds upon this pattern of results and its possible explanations.

Whilst researchers may be interested in measuring differences between the memorial experiences of remembering and knowing, there is recent evidence to suggest that how this is implemented in a practical sense can influence the accuracy of our measures of these. Specifically, the instructions and terminology influence the accuracy of participants’ remember-know judgments (McCabe & Geraci, 2009). In the past, there have been concerns raised about the use of the terms ‘remember’ and ‘know’ because the meaning that participants attach to these terms may be slightly different to those intended by the researchers. In clinical populations this has been a particular concern, and several researchers

have replaced the terms ‘remember’ and ‘know’ with those of ‘type a’ and ‘type b’ (e.g. Levine et al., 1998; Wheeler & Stuss, 2003). Recent evidence has suggested that these changes are also beneficial when measuring ‘remember’ and ‘know’ judgments in non-clinical populations (McCabe & Geraci, 2009). Participants are generally more accurate, in that there are less false-alarms, when ‘remember’ and ‘know’ are replaced with the terms ‘type a’ and ‘type b’ in any instructions given. This procedure was therefore followed here.

2.5.3 Memory Schemata

This thesis uses an approach based on classic findings from memory research, in which schemata are used to explain memory processes. Schemata are knowledge structures of cognitive frameworks based on past experience which facilitate the interpretation of events and influence memory retrieval [Minsky 1975, Brewer and Treyns 1981].

It has been shown that memory performance is frequently influenced by context-based expectations (or ‘schemas’) which aid retrieval of information in a memory task [Minsky 1975]. A schema can be defined as a model of the world based on past experience which can be used as a basis of remembering events and provides a framework for retrieving specific facts. In terms of real world scenes, schemas represent the general context of a scene such as ‘office’, ‘theatre’ etc. and facilitates memory for the objects in a given context according to their general association with that schema in place. Previously formed schemas may determine in a new, but similar environment, which objects are looked at and encoded into memory (e.g., fixation time). They also guide the retrieval process and determine what information is to be communicated at output [Brewer and Treyns 1981].

Schema research has generally shown that memory performance is frequently influenced by schema-based expectations and that an activated schema can aid retrieval of information in a memory task [Minsky 1975]. Brewer and Treyns (1981) proposed five different ways in which schemata might influence memory performance:

- By determining which objects are looked at and encoded into memory (e.g. fixation time).
- By providing a framework for new information.
- By providing information which may be integrated with new episodic information.
- By aiding the retrieval process.
- By determining which information is to be communicated during recall.

[Pichet’s & Anderson’s 1966] schema model predicts better memory performance for schema consistent items, e.g. items that are likely to be found in a given environment, claiming that inconsistent items are mostly ignored. Contrarily, the dynamic memory model [Hollingworth & Henderson 1998] suggests that schema-inconsistent information for a recently-encountered episodic event will be easily accessible and, therefore, leads to better memory performance. Previous VE experiments revealed that schema consistent elements of VE scenes were more

likely to be recognized than inconsistent information [Mania et al. 2005], supporting the broad theoretical position of [Pichet & Anderson 1966]. Such information has led to the development of a novel selective rendering algorithm [Zotos et al. 2010]. In this experimental framework, scene elements which are expected to be found in a VE scene may be rendered in lower quality, in terms of polygon count thereby reducing computational complexity without affecting object memory [Zotos et al. 2009].

2.5.4 Goal

The utility of certain VEs for training such as flight simulators is predicated upon the accuracy of the spatial representation formed in the VE. Spatial memory tasks, therefore, are often incorporated in benchmarking processes when assessing the fidelity of a VE simulation. Spatial awareness is significant for human performance efficiency of such tasks as they require spatial knowledge of an environment. A central research issue therefore for real-time VE applications for training is how participants mentally represent an interactive computer graphics world and how their recognition and memory of such worlds correspond to real world conditions.

The experimental methodology presented focuses upon exploring the effect of gender (male vs female) on object-location recognition memory and its associated awareness states while immersed in a radiosity-rendered synthetic simulation of a complex scene. The space was populated by objects consistent as well as inconsistent with each zone's context, displayed on a head-tracked, stereo-capable HMD. The main premise of this work is that memory performance is an imperfect reflection of the cognitive activity that underlies performance on memory tasks. A secondary goal was to investigate the effect of varied scene context on object recognition tasks post-VE exposure in relation to eye tracking data.

The technical implementation of this project was based on the development of an 3D interactive experimental platform to be displayed on a stereo-capable high-end HMD with eye tracking capabilities. Technical details as well as experiment data are presented in the following chapters.

3 Chapter 3 – Software Architecture and Development Framework

In this chapter, the software architecture and the framework used in the development process will be described in detail. The various tools used to build several parts of this project’s application, such as the Flash authoring environment and the database system used will also be analyzed.

3.1 Game Engine – Unreal Development Kit (UDK)

For the purposes of this project, the power of building and extending upon a framework was preferred to building from scratch. As already discussed, UDK is a powerful framework used mostly in creating computer games and visualization. Its built-in lighting system was a requisite feature, which addressed the needs for realistic lighting effects of this project.

UDK consists of different parts, making it act both like a game engine and a 3D authoring environment. It provides the necessary tools to import 3D objects, create and assign materials on objects that affect the lighting distribution, precompute lighting effects and import and use sounds and sound effects. It, also, allows the designed application to seemingly attach to Flash User Interfaces (UI). UDK can also be used to render the created virtual environments, as well as create and respond to events while navigating the virtual scenes.

3.1.1 Unreal Editor

The Unreal Editor is the tool inside UDK used to create and edit virtual environments. From within Unreal Editor, 3d objects, sounds, videos, textures and images can be imported to the Content Browser library and inserted in the virtual environment. Also, the Unreal Editor can create and assign materials to 3d objects, as well as alter lighting and rendering configurations.

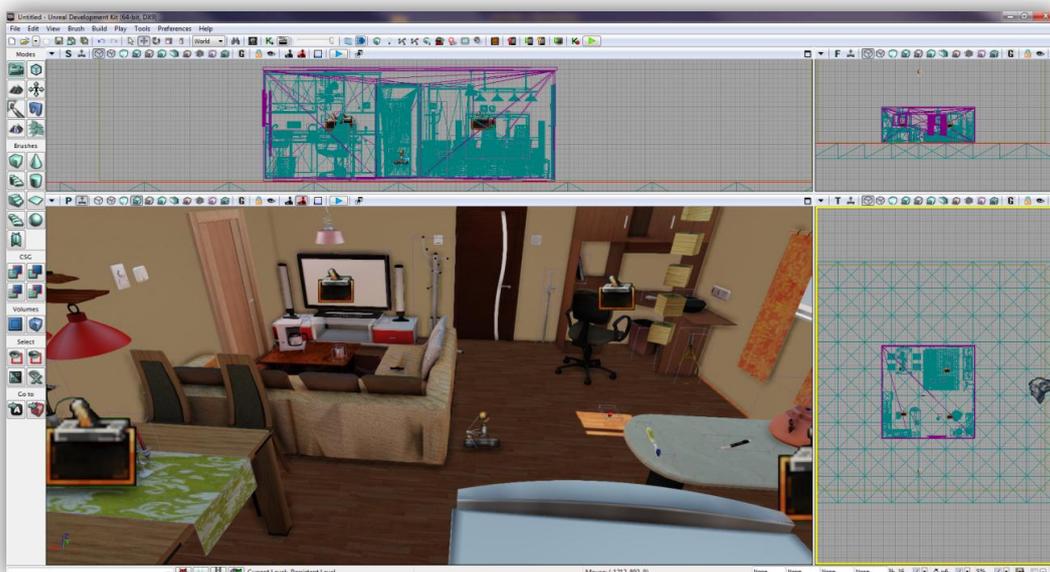


Figure 33: The Unreal Editor with a virtual scene loaded.

Actors, Lights and properties

Everything inside the virtual scene created in the Unreal Editor is considered from UDK to be an “Actor”, from 3d objects to lights. This is in accordance with Unreal Script (see below), which is an Object-Oriented Programming language and every object is assigned to a class that extends from Actor. So, 3d objects are assigned to StaticMeshActor class, lights can be variedly assigned to PointLight, PointLightToggleable, DominantDirectionalLight classes according to their function, sounds are assigned to Sound class, while all these classes extend from the Actor class.

The 3D objects imported into Unreal Editor can be assigned to Static Mesh, used for static objects, or Skeletal Mesh, used for character bodies. All the 3d objects in this project were Static Meshes, as there weren't any characters used. After an object is imported through the Content Browser, we can change its main attributes, like the collision box, materials, light map UVs and polygon count with the Static Mesh Editor. These changes will affect all the instances of this object that will be inserted in the virtual scene, unless they are overridden.

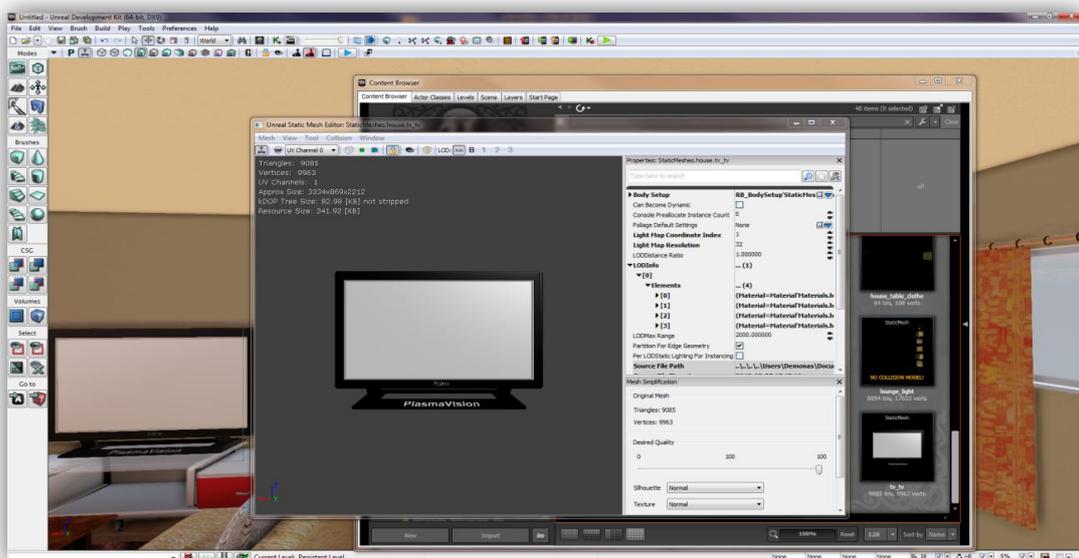


Figure 34: Static Mesh Editor for an imported object.

Once an object is inserted in the editor from the Content Browser library, an instance of its predefined Actor class is created and the editor offers the option to change the configuration of the specific instance, without affecting the other instances. This is true for all kinds of actors loaded in a scene, either being a light or any other possible Actor. The options that can be changed include the object's position, draw scale, properties for the lighting system, materials, collision, components, etc.

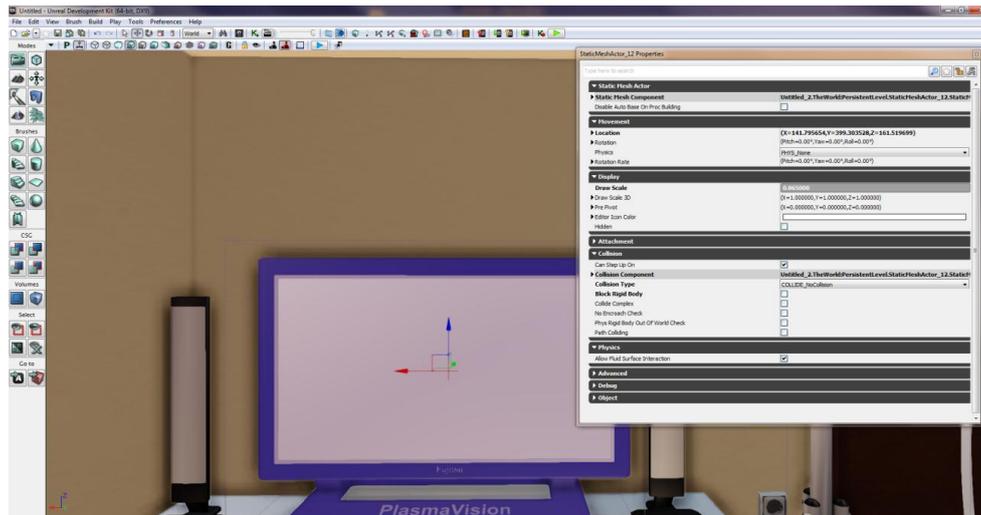


Figure 35: Properties of a Static Mesh Actor instance.

Light Actors

Lights are also considered to be Actors in the Unreal Editor. The type of Actor each light belongs to is different according to the light's function. In this project, only one light source was used, the Dominant Directional Light in order to simulate the sun. Every light actor's properties can be changed as well, like a Static Mesh actor's.

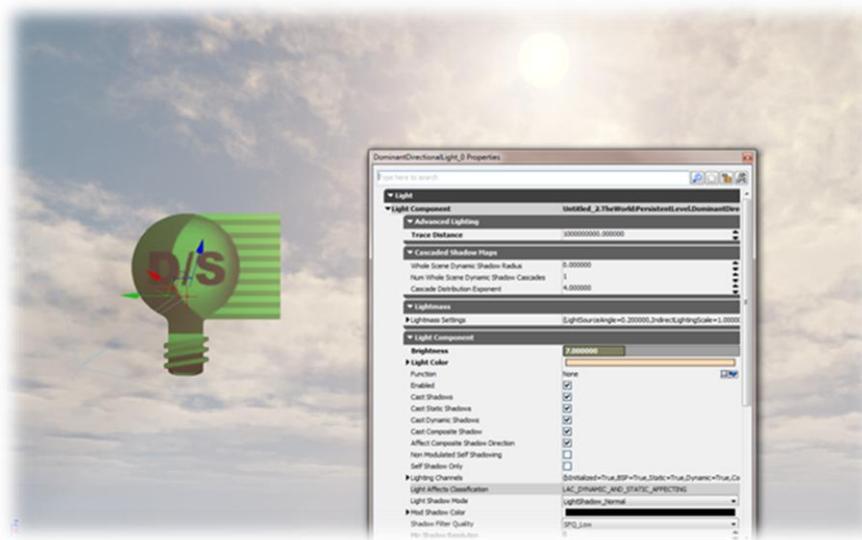


Figure 36: Light Actor properties.

There are many other actor classes that simulate the functions of a variety of other objects, like triggers, sounds, dynamic meshes that move or rotate, etc. All of these can be inserted and manipulated through the Unreal Editor.

Kismet

The Unreal Kismet is a tool inside Unreal Editor and can be described as a graphical system that connects specific events to specific actions. It is node-based and properties of different nodes can be connected with arrows. There are some predefined events and actions, but more can be created through Unreal Script, as described further below.

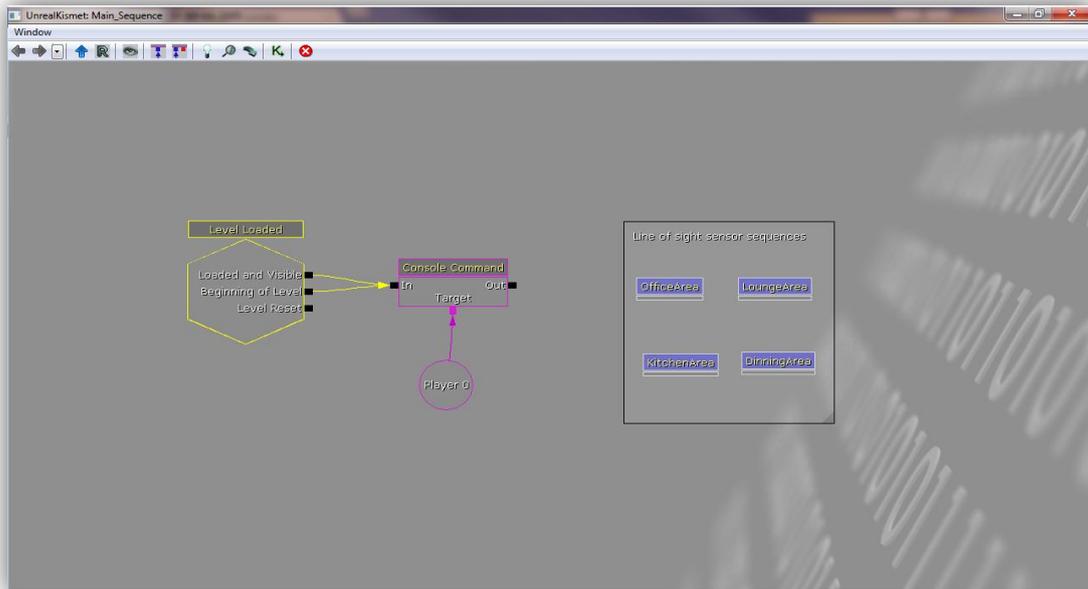


Figure 37: Unreal Kismet set up for testing the experiment scene.

An example can be seen in Figure 37, which depicts an event node which is triggered when a scene is loaded triggering an action event which calls two unreal script functions: *setTestMode true* and *StartExperiment*. As the *setTestMode true* function is called the variable *TestMode* changes its value to *True* in order to display a head up display (HUD) for testing the scene when the experiment starts. Subsequently, the *StartExperiment* function triggers the appropriate actions needed and the experiment is initialized. The box on the right in Figure 37 contains four subsequences representing the four zones of the house and each one is triggered when the user’s visual field is in the bounds of each one. Technical details are included in Chapter 5.

The relevant sequence of events and actions applies only to the currently loaded scene and not to other scenes. Unreal Kismet is not efficient in creating general rules that apply to a complete game or application, in which case Unreal Script is recommended. Unreal Kismet is useful in connecting scene-specific events and actions.

Material Editor

A tool included in the Unreal Editor needed in order to create realistic environments, is the Material Editor. This tool handles the creation and editing of different materials that can be assigned to objects inside the scenes. Materials affect the objects they are assigned to in a variety of ways, mainly in terms of their texture and their interaction with the light.

The Material Editor is node-based, much like the Unreal Kismet, but the nodes here do not represent events or actions, but textures, colors and several processing filters, such as addition of two different textures. The Material Editor provides the main node, which has all the supported properties of the material, such as the diffuse, emissive and specular properties and each property can receive the output from a node or from a sequence of nodes.

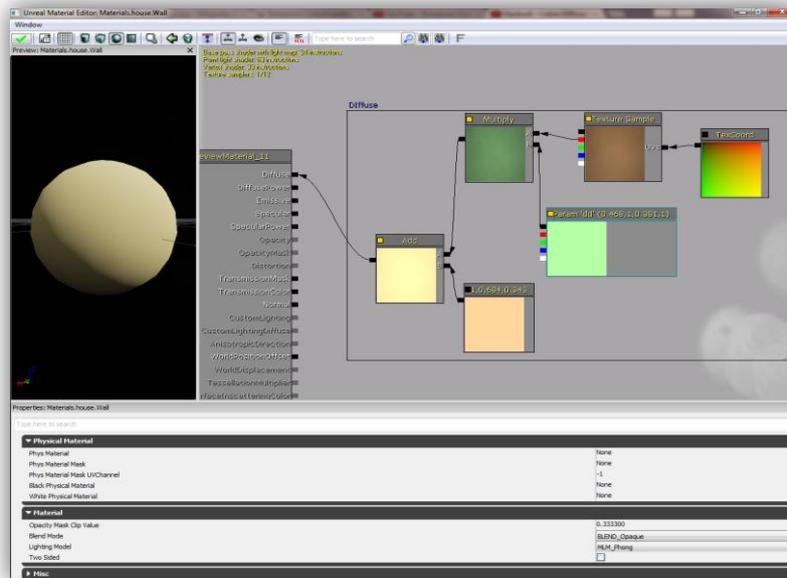


Figure 38: The Material Editor with a sample material.

3.1.2 Sound engine

The Unreal Editor supports its own sound engine and a Sound Editor provides the necessary tools to create various different sound effects. It supports immersive 3D location-based sounds and gives complete control over pitch, levels, looping, filtering, modulation and randomization.

Similar to the rest of the Unreal Editor’s tools, the Sound Editor provides a node-based User Interface to import and use several sound cues, change their properties, mix them together and channel the resulting output to as a new sound effect.



Figure 39: UDK’s Sound Editor and a sound imported into a scene.

3.1.3 DLL File:

UDK provides the option for an UnrealScript class to bind to an external DLL file, by declaring it in the class definition. For example, the following line declares that *ExampleClass* binds to the *ExampleDLL*:

```
class ExampleClass extends GameInfo DLLBind(ExampleDLL);
```

By binding to a DLL file, an UnrealScript class can call the declared in that DLL file methods or functions, which are written in C/C++. This proves to be an easy and efficient way to implement functions that either UDK does not support at all, like I/O operations, or it would slow down the application, due to the fact that UnrealScript is slow.

A function residing inside the DLL must be declared in the UnrealScript class file and then it can be called exactly like it would be if it was an original UnrealScript function. Following the previous example and assuming that the *ExampleDLL* contained a function called *ExampleDLLFunction*, the code inside the UnrealScript class would be:

```
dllimport final function ExampleDLLFunction(); //function declaration
...
ExampleDLLFunction(); //function call
```

3.1.4 Input Manager

The input manager is responsible to handle the communication between the input hardware, such as keyboard, mouse, joystick or button boxes and the application. The input manager examines a configuration file based on *DefaultInput.ini* and according to it binds each input action, such as joystick/mouse movement or key press to a specific method designated to perform the selected action. The Unreal Editor comes with a default configuration file with some predefined bindings between buttons and methods, but this file can be altered to match the needs.

In order to create a new binding between a button press and the performed action, or to change an already defined binding, this change must be reflected in the configuration file. Also, the method defined in the configuration file must exist in the UnrealScript code of the new application being developed.

For the purposes of the experiment the default input settings in the configuration file *DefaultInput.ini* didn't need any variations.

3.1.5 Lighting and Rendering Engine

The Unreal Development Kit comes along with Gemini, a flexible and highly optimized multi-threaded rendering system, which creates a lush visual reality and provides the power necessary for photorealistic simulations. UDK features a 64-bit color HDR rendering pipeline. The gamma-corrected, linear color space renderer provides for immaculate color precision while supporting a wide range of post-processing effects such as motion blur, depth of field, bloom, ambient occlusion and user-defined materials.

UDK supports all modern per-pixel lighting and rendering techniques, including normal mapped, parameterized Phong lighting, custom user-controlled per material lighting models including anisotropic effects, virtual displacement mapping, light attenuation functions, pre-computed shadow masks and directional light maps. UDK provides volumetric environmental effects that integrate seamlessly into any environment. Camera, volume and opaque object interactions are all handled per-pixel. Worlds created with UDK can easily feature multi-layered, global fog height and fog volumes of multiple densities.

It also supports a high-performance texture streaming system. Additionally, UDK's scalability settings ensure that the application will run on a wide range of PC configurations, supporting both Direct3D 9 and Direct3D 11.

The Unreal Development Kit includes the Unreal Lightmass, which is an advanced global illumination solver, built to take full advantage of the rendered. Unreal Lightmass supports the illumination with a single sun, giving off soft shadows and automatically computing the diffuse interreflection (color bleeding). It also offers a variety of options to optimize the illumination solution. It can provide detailed shadows by using directional light mapping with static shadowing and diffuse normal-mapped lighting. An unlimited number of lights can be pre-computed and stored in a single set of texture maps.

3.1.6 Unreal Lightmass

Unreal Lightmass is an advanced global illumination solver. It uses a refined version of the radiosity algorithm, storing the information in each illuminated 3D object's light map, while providing ray-tracing capabilities by supporting Billboard reflections, which allows complex reflections even with static and dynamic shadows with minimal CPU overhead.

Unreal Lightmass is provided as part of the Unreal Development Kit (UDK) and it can only work on scenes created through it. Its performance is dependent on the scenes created and the types of light emitting sources that exist in the scene. It is optimized to increase the renderer's performance.

Its main features include:

Area lights and shadows: With Lightmass, all lights are area lights by default. The shape used by Point and Spot light sources is a sphere, whose radius is set by `LightSourceRadius` under `LightmassSettings`. Directional light sources use a disk, positioned at the edge of the scene. Light source size is one of the two factors controlling shadow softness, as larger light sources will create softer shadows. The other factor is distance from the receiving location to the shadow caster. Area shadows get softer as this distance increases, just like in real life.

Diffuse interreflection: Diffuse Interreflection is by far the most visually important global illumination lighting effect. Light bounces by default with Lightmass, and the Diffuse term of each material controls how much light (and what color) bounces in all directions. This effect is sometimes called color bleeding. It's important to remember that diffuse interreflection is incoming light reflecting equally in all directions, which means that it is not affected by the viewing direction or position.

Mesh Area Lights from Emissive: The emissive input of any material applied to a static object can be used to create mesh area lights. Mesh area lights are similar to point lights, but they can have arbitrary shape and intensity across the surface of the light. Each positive emissive texel emits light in the hemisphere around the texel's normal based on the intensity of that texel. Each neighboring group of emissive texels will be treated as one mesh area light that emits one color.

Translucent shadows: Light passing through a translucent material that is applied to a static shadow casting mesh will lose some energy, resulting in a translucent shadow.

3.1.7 UnrealScript

UnrealScript was designed to provide the developers with a powerful, built-in programming language that maps the needs of game programming. The major design goals of UnrealScript are:

Enabling time, state and network programming, which traditional programming languages do not address but are needed in game programming. C/C++ deals with AI and game logic programming with events which are dependent on aspects of the object's state. This results in long-length code that is hard to maintain and debug. UnrealScript includes native support for time state and network programming which not only simplifies game programming, but also results in low execution time, due to the native code written in C/C++;

Programming simplicity, object-orientation and compile-time error checking, helpful attributes met in Java are also met in UnrealScript. More specifically, deriving from Java UnrealScript offers:

- A pointerless environment with automatic garbage collection;
- A simple single-inheritance class graph;
- Strong compile-time type checking;
- A safe client-side execution "sandbox";
- The familiar look and feel of C/C++/Java code.

Often design trade-offs had to be made, choosing between execution speed and development simplicity. Execution speed was then sacrificed, since all the native code in UnrealScript is written in C/C++ where performance outweighs the added complexity. UnrealScript has very slow execution speed compared to C, but since a large portion of the engine's native code is in C, only the 10%-20% of code in UnrealScript that is executed when called, has low performance.

The Unreal Virtual Machine

The Unreal Virtual Machine consists of several components: The server, the client, the rendering engine, and the engine's support code.

The Unreal server controls all the gameplay and interaction between players and actors (placeable entities). A listen server is able to host both a game and a client on the same computer, whereas the dedicated server allows a host to run on the computer with no client. All players connect to this machine and are considered clients.

The gameplay takes place inside a level, containing geometry actors and players. Many levels can be running simultaneously, each being independent and shielded from the other. This helps in cases where pre-rendered levels need to be fast-loaded one after another. Every actor on a map can be either player-controlled or script-controlled. The script controls the actor's movement, behavior and interaction with other actors. Actor's control can change in game from player to script and vice versa.

Time management is done by dividing each time second of gameplay into Ticks. Each tick is only limited by CPU power, and typically lasts 1/100th of a second. Functions that

manage time are really helpful for gameplay design. Latent functions like Sleep, MoveTo and more cannot be called from within a function, but only within a state.

When latent functions are executing in an actor, the actor's state execution does not continue until the latent functions is completed. However, other actors may call functions from the actor that handles the latent function. The result is that all functions can be called, even with latent functions pending.

In UnrealScript, every actor is as if executed on its own thread. Windows threads are not efficient in handling thousands at once, so UnrealScript simulates threads instead. This means that 100 spawned actors will be executed independently of each other each Tick.

Object Hierarchy

UnrealScript is purely object-oriented and comprises of a well-defined object model with support for high level object-oriented concepts such as serialization and polymorphism. This design differs from the monolithical one that classic games adopted having their major functionality hardcoded and being non-expandable at the object level. Before working with UnrealScript, understanding the object's hierarchy within Unreal is crucial in relation to the programming part.

The main gain from this design type is that object types can be added to Unreal at runtime. This form of extensibility is extremely powerful, as it encourages the Unreal community to create Unreal enhancements that all interoperate. The five main classes one should start with are *Object*, *Actor*, *Pawn*, *Controller* and *Info*.

Object is the parent class of all objects in Unreal. All of the functions in the Object class are accessible everywhere, because everything derives from Object. Object is an abstract base class, in that it doesn't do anything useful. All functionality is provided by subclasses, such as Texture (a texture map), TextBuffer (a chunk of text), and Class (which describes the class of other objects).

Actor (extends Object) is the parent class of all standalone game objects in Unreal. The Actor class contains all of the functionality needed for an actor to be placed inside a scene, move around, interact with other actors, affect the environment, and complete other useful game-related actions.

Pawn (extends Actor) is the parent class of all creatures and players in Unreal which are capable of high-level AI and player controls.

Controller (extends Actor) is the class that defines the logic of the pawn. If pawn resembles the body, Player is the brain commanding the body. Timers and executable functions can be called from this type of class.

Info (extends Actor) is the class that sets the rules of gameplay. Players joining will be handled in this class, which decides which Pawn will be created for the player in the scene and which controller will handle the behavior of the pawn.

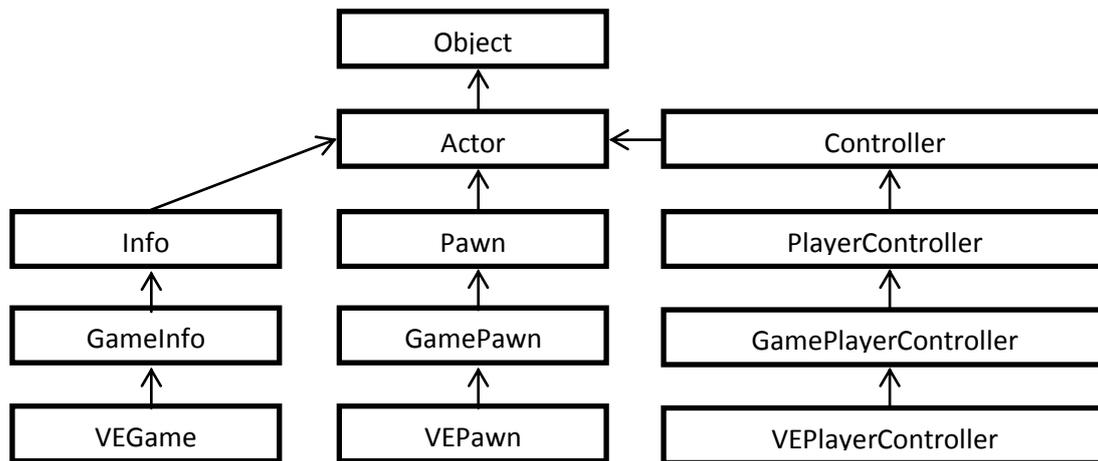


Figure 40: Class Hierarchy Diagram for 3 user-created classes: VEGame , VEPawn and VEPlayerController.

In the example shown above, in Figure 40, the class hierarchy for the three most important classes that control the application flow is depicted. As already described, the *VEGame* class decides how new players entering the scene are treated. The *VEPawn* class describes the properties and the behavior of a Pawn entering a scene that is handled by *VEGame*. Finally, the *VEPlayerController* class handles the *VEPawn* in the scene, according to user input.

Timers

Timers are a mechanism used for scheduling an event to occur. Time management is important both for gameplay issues and for programming tricks. All Actors can have more than one timers implemented as an array of structs. The native code involving timers is written in C++, so using many timers per tick is safe, unless hundreds expire simultaneously causing the execution UnrealScript code.

The following function starts a timer counting the time that the participant has available to navigate the scene and then database is closed properly and returns back to main menu.

```
SetTimer (StageDuration,false,'EndExperiment');
```

This line of code defines that after *StageDuration* – a constant value –seconds, the function ““EndExperiment”” should be called. The false value passed as an argument means that this timer should not repeat the counting, it will only work once.

States

States are known from Hardware engineering, where it is common to see finite state machines managing the behaviour of a complex object. The same management is needed in game programming, allowing each actor to behave differently, according to its state. The usual case to implement states in C/C++ is to include many switch cases based on the object's state. This method, however, is not efficient, since most applications require many states, resulting to difficulties in developing and maintaining the application.

UnrealScript supports states at the language level. Each actor can include many different states, and only one can be active. The state the actor is in reflects the actions it

wants to perform. Attacking, Wandering, Dying are some states the pawns have. Each state can have several functions, which can be the same as another state's functions. However, only the functions in the active state can be called. For example, if an application dictates that an action should only be performed in a specific stage, then this stage could be encapsulated in a different state that implements the function corresponding to that action differently than other states.

States provide a simple way to write state-specific functions, so that the same function can be handled in different ways, depending on which state the actor is in when the function is called. Within a state, one can write special "state code", using the regular UnrealScript commands plus several special functions known as "latent functions". A latent function is a function that executes slowly (i.e. non-blocking), and may return after a certain amount of "game time" has passed. Time-based programming is enabled which is a major benefit that neither C/C++, nor Java offer. Namely, code can be written in the same way it is conceptualized. For example, a script can support the action of "turn the TV on; show video for 2 seconds; turn the TV off". This can be done with simple, linear code, and the Unreal engine takes care of the details of managing the time-based execution of the code.

Interfaces

UnrealEngine3 UnrealScript has support for interface classes that resembles much of the Java implementation. As with other programming languages, interfaces can only contain function declarations and no function bodies. The implementation for these declared methods must be done in the class that actually implements the interface. All function types are allowed and also events. Even delegates can be defined in interfaces.

An interface can only contain declarations which do not affect the memory layout of the class: enums, structs and constants can be declared, but not variables.

Delegates

Delegates are a reference to a function within an instance. Delegates are a combination of two programming concepts, e.g. functions and variables. In a way, delegates are like variables in that they hold a value and can be changed during runtime. In the case of delegates, though, that value is another function declared within a class. Delegates also behave like functions in that they can be executed. It is this combination of variables and functions that makes delegates such a powerful tool under the right circumstances.

UnrealScript Compiler

The UnrealScript compiler is three-pass. Unlike C++, UnrealScript is compiled in three distinct passes. In the first pass, variable, struct, enum, const, state and function declarations are parsed and remembered, e.g. the skeleton of each class is built. In the second pass, the script code is compiled to byte codes. This enables complex script hierarchies with circular dependencies to be completely compiled and linked in two passes, without a separate link phase. The third phase parses and imports default properties for the class using the values specified in the default properties block in the .uc file.

UnrealScript Programming Strategy

UnrealScript is a slow programming language when compared to C/C++. A program in UnrealScript runs about 20x slower than C. However, script programs written are executed only 5-10% of the time with the rest of the 95% being handled in the native code written in C/C++. This means that only the 'interesting' events will be handled in UnrealScript. For example, when writing a projectile script, you typically write a HitWall, Bounce, and Touch function describing what to do when key events happen. Thus, 95% of the time, your projectile script isn't executing any code, and is just waiting for the physics code to notify it of an event. This is inherently very efficient.

The Unreal log may provide useful information while testing scripts. The UnrealScript runtime often generates warnings in the log that notify the programmer of non-fatal problems that may have occurred.

UnrealScript's object-oriented capabilities should be exploited as much as possible. Creating new functionality by overriding existing functions and states leads to clean code that is easy to modify and easy to integrate with other peoples' work. Traditional C techniques should be avoided, like writing a switch statement based on the class of an actor or the state because code like this tends to clutter as new classes and states are added or modified.

3.2 Flash Applications as User Interfaces (UI)

The Unreal Development Kit supports Heads-Up Displays (HUD) that can be constructed in UnrealScript, but in order to create a Graphical User Interface that must be displayed on top of the regular viewport, such as a game which requires user response to it, this is inefficient. For this reason, UDK provides support to integrate a Flash application inside a scene and project it on top of the surface of a 3D object in the scene, or in the center of the screen.

A Flash application can be ideally used as a User Interface in UDK, because it incorporates the ability to display animated graphics, text or buttons on the screen on top of the scene being rendered. It can also receive user input and provide feedback according to it.

A Flash application consists of many frames, placed in the main timeline. The flow of the frames being displayed can be changed through ActionScript - Flash's scripting language, thus allowing to control which frame will be displayed next and when that will happen. Each frame can have its own set of graphics, texts, movie clips and buttons and a script controlling the behavior of the frame's components.

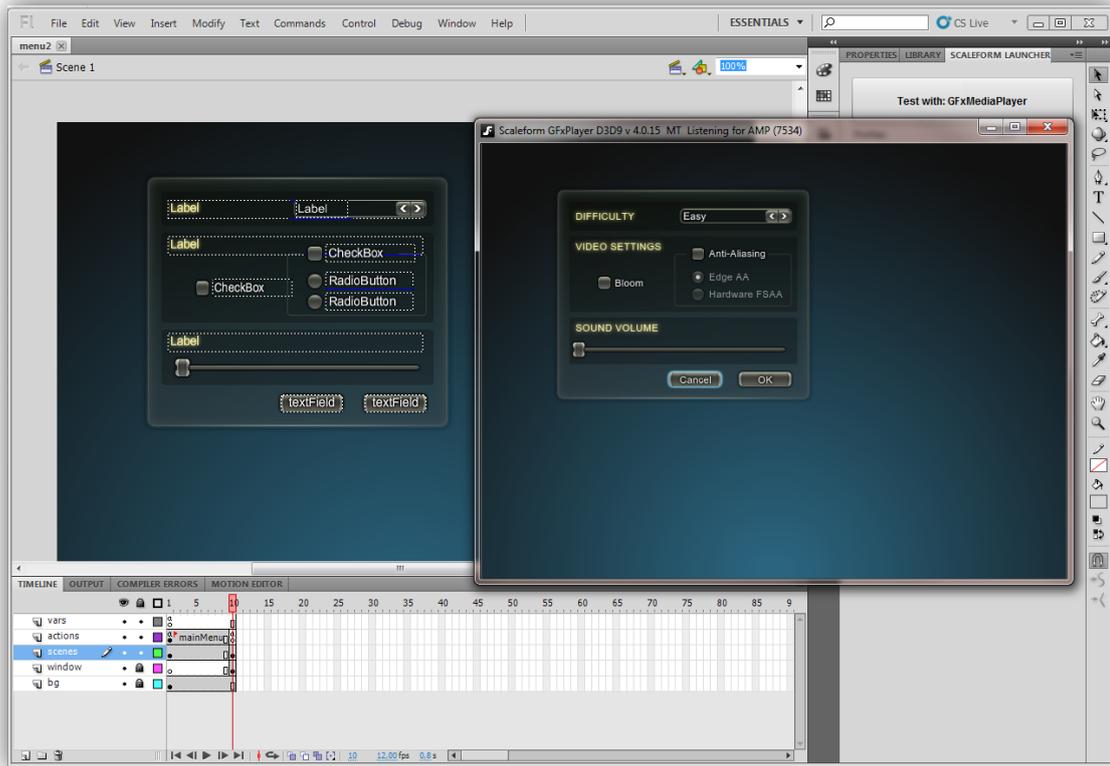


Figure 41: Flash Authoring environment with a Flash User Interface loaded.

3.2.1 Authoring environment for interactive content

In order to create a Flash application, a Flash authoring environment is necessary. There are many different Flash authoring environments available, but the most powerful is Adobe Flash Professional CS5.5. Although it is not free, a 30-day trial version is available for download.

A freshly created Flash application is equipped with an empty stage and an empty timeline. Objects like movie clips, graphics, buttons, text, sounds or other Flash components can be inserted into the application's library, or directly into the scene in the currently selected frame. Various different frames can be created with different components inserted into each frame and the control of the application can be handled through ActionScript.

When the Flash application is fully developed and working, the authoring environment can compile the assets and the ActionScript comprising the application into an executable file in SWF format. Such files can be directly executed by a Flash player and also this is the format that UDK supports.

3.2.2 ActionScript 2.0

Although Flash Professional provides the tools to create applications running in all versions of ActionScript (up to 3.0) and of Flash Player (up to 10.3), UDK currently only supports the integration of Flash applications with ActionScript 2.0 (AS2) and Flash Player 8.

ActionScript is a scripting programming language and it is a dialect of ECMAScript, meaning it has a superset of the syntax and semantics of the more widely known JavaScript. It is suited to the development of Flash applications.

The language itself is open-source in that its specification is offered free of charge and both an open source compiler and open source virtual machine are available. It is often possible to save time by scripting something rather than animating it, which usually also enables a higher level of flexibility when editing.

ActionScript 2.0 primitive data types

The primitive data types supported by ActionScript 2.0 are:

- *String*: A list of characters such as "Hello World"
- *Number*: Any Numeric value
- *Boolean*: A simple binary storage that can only be "true" or "false".
- *Object*: Object is the data type all complex data types inherit from. It allows for the grouping of methods, functions, parameters, and other objects.

ActionScript 2.0 complex data types

There are additional "complex" data types. These are more processor and memory intensive and consist of many "simple" data types. For AS2, some of these data types are:

- *MovieClip* - An ActionScript creation that allows easy usage of visible objects.
- *TextField* - A simple dynamic or input text field. Inherits the MovieClip type.
- *Button* - A simple button with 4 frames (states): Up, Over, Down and Hit. Inherits the MovieClip type.
- *Date* - Allows access to information about a specific point in time.
- *Array* - Allows linear storage of data.
- *XML* - An XML object
- *XMLNode* - An XML node
- *LoadVars* - A Load Variables object allows for the storing and send of HTTP POST and HTTP GET variables
- Sound
- NetStream
- NetConnection
- MovieClipLoader
- EventListener

3.2.3 Connection of the User Interface (UI) with the application

The integration of a Flash application inside a scene in UDK requires that it should first be compiled into an SWF file and imported inside the UDK asset library. Afterwards, either UnrealScript or Unreal Kismet can initiate the Flash application, interact with it, hide it or instruct it to stop playing.

While a Flash application is playing inside a scene, UnrealScript can initiate a call of an ActionScript function and vice versa. This feature allows full interaction between the Flash interface and the application. Consequently, it is easy and efficient to create an application

that initiates a Flash interface whenever it is required and then receive the user’s response and order it to stop playing.

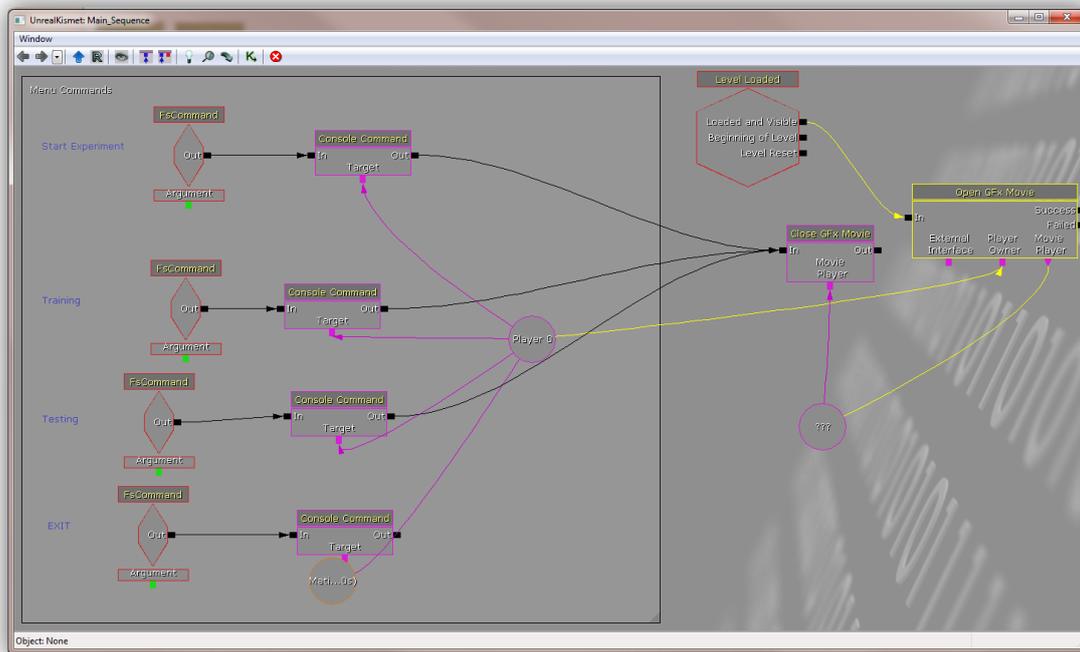


Figure 42: Initiation of a Flash User Interface Application through Unreal Kismet.

An example can be seen in Figure 42, which shows the action “Open Gfx Movie” firing up when it receives the “Level loaded” event. This action performs the required operations to start the Flash Movie that is inserted as an argument in its properties. It can also be set whether this movie can capture user input, as well as where to project this movie, either in the screen or on an Actor’s surface.

Also, Unreal Kismet provides the action “Close Gfx Movie”, which handles the termination of the selected Flash application.

3.3 SQL database system as data storage

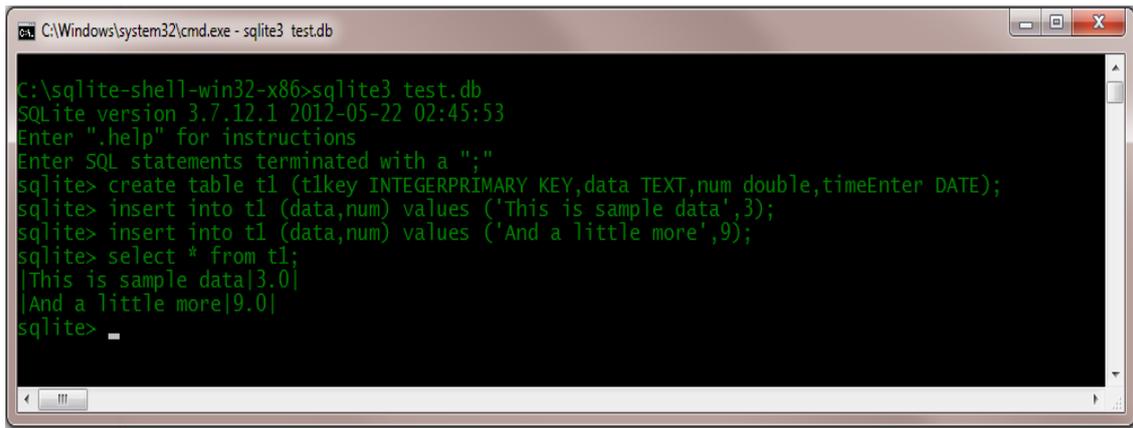
A **database** is an organized collection of data, today typically in digital form. The data are typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies).

The need of using a database system arose when data from eye’s gaze acquired by experimental participants would need processing after the experiment was completed. Keeping the records in a single text file involved two issues. The first one was the abstract format of records making the process of data very difficult and time consuming. The second one was the large numbers of records per participant which made it too difficult for data analysis because each participant’s records had to be extracted separately.

3.3.1 SQLite Database Engine

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The SQLite database engine is free for use for any purpose, commercial or private.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file.



```
C:\Windows\system32\cmd.exe - sqlite3 test.db
C:\sqlite-shell-win32-x86>sqlite3 test.db
SQLite version 3.7.12.1 2012-05-22 02:45:53
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table t1 (t1key INTEGERPRIMARY KEY,data TEXT,num double,timeEnter DATE);
sqlite> insert into t1 (data,num) values ('This is sample data',3);
sqlite> insert into t1 (data,num) values ('And a little more',9);
sqlite> select * from t1;
|This is sample data|3.0|
|And a little more|9.0|
sqlite>
```

Figure 43: Using SQLite shell to make a database.

SQLite is fast, stand-alone, license free and easy setup-configure. In addition, it comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases. Because of the above technical characteristics, SQLite is the perfect choice for use in this project.

3.3.2 SQL Database Schema

A **database schema** of a database system is its structure described in a formal language supported by the database management system (DBMS) and refers to the organization of data to create a blueprint of how a database will be constructed. A database schema is a way to logically group objects such as tables, views, stored procedures etc., e.g. a container of objects. Schemas can be created and altered in a database, and users can be granted access to a schema. A schema can be owned by any user, and schema ownership is transferable.

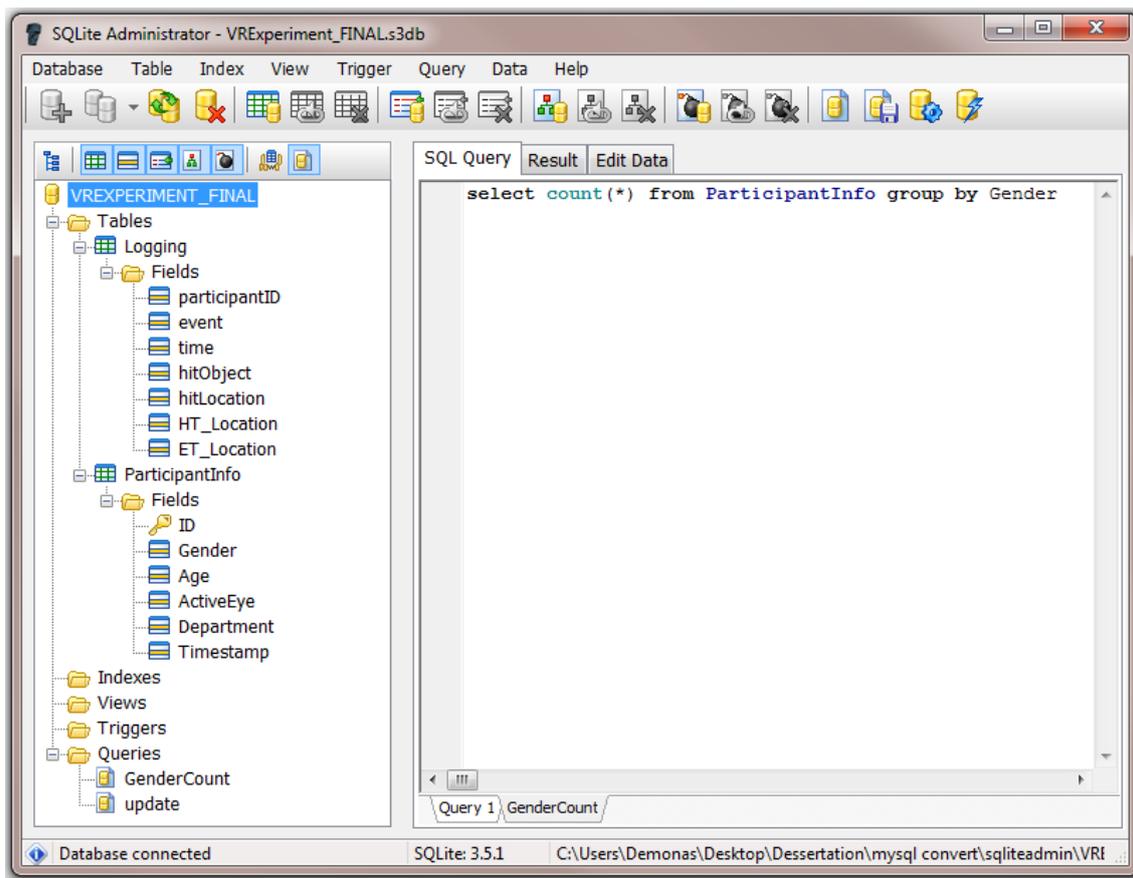


Figure 44: Exploring the database schema “VREXPERIMENT_FINAL”, using the SQLite Administrator tool.

3.3.3 Connection of the Database Engine with the application

In order for Unreal Game Engine to access the database file, the appropriate Driver should be in place. At run time, the game engine dynamically links with the SQLite driver and using the library functions the database is loaded making it accessible within the unreal script code.

The following code creates a new instance of the SQLite Database class which will be the interface to the `SQLiteDB_Driver.dll`. After the instance is created the function `SQLiteDB_Init()` is called.

```
//Create an instance of the SQLite Database class
SQLiteDB = new class 'SQLProject_DLLAPI';

//Load SQLite Database File
SQLiteDB_Init();
```

The call of the following function initializes the SQLiteDriver. After successful initialization of the function, `SQL_loadDatabase ()` is called given the path of the database file to be loaded. If everything was successful, there is access to the database and the appropriate error message is printed on the screen.

```
function SQLiteDB_Init(){
```

```

...
//Load Sqlite driver
SQLiteDB.SQL_initSQLDriver (SQLDrv_SQLite);

//Load SQLite Database File ("VRExperiment.s3db");
if(SQLiteDB.SQL_loadDatabase("SQLite3_Databases\\VRExperiment.s3db")){
...
} else
    `log ("Cannot load database: VRExperiment.s3db");
}

```

The following figure depicts the connection between the Unreal game engine and the SQLite database file through the use of a driver.

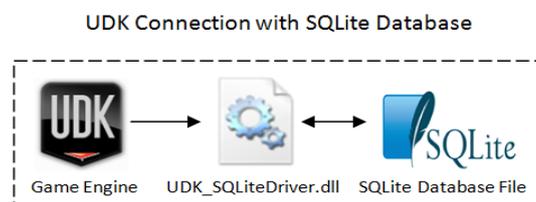


Figure 45: Game engine using a dynamic link library in order to access the database file.

3.4 Eye-tracking device's software

The software used in relation to the eye-tracking device embedded on the HMD is the ViewPoint EyeTracker® of Arrington Research Company which 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 communicating with other applications. It incorporates several methods from which the user can select to optimize the system for a particular application. It provides several methods of mapping position signals extracted from the segmented video image in EyeSpace™ coordinates to the participant's point of regard in GazeSpace™ coordinates.

3.4.1 Overview of the software

An overview of the basic features of the Eye-tracking software will be described. For full documentation please refer on the user manual of the Viewpoint application found on Bibliography-References Chapter.

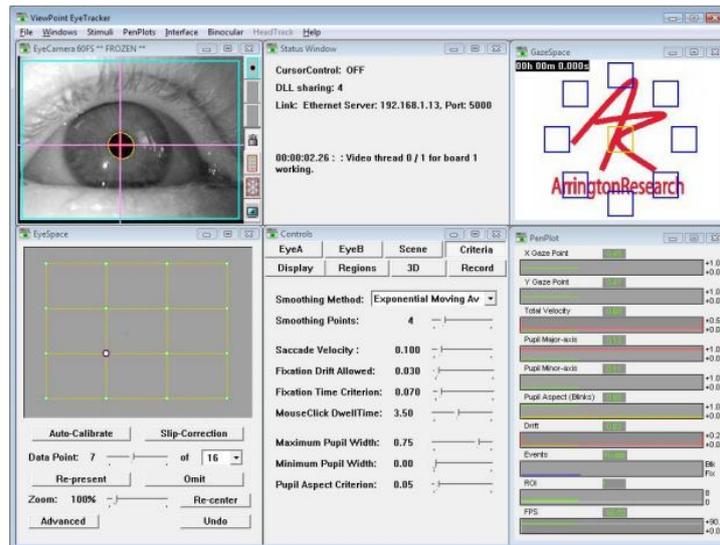


Figure 46: Start-up arrangement of the user windows.

EyeCamera Window

The EyeCamera window tool bar provides easy controls to make the eye tracking results more reliable and to extend the range of traceable subjects. It is very easy to limit the areas within which the software searches for the pupil and corneal reflection to exclude extraneous reflections and dark shadows.

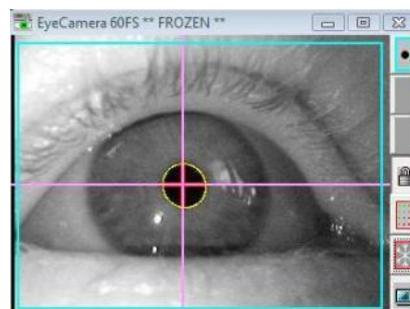


Figure 47: EyeCamera window.

EyeSpace Window

The calibration procedure is easy, flexible and intuitive. The number of calibration points to use could be set, their color, and how fast to present them to the subject or the defaults set within ViewPoint could be kept. The EyeSpace window provides a visualization of the calibration mapping so that it is evident, quickly and easily, whether the calibration was successful. Outliers can be repeated individually if necessary to avoid recalibration. Individual points are easily represented and correction can be performed at any point. Each individual subject's calibration settings can be saved and reloaded for use in subsequent trials.

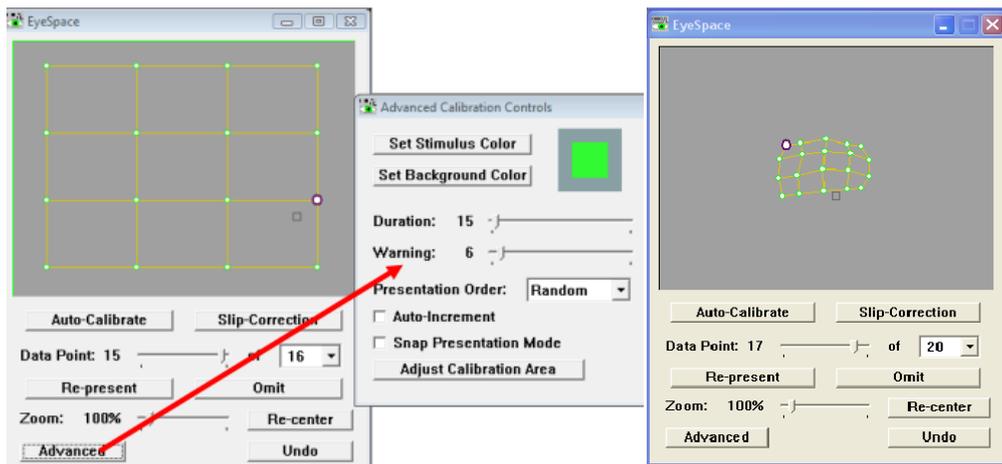


Figure 48: EyeSpace window on the left before calibration. EyeSpace window to the right after calibration of a participant's eye.

Real-Time Feedback

The PenPlot window provides real-time eye position, velocity, torsion and pupil size measurement feedback. The GazeSpace window provides real-time gaze position and fixation information. User defined regions of interests make the analysis task easy.

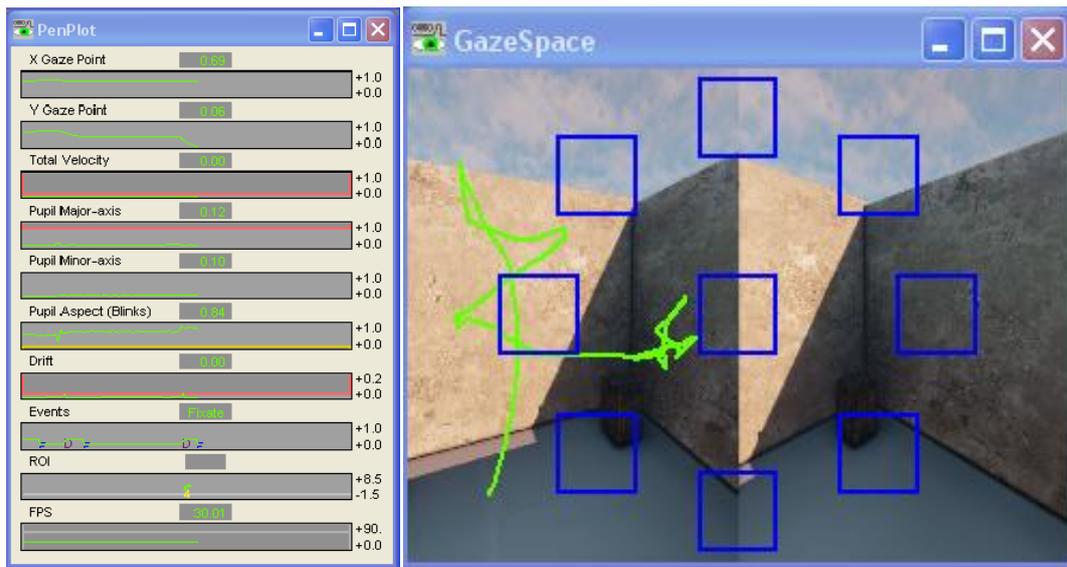


Figure 49: PenPlot and GazeSpace windows on the left and right images respectively.

Controls Window

The controls window provides many features. The most important are: Recording scene and screen movies; locating the pupil and glint methods; feature criteria and regions of interests. The most intricate option to decide on was choosing the most appropriate method for the pupil and glint. A good choice on this option leads to reliable data acquisition from the eyes with minimum adjustments. It's worth mentioning that out of all feature methods "Pupil Location" was chosen as it reliably produced the most accurate and stable results with minimum tuning. The "Pupil Location" method simply identifies the pupil of the eye as the darkest area in the grayscale image received from the infrared cameras. "Pupil Location"

tolerates well in-and-out (closer or farther from the camera, z-axis) movement of the head but is more sensitive to translation of the head in a horizontal (sideways, x-axis) or vertical (up/down, y-axis) direction.

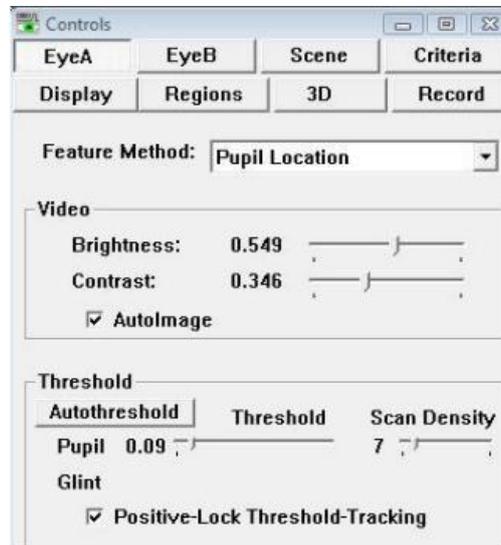


Figure 50: Control Window in tab of the EyeA (right eye) where can be adjusted various parameters for the “Pupil Location” method.

3.4.2 Binocular vs. Monocular Method

The Eye-tracker used in this experiment supports both binocular and monocular eye tracking, i.e. using either both cameras or a single one. Binocular eye tracking has the added advantage that by using both eyes' point gaze data it can calculate the z-axis coordinates correlated to the participant's point-of-view within the virtual scene. In other words, by exploiting binocular parallax the depth of a gaze point can be estimated, thus discriminating between collinear objects that project on the same 2-dimensional coordinates on the retina. On the other hand, calibrating two cameras is time consuming and makes participants feel dizzy and uncomfortable. Additionally, the EyeSpace coordinate system acquired after calibration is harder to manipulate, requiring complex calculations to get mapped to our eye tracking data acquisition infrastructure. On the other hand, the monocular method lacks the depth discrimination of gaze points that was mentioned above, but is easier to utilize and needs less time to calibrate its EyeSpace coordinate system. For this reason we opted for monocular tracking of eye gaze.

4 Chapter 4 – UI Implementation

In this chapter, the implementation of the User Interface will be described, e.g., the way in which the 3D Virtual Scenes were designed and the applications which were used for the needs of the experiment. In addition, the lighting effects simulating the sun and the Flash UI which included the main menu in order to control all stages and the procedure of the experiment will be described.

4.1 Creating the 3D Virtual Scenes

In order to create the 3D Virtual Scenes that the application would render, several steps were required, from creating the actual 3D objects placed in the scenes to importing them inside UDK and placing them to a virtual scene, as well as creating and assigning the materials to these objects. These steps will be further explained below.

4.1.1 Creating the Visual Content

A six by six meters virtual house was chosen as the rendered displayed environment, divided in four zones according to the experiment's specifications. The four zones designed was the lounge, office, kitchen and dining area. Scattered around each zone, six 3D objects as shown in Table 1 were placed in random order.

3D models were created or downloaded from 3D models' repositories and were placed in a scene with the help of an industry-standard 3D modeling software (Autodesk 3ds Max Design 2012). The final result can be seen in Figure 54, which depicts the scene in lit mode (after calculating the lighting space of the scene).

Before the 3D objects could be exported, two UV channels had to be set for each one of them. The first channel was designed to reflect the way a texture wraps up the 3D model and the second channel laid down the surfaces of the object for the computation of the lighting effects on the object later in UDK.

Lounge	Office	Kitchen	Dining
Coats hanger	Sword	Basketball	Candlestick
Television	Tennis racquet	Pen	Clothe hanger
Coffee mug	Pan	Toothbrush	American football
Coffeepot	Office basket	Teapot	Screwdriver
Cowboy hat	Book	Wine	Flower vase
Hammer	Lighter	Plate	Tablecloth

Table 1: Objects existing in each zone of the house.

Figure 51 shows the 3D representation of a lounge table object, from different viewpoints in 3ds Max. The object has only a basic gray material applied on it, since these were assigned at a later stage in UDK, because UDK does not support the import of materials created in 3ds Max. These materials were applied on the object in order for UDK to find out the different material slots that exist for that object. The geometry of the object will be exported to the UDK and can then be used as an actor.

Chapter 4 – UI Implementation

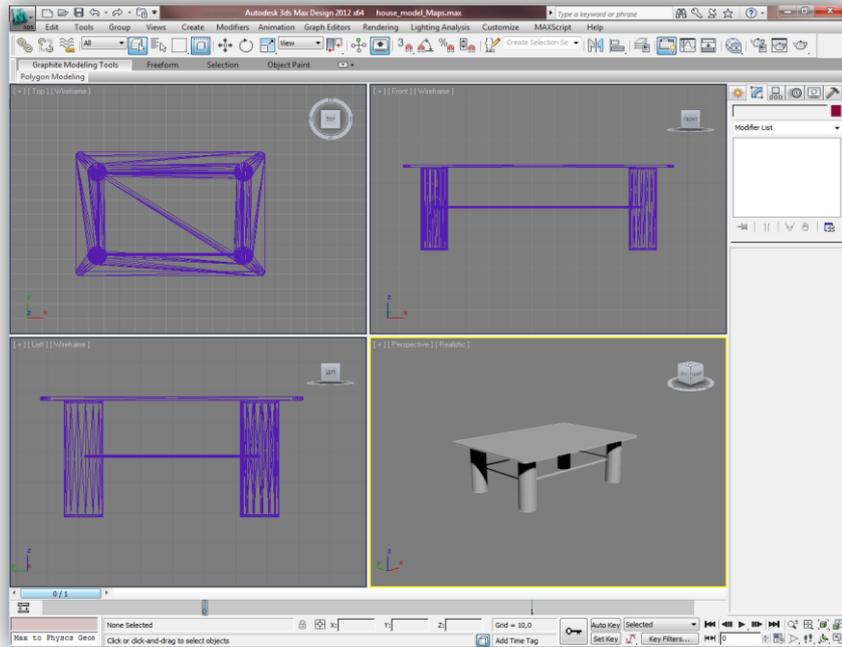


Figure 51: A 3D object representation of a lounge table. The 3d object was created in 3ds Max and its geometry will be exported to UDK.

In order to create and assign a material to an object with correctly assigned textures, as well as realistically illuminate the object imported in UDK, there must be two UV channels defined for each 3D object in 3ds Max. The first UV channel will be used to apply and align a material onto the object and can be created by applying a UVW map modifier to the editable mesh representing the object, as seen in Figure 52.

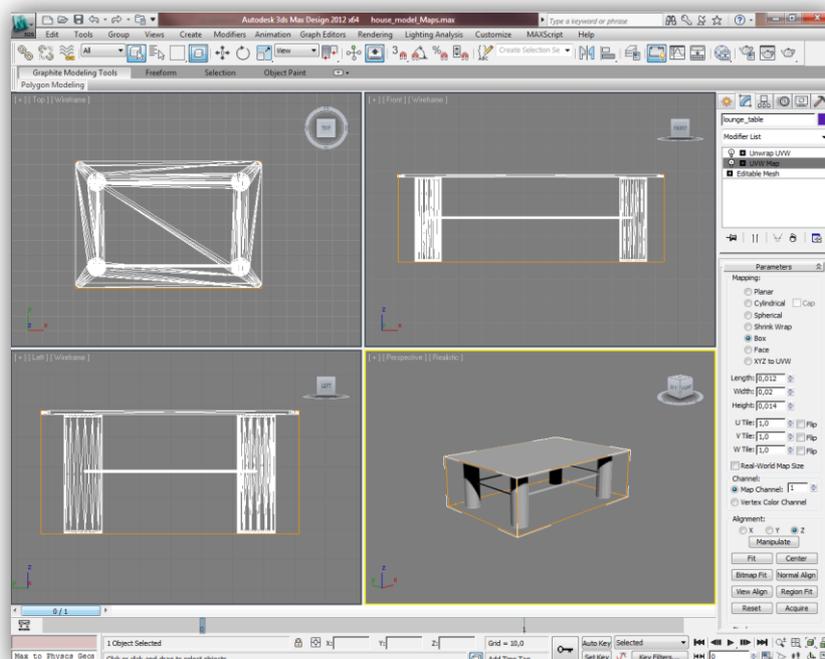


Figure 52: The UVW mapping of an object in 3ds Max. This UV channel is used by UDK in order to apply a material correctly onto the object.

The second UV channel is needed by UDK in order to correctly apply illumination and shading effects on the object, giving more detail on the bigger polygons of the channel. This channel can be created in 3ds Max by applying a UVW unwrap modifier on the editable mesh that will be exported and then, in the respective editor, select to automatically flatten the UVW mapping. An example of this can be seen in Figure 53.

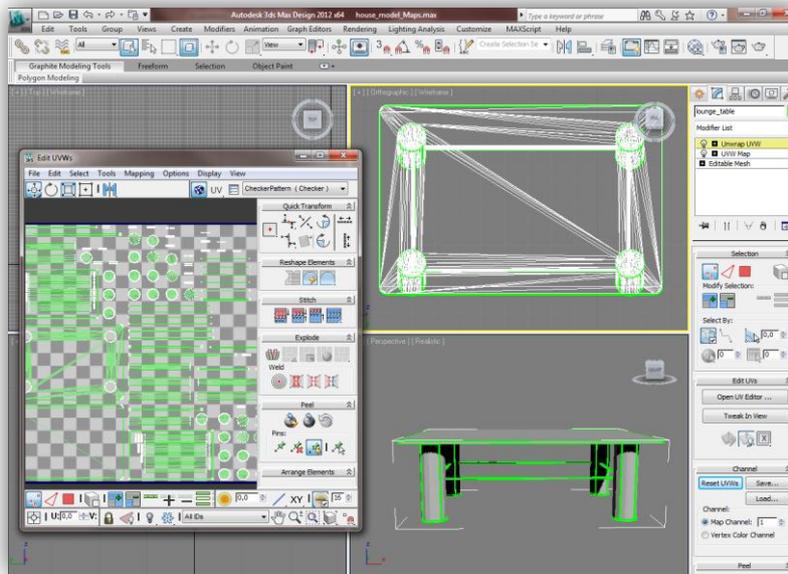


Figure 53: The UVW unwrapping of an object in 3ds Max. This UV channel is used by UDK in order to realistically illuminate an object, giving more shading detail in the bigger polygons of the object in the channel.

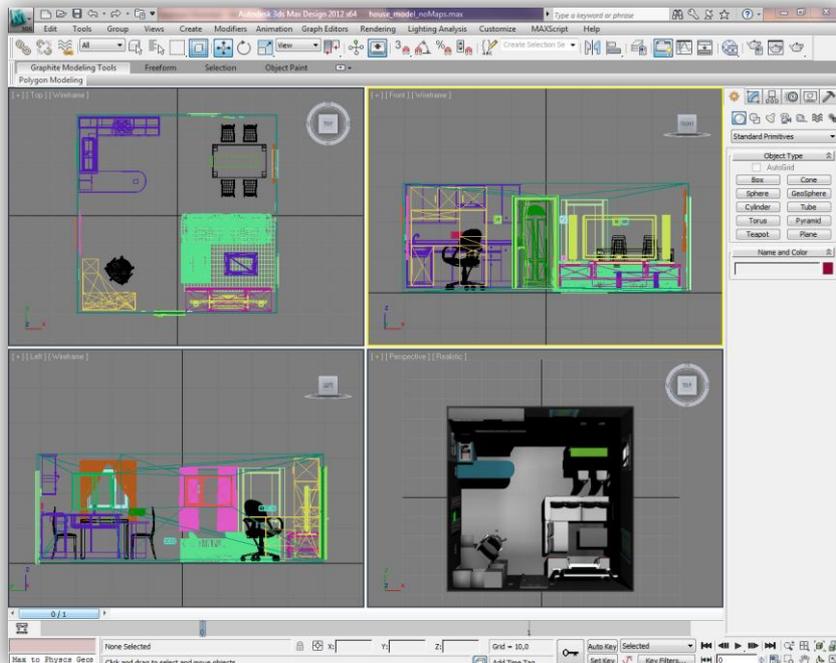


Figure 54: The final scene with all the 3D objects created in 3ds Max. The final scene does not include items that were placed in each zone because their choice was made much later.

4.1.2 Creating texture maps

When the geometry of the scene was completed, textures were applied to elements of the scene and objects.

In order to create the textures maps of the models, Shader Map Pro was used. Although it is not free, it can be used without license limited to using only the command line interface (CLI).

An example of creating a texture map for a 3D carpet model is presented (Figure 55). In order to give realistic detail to the 3D carpet model a normal map is applied. The normal map creates the visual appearance of uneven surface, therefore, it adds realism to the object.

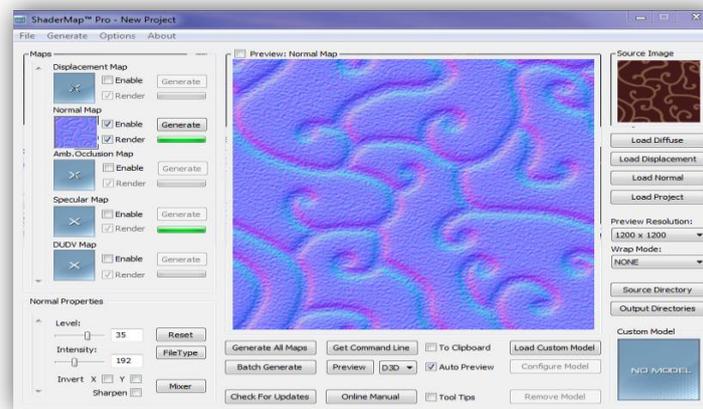


Figure 55: The image represents the normal map applied on the cylinder model as showing in the figure below (second image).

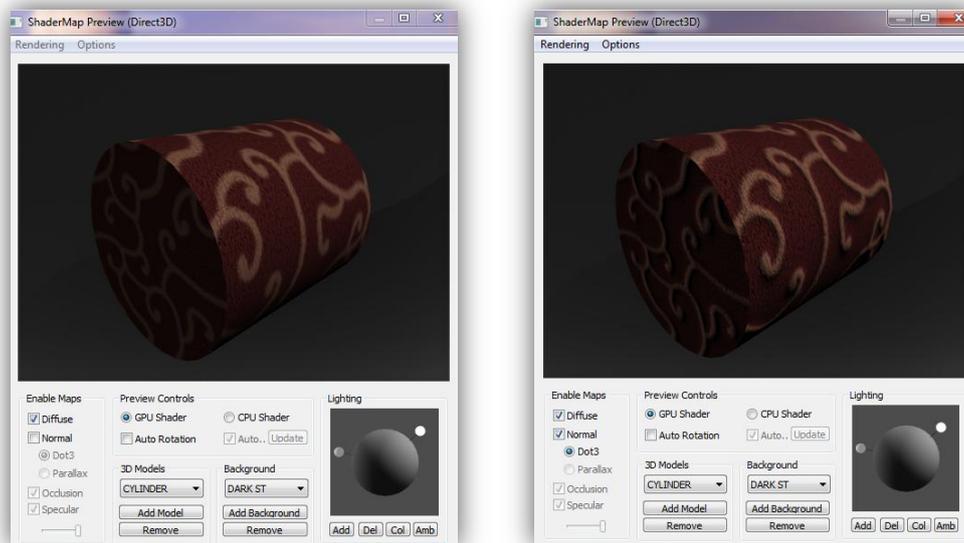


Figure 56: First picture shows the original diffuse map (image), second one is after applying a normal map on it.

Another example of creating a texture map is the need for surfaces that reflect the light. The following figure depicts the specular map of a wooden surface. Applying the specular map makes the surface of the 3D object reflect any sort of light.

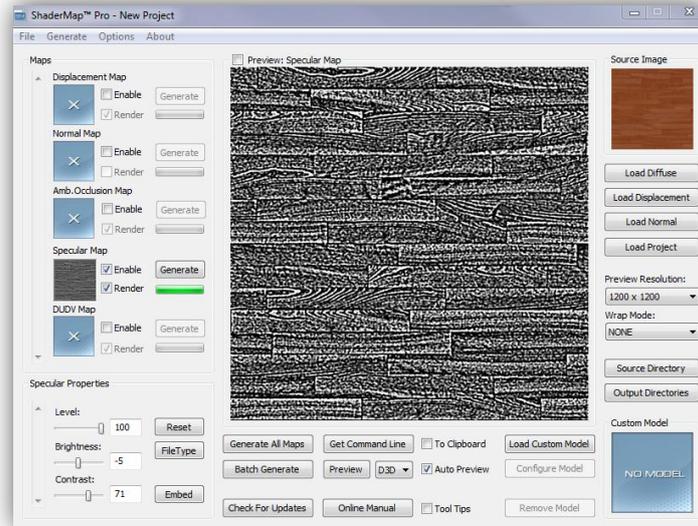


Figure 57: The image represents the normal map applied on the cylinder model as showing in the figure below (second image).

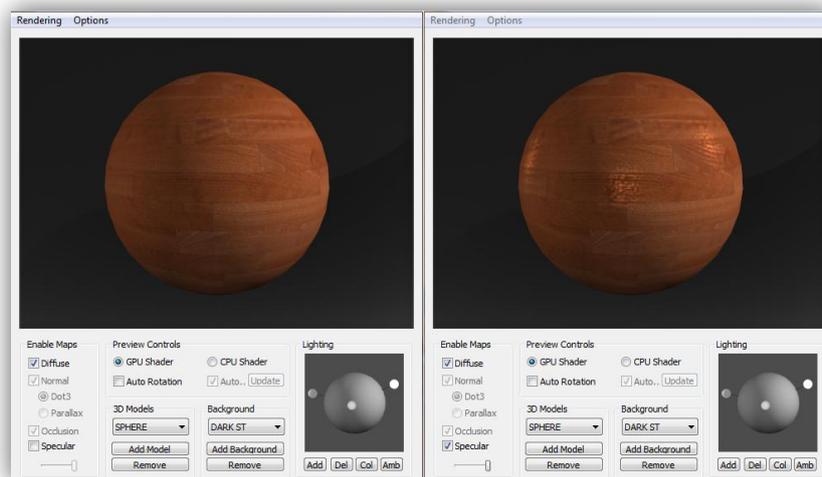


Figure 58: First picture shows the original diffuse map (image), second one is after applying a specular map on it.

It must be noted that certain objects do not require any other maps except a diffuse map such as, for example, a 3D book model which is placed under shadow.

4.1.3 Setting up the virtual scene in UDK

Inside UDK, the created 3D objects were imported, by selecting the import option in the asset library of the Unreal Editor. UDK reads the file containing the exported 3D objects and recreates the geometry of the objects. It also creates the different material slots for each part of the 3d object and initializes the UV channels of the object.

Although the collision shape of a 3D object can be automatically performed by UDK using the collision detection system, the experiments required a different solution instead of the automatic collision one. Blocking volumes were created for each 3D object, according to its

geometry. The advantage of this choice was that it provided the capability to manually change the dimension of blocking volumes which was not possible to do with a collision shape. For the purposes of the experiments, the collision detection mechanism had to be precise and accurate. Blocking volumes were applied only on the twenty-four 3D objects scattered around the scene. The technical details and justification in relation to employing blocking volumes will be described in the next chapters.

An example of the creation of a blocking volume is presented in the following figure, which shows a rose box and inside a wine bottle 3D model imported in UDK. The rose box surrounding the geometry of the 3D object is the blocking volume that was created for that object.

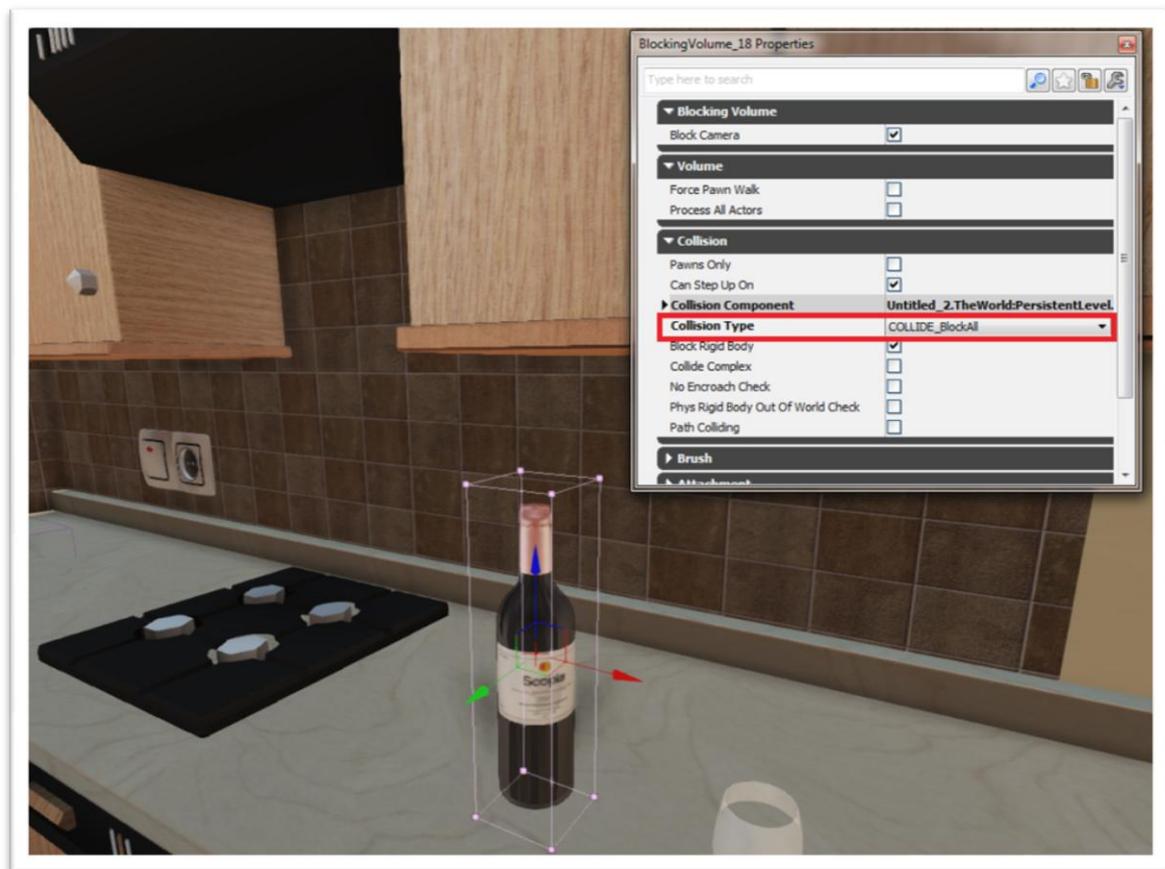


Figure 59: A manually blocking volume was created and applied on the wine bottle 3D model. One thing to pay attention is that in the properties of the blocking volume is the option in the red rectangular. Collision type is set to “COLLIDE_BlockAll” which means that nothing can go through that volume.

The imported models were placed inside a new virtual scene. Then the texture images for each object were imported and materials were created and assigned to the objects in the scene, respecting the textures and the properties of the material for each object. As already mentioned, UDK’s Material Editor offers the ability to not only set a diffuse texture or expression for a material, but also alter its properties, such as setting the specular color and power of the material, or defining a normal map, which can realistically differentiate the lighting of a rough (not smooth) surface.

Chapter 4 – UI Implementation

For example, Figure 60 shows two materials in the Material Editor, applied on a sample sphere object. The first material on the left is a matte grey one, with no specular properties and consequently does not produce any lighting effects on the object's surface. The material on the right has a white color connected to its specular property, with high specular power (20.0), so it produces a specular effect due to the light that the object receives.

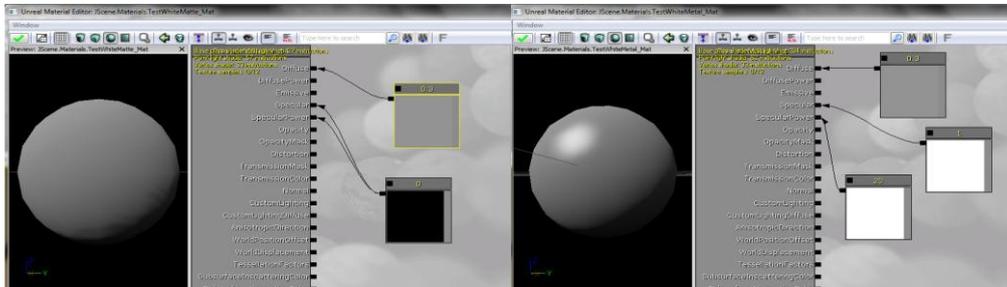


Figure 60: On the left is a grey matte material, with no specular color. On the right is the same grey material with white specular color.

The Setting up a normal map for a material helps Lightmass calculate the light that bounces on the object with that material applied and scatter it according to the supplied normal map. This is very helpful in order to create the illusion of rough or relief surfaces on objects with smooth geometry, such as the wooden surface of a table, or a carpet.

For example, in order to create a wool carpet, the geometry of the carpet should be extremely complex, so as to describe the wool surface. The other option is to create a normal map for the carpet material, defining that although the geometry's surface is smooth, the light should scatter as though bouncing on a wool surface, as can be seen in Figure 61.

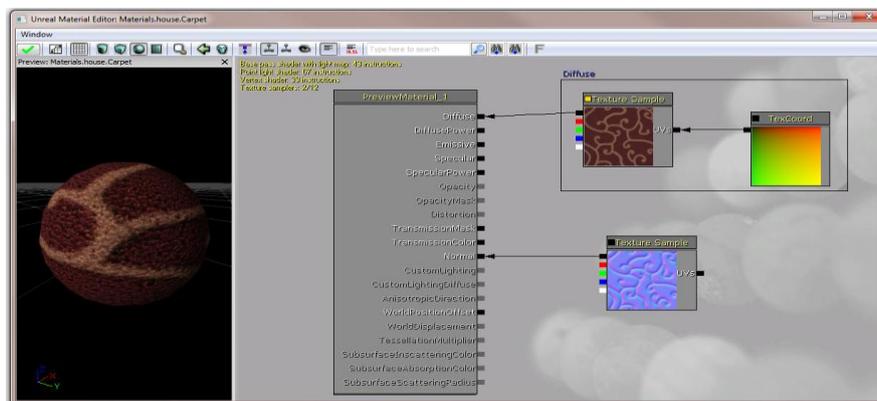


Figure 61: A material for the surface of a carpet with a normal map defined. As can be seen from the sample sphere with the material applied on it, the surface of the carpet is relief.

This is a very efficient way of creating the illusion of rough or relief surfaces, with realistic lighting effects, while keeping the triangle count of the 3D objects at low levels. This is demonstrated in Figure 62, where the normal-mapped wool material was applied on a simple box object, with only 43 triangles, representing the surface of a carpet. As can be seen, the lighting effects seem very realistic, just as a very complex – in terms of triangle count – geometry would produce.

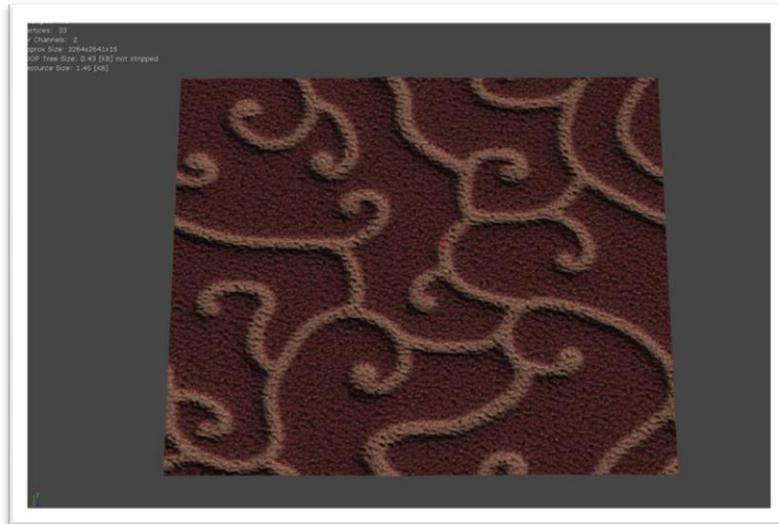


Figure 62: A carpet surface object with the normal mapped material applied on it. Although the carpet's geometry is simple, with only 43 triangles, the lighting effects give the illusion of a wool (and complex) surface.

For the experiments, it was required that the lighting effects of the sun light could be visible from the indoors room through a door with glass windows. This means that the materials for these windows should be transparent, representing a glass material. Also, they should allow the light to come through and illuminate the room. UDK supports translucent materials, by offering the option to set a material as such, thus, allowing the light to pass through it.

The diffuse and emissive properties of the translucent material affect the color of the light that passes through the material. There are different levels of translucency supported, defined by the opacity property of the material, so that the higher the opacity value of a material, the less light it allows to pass through. There is an example displayed in Figure 63, showing the glass material used on the glass window 3D objects in the scene.

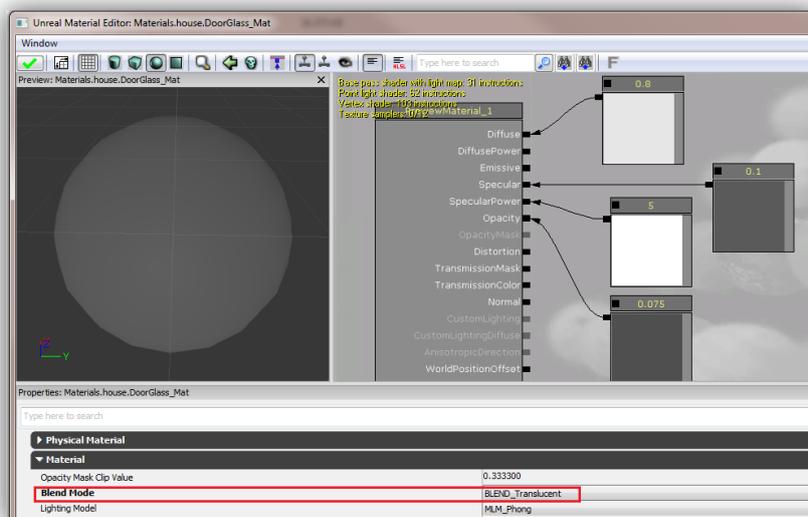


Figure 63: A translucent material representing the glass in a window.

As explained above, creating the materials for the 3D objects and inserting them into the required virtual scene gave the impression of them appearing as real-world objects. The resulting scene including the 3d objects placed inside is shown in Figure 64. It must be noted that this virtual scene is still far from being considered as realistic, since there are no lighting effects present. For this reason, the lighting of the scene must be configured and Lightmass should be allowed to execute and handle the lighting computation.



Figure 64: An unlit scene with the modeled 3D objects in UDK. The created materials are applied to the respective objects.

Lighting effects provide a huge difference in how realistic a virtual scene is perceived to be and UDK's Lightmass will be used to precompute these lighting effects.

For the experiment to go through, it was required to compute the lighting effects simulating the sunlight, since it was the only one light source. In order to achieve that a dominant directional light was used, representing the sun. The sunlight was rendered according to the default midday lighting settings provided by UDK. The default lightning configuration settings can be viewed in Figure 65.

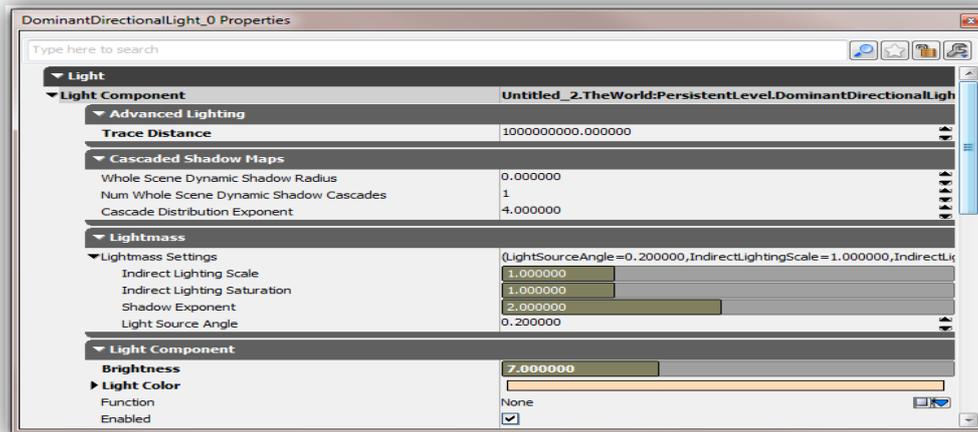


Figure 65: Default Midday lighting configuration.

During the rendering process Lightmass was instructed to simulate three bounces of indirect lighting between all geometry of the virtual scene and the final results can be seen in the following figure. The resulting scenes of the Lightmass algorithm computation are shown in the following figure.



Figure 66: Screenshot of final scene after the light computation using unreal light mass.

4.2 Application menu as Flash UIs

Although UDK has preliminary support for User Interfaces (UI), the ability to embed Flash User Interfaces was preferred for the requirements of the experiments. The Flash applications that were used as User Interfaces included the need for main menu.

The application used in the experiments included only a main menu in order to control all the stages of the experiment including the testing phase. The menu screen that appeared when the experiment started is depicted on the following figure.



Figure 67: The start Flash menu that was displayed when the experiment application was started.

In order for UDK to embed a Flash application as a user interface and display it, it is required to import the .SWF file of the Flash UI. Then, the user interface can be loaded and displayed in an empty scene, by connecting the "Open Gfx Movie" action to the "Level Loaded and Visible" event, which is automatically generated and activated by UDK when the virtual scene becomes visible. The "Open Gfx Movie" action is provided by UDK and it accepts an imported Flash UI as an argument, which it loads and displays on the screen or on the specified surface, if one has been specified. For the specific needs of the menu screens in the experiments, the Flash menus should be displayed at the center of the screen.

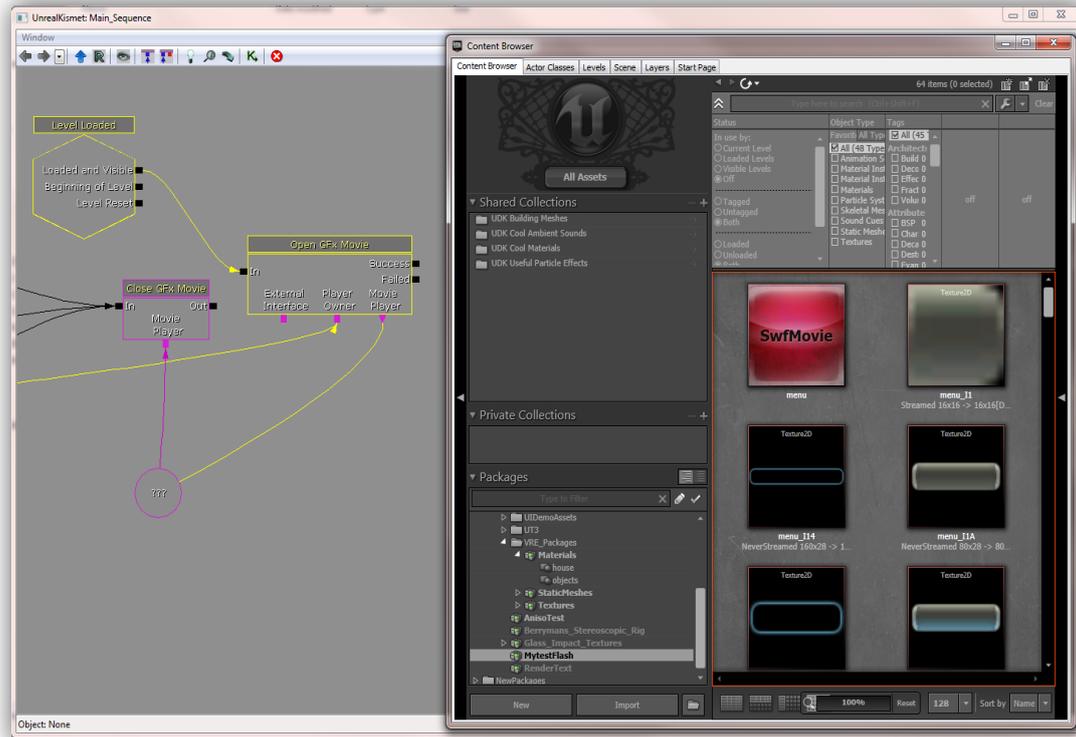


Figure 68: The start menu Flash UI is loaded and displayed immediately after the virtual scene becomes visible.

The experiment consisted of two phases. During phase one each participant navigates a training scene in order to get familiar with the Immersive Virtual Environment (IVE) and the equipment. The second phase is the main experiment, when each participant navigates the house scene. For debugging purposes, an additional testing stage was included in order to identify any possible errors during the main experiment such as eye tracking validation testing.

To start each stage the “Start” or “Training” button was pressed by the researchers. Pressing the “Start” or “Training” button in the Flash UI results in the execution of a call to a respective method in the controller class to start the preferred stage. In particular pressing the “Start” or “Training” button results in the following code to be executed:

```
ExternalInterface.call("startExperiment"); //Start the main experiment and load the house scene
```

```
ExternalInterface.call("startTraining"); //Load the training scene
```

The Flash player of the UDK then searches for the *StartExperiment* or *StartTraining* methods in the controller class and executes them, which initiates the respective stage of the experiment. The controller then calls the appropriate method, initializes and loads the proper scene. Each stage had to be initiated by the researchers so as to avoid accidental choices by the subjects.

5 Chapter 5 – Implementation

In this chapter, the implementation and development of the application used in the virtual experiments will be described. More specifically, the way in which the 3D Virtual Scene was configured, in order to make the virtual environment stereoscopic and work according to the head mounted display's specifications. In addition, the participants' actions logging system will be described in detail. Experimental design issues, which were requisite in order to conduct eye-tracking experiments and be able to examine and analyze the gaze data, will also be presented.

5.1 UnrealScript Classes

In order to develop the application required for the experiments, several classes had to be created in UnrealScript, which would handle the aspects of the application's rendering, navigation and interaction with the Virtual Scenes. The most important classes that were created will be presented here.

VEGame: This was the class of the application that defined the main properties, such as the Pawn that would be used in the application and the Controller that would handle the pawn. This class extended the `GameInfo` class, as can be seen in its declaration:

```
class VEGame extends GameInfo;
```

This class was only needed to define the default head up display (HUD), pawn and controller for the pawn that would be used in the experiments, when the virtual scene was loaded. It defines the game being played: the game rules, scoring, what actors are allowed to exist in this game type, and who may enter the game. While this class is the public interface, much of its functionality is delegated to several classes to allow easy modification of specific game components. A `VEGame` actor is instantiated when the level is initialized for gameplay (in C++ `UGameEngine::LoadMap()`). The class of this actor is determined by the `DefaultGame` entry in the game's `.ini` file (in the `Engine.Engine` section), which was set to be `VEGame`. The `GameType` used can be overridden in the class's script event `SetGameType()`, called on the game class picked by the above process.

The experiment flow was controlled from the controller of the pawn, so the only code needed in this class was simply to define these two classes – the default pawn and the default controller of the pawn – in the default properties block of the class. In addition, after the virtual scene was finished a custom HUD class for testing purposes was set as default in order to override the standard one. Follows the code for setting as defaults the three classes:

```
DefaultProperties
{
    //Start game immediately
    bDelayedStart=false;

    //Pawn class
    DefaultPawnClass=class'VirtualExperiment.VEPawn';
    //Player Controller class
```

```

PlayerControllerClass=class'VirtualExperiment.VEPlayerController';
//My custom head up display
HUDType=class'VirtualExperiment.VEHud';
}

```

VEPawn: This class defined the Pawn that would be used in the application. When a scene was started, a new VEPawn was instantiated, as instructed from the VEGame class. It extended from the GamePawn class and defined the main properties of the used Pawn, such as the height, the speed and the collision radius of the Pawn. The class declaration was the following:

```
class VEPawn extends GamePawn;
```

The only pawn used in the experiments was that belonging to the participant and it was assumed to be a simple camera rotating inside the virtual scene. Also, UDK supports the ability for pawns to jump, swim, fly, climb ladders, etc. This was not wanted for the experiments, so these abilities were disabled in the default properties of the pawn class. One more important thing that was configured through the pawn class is the EyeHeight of the pawn which was calculated based on the proportions of the house inside the virtual scene.

The settings used for the pawn in its default properties block can be seen below:

```

DefaultProperties
{
    // These variables force palyer to remain in a
    // static position inside the house sence
    bCanBeDamaged=false
    bCanCrouch=false
    bCanFly=false
    bCanJump=false
    bCanSwim=false
    bCanTeleport=false
    bCanWalk=false
    bJumpCapable=false
    bCanCrouch=false;
    bProjTarget=true
    bSimulateGravity=true
    bShouldBaseAtStartup=true

    // Locomotion
    WalkingPhysics=PHYS_Walking
    AccelRate=+0.0
    DesiredSpeed=+0.0
    MaxDesiredSpeed=+0.0
    AirSpeed=+0.0
    GroundSpeed=+0.0
    JumpZ=+0.0
    AirControl=+0.0
}

```

```

// Physics
AvgPhysicsTime=+00000.100000
bPushesRigidBodies=false
RBPushRadius=10.0
RBPushStrength=50.0

// FOV / Sight
ViewPitchMin=-3500 //Limiting the Pitch axis
ViewPitchMax=6000 //Limiting the Pitch axis
RotationRate=(Pitch=20000,Yaw=20000,Roll=20000)
MaxPitchLimit=3072
SightRadius=+05000.000000

MaxStepHeight=0
MaxJumpHeight=0
WalkableFloorZ=0.0 // 0.7 ~= 45 degree angle for floor
LedgeCheckThreshold=4.0f

//Eye Height
BaseEyeHeight=+160
EyeHeight=+160
}

```

VEPlayerController: This class was used as the Controller of the VEPawn and took control of the created Pawn when a scene was started, as instructed by VEGame. It was the class that handled all aspects of the application and in which all computations were taking place, such as navigation, interactions, or loading the scene in either normal or test mode. It extended from GamePlayerController and the declaration of the class was the following:

```
class VEPlayerController extends GamePlayerController ;
```

Unlike the previous classes, the VEPlayerController class did not consist only of the default properties block, since it was responsible for the flow of the experiments. It contained all the functions and code necessary to control the state of the experiment and the events that should occur at specific time points. All of these will be described in different parts in the following subsections. In the default properties block of this class, are all the default variables which are responsible on how the experiment will be executed. For example, if the variable *TestMode* is set *True* the experiment will be started in test mode and a custom HUD will be showed up in the scene. Follows the default properties of the *VEPlayerController* class:

```

DefaultProperties
{
    participantID =-1;
    ActiveEye =RightEye; //By default active eye is consider the right eye
    ClonePlayer = none; //ClonePlayer controller when in stereoscopic vision
    Stereoscopy =false; //Variable becomes true when stereoscopic view is setup
}

```

```

StageDuration = 120; //Experiment Duration (time in sec)
TestMode = false; // (if TestMode = true then scene is loaded within test mode)
FirstController= false;
}

```

5.2 Handling User Input

The navigation of the scene by the participants was limited according to the specifications of the experiment. Thus, participants could only rotate around the scene from a static position in the center of the house, sitting on a swivel chair. The rotation in the horizontal (yaw) axis was 360° and for the vertical (pitch) axis was bounded into -20° downwards and $+33^{\circ}$ upwards from eye's height of the pawn. The reason for that is to constrain the participants from looking at the floor or the ceiling. The user input was captured through the head-tracking device InertiaCube3 provided by the manufacture company InterSense while each participant looked around the scene. The software of the device supports the ability of the head-tracker to act as a computer mouse. Overriding the mouse signals with those of the head-tracker, each participant's input was taking place in the virtual scene. That's was a perfect solution because a mouse device has the same axis as those participants needed in the virtual scene.

5.3 Logging of events and participants' actions mechanism

The application was recording every event that was happening, as well as every action of the participant in the database system, in order to be able to understand and analyze the data gathered from each experiment. The communication between UDK and the database system was described in Chapter 3. An event can be defined as the record that was generated every 0,05 second during the experiment in the virtual scene. An action can be defined as the participant's activity at the specific event. For example, a participant's action could be the viewing of an object observed or the region in the house observed at a specific time. Every generated event has a specific number of attributes in the *Logging* table as following:

participantID: The identification number of each participant.

event: The zone in the house where an event has occurred.

time: The current time in experiment

hitObject: The object that a user was looking at.

hitLocation: The location of the hitObject in the virtual scene.

HT Location: The Head tracker device data (2-axis yaw, pitch).

ET Location: Eye tracking device data (coordinates x,y in the right/left screen depended on the active eye of participant).

participantID	event	time	hitObject	hitLocation	HT_Location	ET_Location
1	office	15,9614	BlockingVolume_12	-412.88,94.76,20.	353,139,0	-187.79,-82.22
1	office	16,0091	None	-391.59,94.84,0.6	350,140,0	-187.13,-82.20
1	office	16,0576	None	-395.13,81.38,0.6	346,142,0	-187.09,-82.75
1	office	16,1069	BlockingVolume_12	-390.96,94.03,16.	343,144,0	-186.75,-82.53
1	office	16,1562	BlockingVolume_12	-413.70,108.88,20.	340,146,0	-186.85,-82.58
1	office	16,2066	None	-443.47,156.55,0.	340,148,0	-186.68,-82.17
1	office	16,2573	None	-451.63,159.56,0.	340,150,0	-186.79,-82.18
1	office	16,3089	BlockingVolume_12	-401.25,68.19,20.	340,152,0	-187.07,-83.31
1	office	16,3576	None	-451.18,101.54,0.	340,154,0	-187.28,-83.21
1	office	16,4077	None	-466.45,107.85,0.	340,155,0	-187.42,-83.23
1	office	16,4576	None	-453.98,102.75,0.	340,157,0	-187.36,-83.18
1	office	16,5076	None	-468.85,114.28,0.	340,158,0	-187.46,-83.08
1	officekitchen	16,5573	BlockingVolume_12	-407.80,75.63,20.	340,160,0	-187.24,-83.11
1	officekitchen	16,6072	None	-451.39,108.55,0.	340,161,0	-187.41,-82.95
1	officekitchen	16,6576	BlockingVolume_12	-397.56,70.82,20.	340,162,0	-187.22,-83.04
1	officekitchen	16,7081	BlockingVolume_12	-396.69,67.91,18.	340,163,0	-187.26,-83.09
1	officekitchen	16,7588	BlockingVolume_12	-397.64,67.91,19.	340,164,0	-187.30,-83.08
1	officekitchen	16,8097	BlockingVolume_12	-397.47,69.52,20.	340,164,0	-187.31,-83.02
1	officekitchen	16,8612	None	-412.68,77.16,0.6	340,165,0	-187.31,-83.12
1	officekitchen	16,9114	None	-403.19,75.82,0.6	340,165,0	-187.22,-83.02
1	officekitchen	16,9627	BlockingVolume_12	-390.94,69.05,16.	340,165,0	-187.22,-82.94
1	officekitchen	17,0134	BlockingVolume_12	-390.94,69.05,16.	340,165,0	-187.22,-82.94
1	officekitchen	17,0644	BlockingVolume_12	-390.93,70.05,18.	340,165,0	-187.23,-82.90
1	officekitchen	17,1154	BlockingVolume_12	-391.02,70.69,20.	340,165,0	-187.25,-82.87
1	officekitchen	17,1661	BlockingVolume_12	-394.27,71.47,20.	341,165,0	-187.31,-82.88
1	officekitchen	17,2162	BlockingVolume_12	-390.18,72.06,20.	340,164,0	-187.40,-82.89

Figure 69: Logging table showing sample participant’s records from the database file.

The data contained in the red rectangle in Figure 69 are meaningless for the statistical analysis. However, it can be used to reproduce the participants’ movements and their exact Field of View (what they were looking at each moment) within the virtual scene. In essence, we could regenerate the navigation patterns for each participant during exposure time for analysis, demo or fun purposes.

Except the *Logging* table which automatically records participants’ actions another table existed in the database file. The *ParticipantInfo* table stores information about each participant and its records were manually inserted before each experiment started. The *ParticipantInfo* table also has a specific number of attributes following:

- ID:** Unique identification number for each participant.
- Gender:** Gender of each participant.
- Age:** Age for each participant.
- ActiveEye:** The dominant eye of each participant either left or right.

Department: The Department of the university for each participant.

Timestamp: Current date of participation in experiment.

ID	Gender	Age	ActiveEye	Department	Timestamp
1	M	24	R	HMMY	28/6/2012
2	M	23	R	HMMY	28/6/2012
3	M	28	R	HMMY	28/6/2012
4	F	20	R	MHPER	29/6/2012
5	M	21	R	HMMY	29/6/2012
6	F	20	R	HMMY	29/6/2012
7	F	20	R	MPD	29/6/2012
8	M	19	R	HMMY	29/6/2012
9	M	22	L	HMMY	28/6/2012
10	M	22	L	HMMY	29/6/2012
11	M	26	R	HMMY	29/6/2012
12	F	27	R	MPD	29/6/2012
13	M	24	L	HMMY	29/6/2012
14	M	22	L	HMMY	29/6/2012
15	M	25	R	HMMY	29/6/2012
16	F	20	R	MHPER	2/7/2012
17	M	26	R	HMMY	2/7/2012
18	M	25	R	HMMY	2/7/2012
19	M	22	L	MPD	2/7/2012
20	M	25	L	HMMY	2/7/2012
21	M	23	R	HMMY	2/7/2012
22	F	37	L	MHPER	2/7/2012
23	F	22	L	HMMY	2/7/2012
24	F	26	R	HMMY	2/7/2012
25	F	24	R	HMMY	3/7/2012
26	M	23	L	HMMY	3/7/2012

Figure 70: ParticipantInfo table showing sample participant’s records from the database file.

Each log entry reported the specific event that occurred, along with the exact time it happened. The time was measured in seconds after the start of the main experiment which was assumed to be time point 0. When the main stage of the experiment started and the controller entered the respective state, the function *SQLiteDB_Init()* is triggered. This function also explained in Chapter 3 is responsible for initializing the driver and connecting to the database file system. Furthermore, it is responsible to read the participant’s *Active Eye value* from the *ParticipantInfo* table and store it in variable within Unreal engine. The respective code is listed here:

```

exec function StartExperiment(){
...
  //Load SQLite Database File
  SQLiteDB_Init();
...
  //Start recording participant actions
  startRecordAction();
...
}

```

```

function SQLiteDB_Init(){
...
  //Read participant current DominantEye
  if(SQLiteDB.SQL_queryDatabase("select ActiveEye from ParticipantInfo where ID= "$participantID$
";")){
  AEye= class'SQLProject_Defines'.static.initString(1);

      if (SQLiteDB.SQL_nextResult())
        SQLiteDB.SQL_getStringVal("ActiveEye",AEye);
        //The default ActiveEye is Right
        if (AEye == "L")
          ActiveEye = LeftEye;
        }
        else
...
  }
}

```

After that, the function *startRecordAction()* which is responsible for determining which controller's data to record is called. This function sets the loop function *recordParticipantAction()* which is called every 0,05 seconds which is the record rate of the data as was defined in the *RecordRate* variable.

```

function startRecordAction(){
...
  if ((FirstController) && (ActiveEye == LeftEye)){
    SetTimer (RecordRate,true,'recordParticipantAction');
    return;
  }

  if ((FirstController) && (ActiveEye == RightEye)){
    CloneController.startRecordAction();
  }
  if (!(FirstController) && ( ActiveEye == RightEye)){
    SetTimer (RecordRate, true,'recordParticipantAction');
  }
}
}

```

Finally after the function *recordParticipantAction()* is initialized for the first time, it is called sequentially every 0,05 seconds which is the record rate of the events, until the end of the experiment. This function is responsible for generating every event and store it in the Database file system. The respective code is listed here:

```
function recordParticipantAction()
{
...
    //Check if participant is not looking an object or looks at the wall
    if ((traceHit == none) || ( tracehit.Name == 'StaticMeshActor_3'))
        traceHit = none;
    //If experiment started in TestMode print additional information on screen
    if (TestMode) { ... }
    ...
    //Insert the data of the event in the database
    SQLiteDB.SQL_queryDatabase("INSERT INTO Logging (participantID , event, time, hitObject,
hitLocation, HT_Location, ET_Location) VALUES
('$participantID$', '$Zones[0]$Zones[1]$Zones[2]$Zones[3]$', '$WorldInfo.TimeSeconds$', '$traceHit
$', '$loc$', '$Rotation* UnrRotToDeg$', '$MousePosition3D$');");
}
```

The tracking of the participant's movement inside the virtual scene is challenging. As the participant's movement was strictly defined to only to rotate from a specific spot it was impossible to define which region of the house participants were looking at. This issue was addressed by dividing the house into 4 equal-sized numbered cylinder zones and assigning an *LOS Trigger* actor in the center of each zone, as can be seen in following figure. The Line-Of-Sight Trigger generates an event when a pawn's line of sight interacts with the trigger's region bounds.

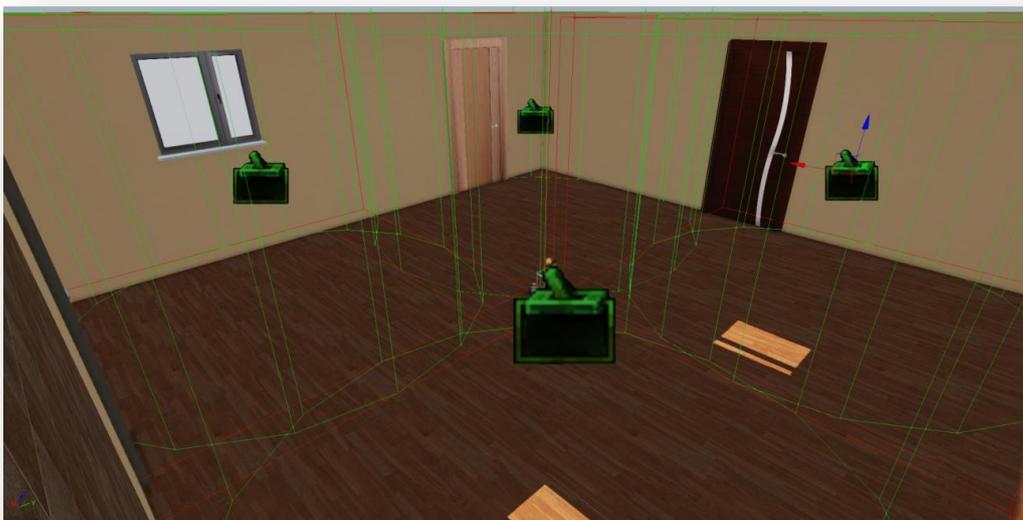


Figure 71: The four LOS Triggers positioning in the center of each zone.

Each one of these Actors was invisible to the participant navigating the virtual scene, but they automatically generated a *LOS* event whenever the participant's pawn interacted with a respective bound zone. All of the *LOS* events were captured in Kismet and an action to record the event in the database file was performed. For example, the figure below depicts the "LOS" event node of such a trigger and its connection to the *Zone Action* action.

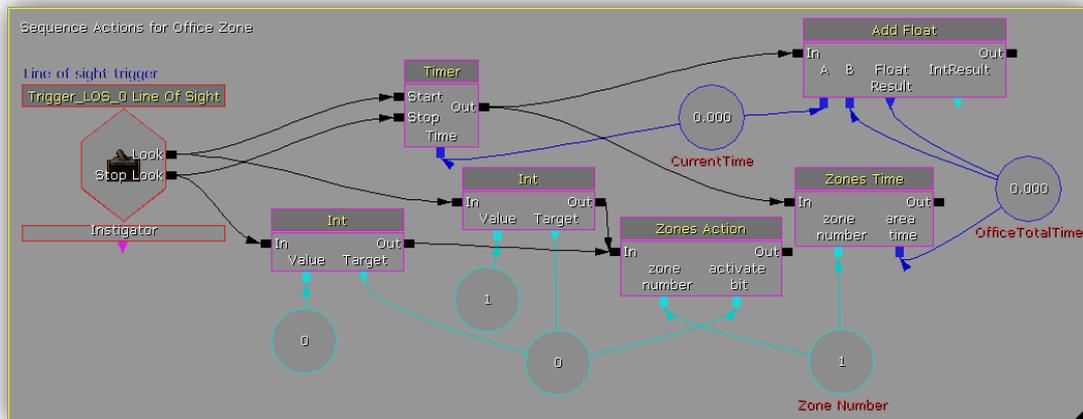


Figure 72: The trigger's "LOS" generated event evokes the Activated method in the Zones_Sequence class.

The *Zones_Sequence* class's Activated method was evoked from the generated event and it requested that the event was passed in the *Controller* class in order to be recorded in the next insertion to the database file. The respective code is listed here:

```
class Zones_Sequence extends SequenceAction ;
...
event Activated()
{
...
    //Identify which LOS Trigger was activated
    if (Active == 1) {
    if (znumber ==0)
        text = "loung";
    if (znumber ==1)
        text = "office";
    if (znumber ==2)
        text = "kitchen";
    if (znumber ==3)
        text = "dinning";
    }
    else {
        text ="";
    }
    //Pass the event's data to the controllers' classes
    VEPlayerController(GetWorldInfo().GetALocalPlayerController()).Zones[znumber] = text;
    VEPlayerController(GetWorldInfo().GetALocalPlayerController()).CloneController.Zones[znumber] = text;
}
```

It must be noted that it is possible that two *LOS triggers* can be active at the same time due to the fact that the participant's Field-of View may be between two regions of the house.



Figure 73: Simultaneous activation of two LOS Triggers.

5.4 Stereoscopic view setup

The HMD device that was used in these experiments uses the partial overlap method to achieve stereoscopic effect. However, the device does not automatically adjust the input image signal to match the partial overlap requirements. Thus, the output image signal from the PC has to be in the correct format as described below, to match the HMD requirements. In order to develop this inside UDK several actions had to be taken. The picture below depicts the final view that had to be implemented inside UDK in order to meet the partial overlap requirements of the HMD.

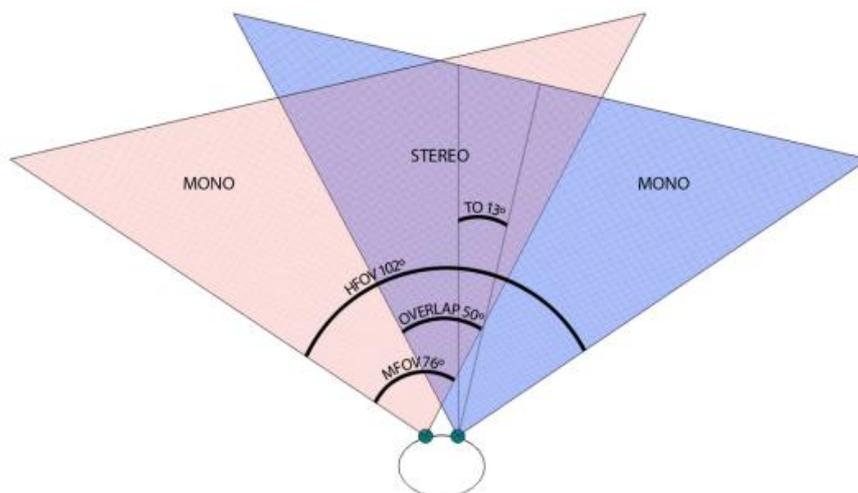


Figure 74: The partial overlap view in the HMD.

First of all, two input video signals were needed for the device to work with each eye receiving its own video. The ideal solution was to output two video signals from UDK with the proper adjustments for each screen of the HMD. Unfortunately, the Unreal engine, at least in relation to its free edition, is currently supporting only one viewport. The alternative solution adopted was to initially split the viewport of UDK in two identical views. The problem from splitting up the viewport was that the resolution was decreased and the HMD required SXGA resolution to work. As SXGA is the resolution of 1024 x 1280 pixels which means that for both screens of the HMD, this amounts to a total of 1024 x 2560 pixels, the resulted resolution of 1024x640 pixels for each separated view was not adequate. In order to fix that, the viewport of the Unreal engine was expanded using the graphics card's option to select two displays with a total resolution of 1024 x 1280 pixels each. Thus, the output signals from the graphics card matched the requirements of the input signal of HMD. In order to achieve the split screen a second controller had to be created and added to the game world space of the Unreal engine as shown in the following code:

```
exec function Stereoscopic()
{
    local int ControllerId;
    local string Error;
    ControllerId =1;
    CreatePlayer(ControllerId, Error, TRUE); //Create a new controller
    SetSplit(2); //Split screen vertical
}
```

The Second step was to make the second controller act exactly as the first; in short this controller had to be a clone of the first one. As the first controller was handled by the user input the second controller had to react exactly as the first one. To make this happen, the position coordinates of the clone controller was equaled with the first controller. In order to avoid any possible delay on the user's view the change of position coordinates of clone player were taking place before the render of each frame. The respective code is shown here:

```
//Function which is called by the game engine before each frame is rendered
function DrawHUD( HUD H )
{
    ...
    parallaxView(); //Make changes before rendering
    super.DrawHUD(H); //Render the scene
    ...
}

function parallaxView(){
    ...
    //If stereoscopic view is activated start Camera Cloning...
    if ( (ClonePlayer!=none) && (Stereoscopy == True) && (self.FirstController)){

        newRot=Rotation;
        newRot.Yaw = self.Rotation.Yaw + 26 *DegToUnrRot; //Coordinates plus offset
        ClonePlayer.Actor.SetRotation(newRot);
    }
}
```

Chapter 5 – Implementation

So far what was achieved was to output two identical video signals, while each user input was handled by the first controller and passed to the second one. However, a full overlap stereoscopic effect was not achieved with both eyes viewing the exact video signal. The final adjustments had to be done according to the partial overlap requirements as shown on the previous figure. Initially the camera of each controller was rotated by 13 degrees left and right respectively. After that, a distance between the positions of each controller's camera was set as the role of the Interpupillary distance (IPD). Finally the field-of-view for each controller's camera was configured and then activated the stereoscopic view. All steps taken as showing the following code:

```
exec function partialOverlap(){
...
//Set initial Rotation difference for each player (controller)
newRot=Rotation;
newRot.Yaw = self.Rotation.Yaw - 13 *DegToUnrRot;
self.SetRotation(newRot);
newRot.Yaw = self.Rotation.Yaw + 26 *DegToUnrRot;
ClonePlayer.Actor.SetRotation(newRot);

//Set a digital fixed Interpupillary Distance
IPD.X = self.Location .X + 15;
ClonePlayer.Actor.SetLocation(IPD);

// Set field of view for each camera to match specifications of HMD
ClonePlayer.Actor.FOV(76);
self.FOV(76);

//Final step activate CameraClone
Stereoscopy =true;
...
}
```



Figure 75: The two output images displayed each on left and right screen respectively in the HMD.

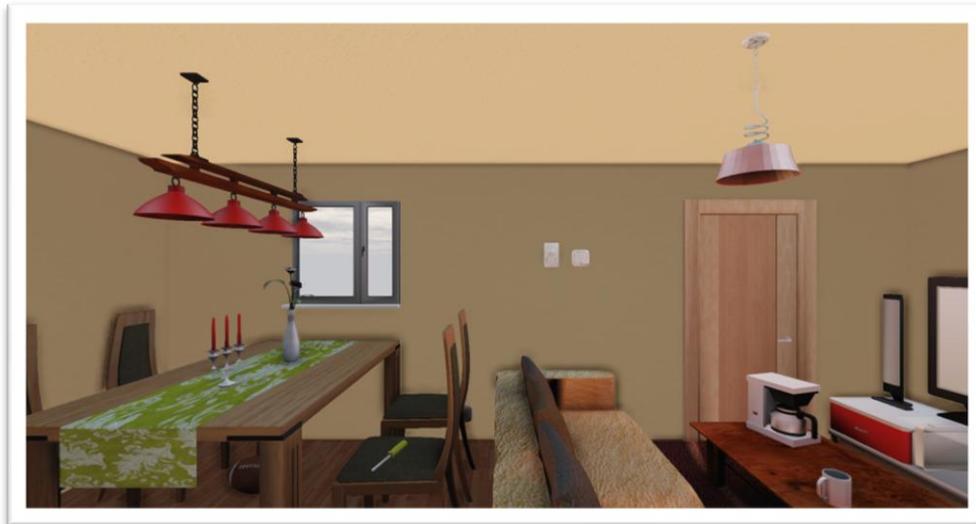


Figure 76: The final scene that user views within the HMD.

5.5 Deprojection of eye's gaze data

Projection refers to the transformation between world coordinates and screen coordinates. This is a very useful concept in certain situations. It can be used to draw elements on the HUD that appear to be located at specific locations in the world, such as floating above a player, or it can be used to determine what objects in the world the player is aiming at or the mouse cursor is over. A real time strategy (RTS) game would make heavy use of these concepts in order to select units in the world. The ability to display pertinent information about an item in the world can be useful in just about any type of game.

Projection refers to the transformation of a 3D world-space vector into 2D screen coordinates and Deprojection refers to the transformation of a 2D screen coordinates into a 3D world-space origin and direction. The receiving data from the Viewpoint application is the gaze space of the eye of each participant representing 2D screen coordinates. This data is not useful in 2D as it cannot be used to determine which object/actor in the 3D world space a participant is looking at. The *DeProject()* function of the Unreal game engine takes a set of screen coordinates in the form of a Vector2D and transforms those into origin and direction Vectors, which are the components of a ray as showed in the following figure.

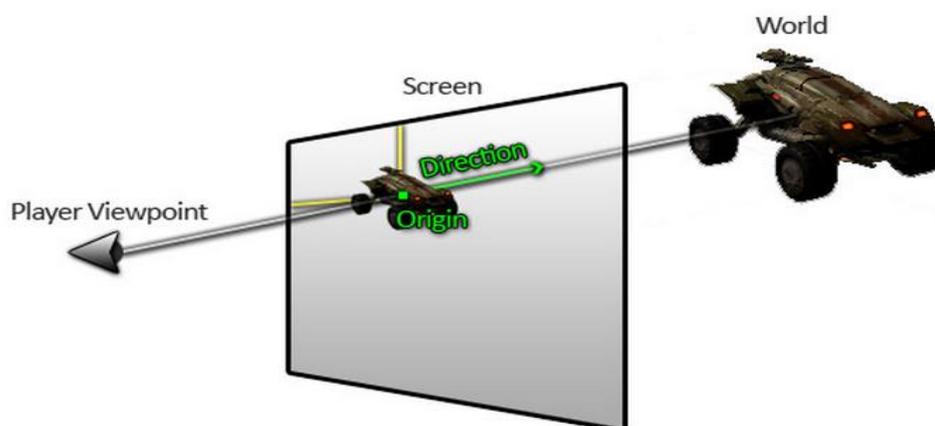


Figure 77: Transformation of the user's viewpoint from a 2D vector into 3D world-space origin and direction.

The eye's gaze data from the Viewpoint application are normalized between values zero and one. Thus, this data has to be converted in 2D screen coordinates. In order to do this, every value that is received in the application is multiplied by the total resolution of the screen which is 1024 x 2560 pixels. The respective code is the following:

```
function DrawHUD( HUD H )
{
...
//Left screen controller and Left eye
if ((FirstController == true) && (ActiveEye == LeftEye) && (EyeTrack.EyetrackerData.gazePoint.x<=0.5)){

    MousePosition.X = EyeTrack.EyetrackerData.gazePoint.x*1280*2;
    MousePosition.Y = EyeTrack.EyetrackerData.gazePoint.y*1024;
    hud.Canvas.DeProject(MousePosition, MouseWorldOrigin, MouseWorldDirection);
    ...
}

//Right screen controller and Right eye
if ((FirstController == false) &&(ActiveEye == RightEye) && (EyeTrack.EyetrackerData.gazePoint.x > 0.5)
){
    MousePosition.X = (EyeTrack.EyetrackerData.gazePoint.x-0.5)*1280*2;
    MousePosition.Y = EyeTrack.EyetrackerData.gazePoint.y*1024;
    hud.Canvas.DeProject(MousePosition, MouseWorldOrigin, MouseWorldDirection);
    ...
}
}
```

Afterwards, the call of the *Deprojection()* function with the correct 2D screen coordinates returns the *MouseWorldOrigin* and the *MouseWorldDirection* vectors. In order to determine what each participant is looking in the 3D game world the function *Trace()* is called. *Trace* is one of the most useful functions in UnrealScript. It casts a ray into the world and returns what it collides with first. *Trace* takes into account both this actor's collision properties and the collision properties of the objects *Trace* may hit. If *Trace()* does not hit anything it returns *NONE*; if *Trace()* hits level geometry (BSP) it returns the current *LevelInfo*; otherwise *Trace* returns a reference to the Actor it hit. *Trace* function takes as input *TraceEnd* which is the end of the line we want to trace and *TraceStart* is the beginning. Thus, in our case, *TraceEnd* is the *MouseWorldDirection* vector multiply by the value 65536 in order to expand this ray to infinite and *TraceStart* is the *MouseWorldOrigin* vector. If *Trace* hits something, *HitLocation*, *HitNormal*, and *Material* will be filled in with the location of the hit, the normal of the surface *Trace* hit, and the material *Trace* hit. Note that *HitLocation* is not exactly the location of trace hit, it is bounced back along the direction of trace a very small amount. This is the reason why we used blocking volumes as described in Chapter 3 instead of a collision mechanism. The use of blocking volumes gives the advantage to add an offset for each object collide area. The respective code is the following:

```

function recordParticipantAction()
{
...
    local vector loc, norm, end;
    local TraceHitInfo hitInfo;
    local Actor traceHit;

    //Expand "TraceEnd" to infinite to ensure that it collide with all the 3D world-space
    end =MouseWorldDirection* 65536.f;
    //Call the trace function
    traceHit = trace(loc, norm, end, MouseWorldOrigin, false,, hitInfo);
    //Bypass StaticMeshActor_3 which is the wall
    if ((traceHit == none) || ( tracehit.Name == 'StaticMeshActor_3'))
    {
        //    ClientMessage("Nothing found, try again.");
        traceHit = none;
    }
...
}

```

The following figure represents an example of use the function *Trace()* in combination with function *Deprojection()*.

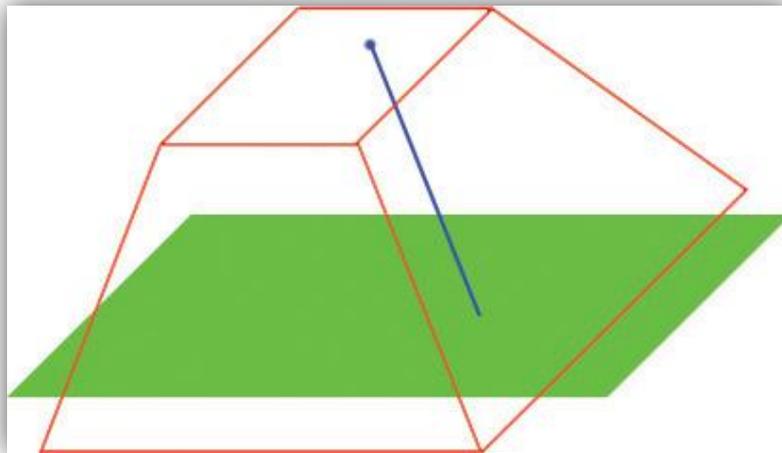
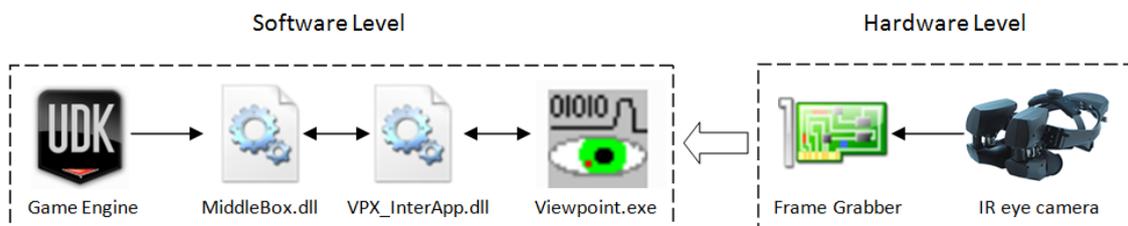


Figure 78: In the image above the green plane represents the world, the red frustum represents the world view frustum, the blue spot represents the 2D mouse position and the blue line represents the trace. The flat top of the frustum represents the screen (in this case, viewing the world over the top). The canvas deprojection function converts a 2D position into the world.

5.6 Communication with the Eye-tracking Device

One of the most important issues of the implementation was to find out a way to pass the data from the infrared camera of the Eye-tracking device within the application used for the experiments. Fortunately, the software ViewPoint EyeTracker® of the Arrington Research Company provides an SDK (software development kit) for third party applications. More specific SDK comes with a dynamic link library named *VPX_InterApp.DLL* which allows any third part application to interact with the Eye-tracking Device.

That was very convenient, however, there was an issue that had to be solved. The ViewPoint EyeTracker® allowed any third part application to interact with the Eye-tracking Device and needed a registration through the dynamic link library. After the registration, the application obtains a unique message identifier used by ViewPoint for inter-process communication. The problem arises is that as the name of *Unreal script* -the programming language for unreal game engine- discloses, it is a scripting language, meaning that the source code is already precompiled. As the source code is already precompiled it was impossible to write our code in order to register with the ViewPoint application. It was possible to use the functions of the *VPX_InterApp.DLL* by binding the library with the Unreal application as we did with the *SQL_Driver* library mentioned on Chapter 3. However, even if this was done without the registration required by the Viewpoint application, which couldn't be achieved through the Unreal application, this did not solve the problem. Eventually the solution was to write our dynamic link library named *MiddleBox.dll* which was registering with the ViewPoint application and bind our library with the Unreal engine instead of the *VPX_InterApp.DLL*. The whole communication including software and hardware level with the Eye-tracking device can be seen in the following figure.



79: Communication of Unreal Game Engine with the Eye-tracking Device.

As for the purposes of this thesis only the Software Level will be described in detail. As for the hardware level, the infrared camera was constantly capturing eye movement and a frame grabber converted this analog signal to a digital signal. The input signal is sampled at a specific rate and then the Eye-tracking software received each frame of the sampling. In spite of the hardware level, the software level is more complicated. It was required that a library was coded in order to register with the Viewpoint application. The role of the library *MiddleBox.dll* was to actually act as messenger between the Unreal game engine and the Viewpoint application. Thus, this library has functions that are called from inside the Unreal application and functions that are called from *VPX_InterApp.DLL* after the registration with the application. Below are the functions in the library that are called inside the Unreal application:

```
//Launch the Eye-tracking application
_declspec(dllexport) int LaunchApp(char* filename, char* arguments){
```

```

        return VPX_LaunchApp(filename, arguments); //Return a status code
    }

    //Register with the Viewpoint application
    _declspec(dllexport) int InsertMessageRequest(){
        //RegisterWithDllToReceiveMessages
        return VPX_InsertCallback( theCallbackFunction2 );
    }

    //Terminate Eye-tracking software
    _declspec(dllexport) void CloseApp(){
        VPX_RemoveCallback(theCallbackFunction2 ); //Remove window
        VPX_SendCommandString("QuitViewPoint"); //Close program
    }

    //Receive data from Infrared Camera
    _declspec(dllexport) eyetrackerStruct* getEyeData() {
        return &EyeTracker ;
    }
}

```

Below is the respectively Unreal script code of the class EyeTracker.uc which binds with the library *MiddleBox.dll* and calls the functions.

The unreal script code to bind with the library *MiddleBox.dll*

```

/*Class to read the input from eye tracker device using dynamic link library*/
class EyeTracker extends Actor notplaceable DLLBind(MiddleBox);

```

Declaration of the functions of the library *MiddleBox.dll*.

```

dllimport final function int LaunchApp(string fame,string args); //Function to launch the application
dllimport final function int InsertMessageRequest(); //RegisterWithDllToReceiveMessages
dllimport final function CloseApp();//Remove notification window and close eyetracker program
dllimport final function eyetrackerStruct getEyeData(); //Get new data from eye(s) depends on
monocular/binocular method

```

The function *StartEyeTracker()* is responsible for calling the appropriate functions to launch the Viewpoint application and then register with it.

```

function StartEyeTracker(){

    //Launch eyetracker program "location of the application", "arguments"
    errorcode1 = LaunchApp(filename,arguments);
    //Attempt to Register with Dll to ReceiveMessages
    errorcode2 = InsertMessageRequest();

    //Checks if everything is done correctly!
    if ((errorcode1 ==-1) && (errorcode2 !=0) )
        EyeTracker_State =1;
    else{
        WorldInfo.Game.Broadcast(self,"Command LaunchApp return code :"$errorcode1);
        WorldInfo.Game.Broadcast(self,"Command InsertMessageRequest return code :"$errorcode2);
    }
}

```



Figure 80: The status windows of Viewpoint application. In the red rectangular is the total number of registrations from third party applications.

Eventually after the successful registration with the Eye-tracking software, the library *MiddleBox.dll* defines a callback function which is managed by the library *VPX_InterApp.DLL*. Every new “fresh” data arrives in the Viewpoint application from the frame grabber and the application sends it to all the other programs are registered. Then the library *VPX_InterApp.DLL* calls every function that was defined as callback for each application, passing the “fresh” data. The “fresh” data is actually a message which can inform the library/application that new data is available about a change on eye’s gaze or on infrared camera or other useful data. The message which is meaningful and called by the Unreal application is the one which contains information about a change on the eye’s gaze. Every time a message identified that new data are available, two functions acquiring this data are called within library *MiddleBox.dll*. The first function named *VPX_GetGazePoint2()* is responsible to get the new eye’s gaze point data for the current active eye. The second function named *VPX_GetDataQuality2()* is responsible to get a “quality code” in order to validate the new data, as showed in the below table.

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 2: Quality code and the respectively information about the new data received.

Every time “fresh” data was received in the Unreal application and the quality code was five, four or three the data was discarded. Possible reasons for these errors are either a participant blinked his eye or an inappropriate eye movement or the infrared camera was instantly shaken due to a sharp spin of the HMD. Females appeared to produce more errors than males. The main reason for this was the cosmetic eye products that females were wearing during the experiment. Thus the calibration was not successful or was too difficult for the Viewpoint application to identify and lock the pupil of the participant. Error codes two and one are not given for the “Pupil Location” method.

The code of the callback function is shown here:

```
int theCallbackFunction2( int msg, int subMsg, int param1, int param2 )
{
    switch (msg)
    {
        case VPX_DAT_FRESH : //New eye's gaze point data
        {
            //Right eye by default
            if ( ( subMsg == EYE_A ) || (subMsg == EYE_B)) {
                // Eye to get data
                VPX_GetDataQuality2( subMsg, &EyeTracker.qualityCode );
                //Retrieves the quality code for the eye data
                VPX_GetGazePoint2(subMsg,&EyeTracker.gazePoint);
                EyeTracker.Eye = subMsg;
            }
            else
                ...
        }
        //Other messages
        ...
    }
    ...
}
```

Finally, after this data has been received, a function is called by the Unreal application in order to get this data and handle it, as shown on the following Unreal script code.

```
//Function which is called by the engine whenever time passes, more specific this function is executed
on very frame when the game updates
event Tick(float DeltaTime){
    ...
    //Eyetracker is runing normaly
    if (EyeTracker_State == 1){
        //Call the proper function from the library "MiddleBox.dll"
        to receive Eye-tracking device's data
        EyetrackerData = getEyeData();
        ...
    }
    ...
}
```

6 Chapter 6 – Experiments

6.1 Materials and Methods

This experiment was designed to explore gender differences in spatial navigation and spatial knowledge through a complex virtual environment. We aim to explore the effect of gender differences on object recognition performance after exposure to an immersive VE, in terms of both scene context and associated awareness states. In addition, investigations of possible correlations between memory recognition performance and eye gaze information will be conducted. The experimental methodology and procedure will be described in detail in the following sections.

6.1.1 Participants and Apparatus

Participants of the experiment were recruited from the postgraduate population of the Technical University of Crete. Participants were separated into two groups based on their gender. Groups were age-balanced and participants in all conditions were naive as to the purpose of the experiment. In total, forty participants were male and twenty-nine participants were female. 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 VEs were presented in stereo at SXGA resolution on a NVIS nVisor SX111 Head Mounted Display with a Field-of-View comprising 102 degrees horizontal and 64 degrees vertical. An InterSense InertiaCube3, three degrees of freedom tracker was utilized for rotation. The viewpoint was set in the middle of the virtual room and navigation was restricted to 360 degrees circle around that viewpoint (yaw), vertically (pitch) into -20 degrees downwards and +33 degrees upwards from the eye's height of the pawn. Participants sat on a swivel chair during exposure.

6.1.2 Visual Content

For the proposed experiment, we have developed a complete system based on the Unreal Development Kit which renders photorealistic illumination of high-quality synthetic scenes including full textures and materials over the 3d objects. The VE represented a square six by six meters room as shown in Figure 71. The radiosity-rendered space was divided in four zones including a dining area, a kitchen, an office area and a lounge area located on northeast, northwest, southwest and southeast side of the house, respectively. The space was populated by objects consistent as well as inconsistent with each zone's context. Three consistent objects and three inconsistent objects populated each zone resulting in twenty-four objects located in the scene overall, six in each zone. The VE is lit by only one light being a dominant directional light representing the sun. The sunlight was rendered to simulate midday lighting in terms of brightness and colour.

The between-subjects factor was 'Male' vs 'Female'. The within-subjects factor was 'object type' comprised of two levels: 'consistent' and 'inconsistent'. According to the training

group that they were assigned to, participants completed a memory recognition task including self-report of spatial awareness states and confidence rating for each recognition after exposure to the VE.

The scene consisted of basic modules such as walls, floors, ceilings, doors, etc (frame objects). According to Brewer & Treyens, 1981, “the room frame contains the information about rooms that one can be nearly certain about before encountering a particular room”. As mentioned, the scene was populated by three consistent objects as well as three inconsistent objects for each zone. The list of objects was assembled based on an initial pilot study which explored which objects were expected to be found in each area and which were not [Zotos et al. 2009]. According to this study, twenty-five participants ranked the objects on the list. The consistency of each item was rated on a scale from one to six according to whether each object was expected to be found in each area or not, with six being the most expected, and one being the least. Based on these ratings, consistent objects were selected from the high end of the scale, and the inconsistent ones from the low end. The objects were distributed over locations and participants were required to select from a recognition list provided which object was present in each location. All twenty-four objects positioned in the house scene as follow:

Consistent objects in house			
Kitchen	Teapot	Wine	Plate
Dining	Candlestick	Flower vase	Tablecloth
Office	Office basket	Book	Lighter
Lounge	Coats hanger	Television	Coffee mug

Table 3: Consistent objects existing in each zone of the house.

Inconsistent objects in house			
Kitchen	Basketball	Pen	Toothbrush
Dining	Clothe hanger	American football	Screwdriver
Office	Sword	Tennis racquet	Pan
Lounge	Coffeepot	Cowboy hat	Hammer

Table 4: Inconsistent objects existing in each zone of the house.



Figure 81: The experimental scene (side-view).

Adding to previous research utilizing similar methodologies, in this project a system to track and record eye movement was utilized. The system also recorded each participant's head movement, which was monitored through software as exposure time may affect memory encoding. This information is at a high enough resolution to be useful in determining the time spent looking at each object, the amount and location of participants' idle time and other valuable data. Idle time is defined as the time during which participants' viewpoint or view direction doesn't change. Such measurements were significant in order to meaningfully compare memory recognition scores and confidence ratings across conditions as participants might not have distributed exposure time evenly while observing the experimental scene. Participants who, for instance, spent one hundred and twenty seconds of exposure time observing just one wall of the room were excluded from the statistical analysis. Therefore, the goal of monitoring idle time was to ensure that idle time of participants across conditions as well as idle time for each participant observing sections of the room would be similar. The measurement rate was 20Hz, providing twenty measurements every second across all conditions.

6.1.3 Experimental Procedure

The experiment was set to be completed in three stages by each participant. Initially, a preliminary training phase took place requiring participants to wear the HMD device. During this stage, all appropriate adjustments were conducted accordingly for each individual. After participants familiarized themselves with the device and the stereoscopic VE then the second stage of the experiment was initiated which was the main phase of the experiment. The third stage took effect when participants finished the main experiment, requiring them to complete an online memory recognition questionnaire.

First stage – Training

During the first stage, appropriate adjustments were conducted accordingly for each individual. Initially, personal information individual to each participant were inserted to the database file. Secondly, participants were instructed to follow the Eye Dominance test in order to determine which one of their eyes was the dominant one. Thirdly, participants wore the HMD and a sample stereoscopic picture was displayed on the HMD as shown in the following figure.



Figure 82: The sample image projected on the HMD.

Based on that scene participants adjusted accordingly the HMD to feel more comfortable and find the best position for the stereoscopic view. In some cases, the Inter Pupillary Distance (IPD) which was preset in a standard position, had to be changed in order to match the participant’s individual IPD. Adjustments to the IPD were conducted in order to avoid participants from viewing dissimilar imagery through their two eyes. After participants felt comfortable with the HMD and having a good sense of the stereoscopic view without any side-effects, the infrared camera for the dominant eye was fixed in a static position in order to start the calibration of the eye’s gaze as shown in the following figure. Participants were instructed how to conduct the calibration process which was described in Chapter 3. In most cases, more than one calibration was need to achieve the desirable results.

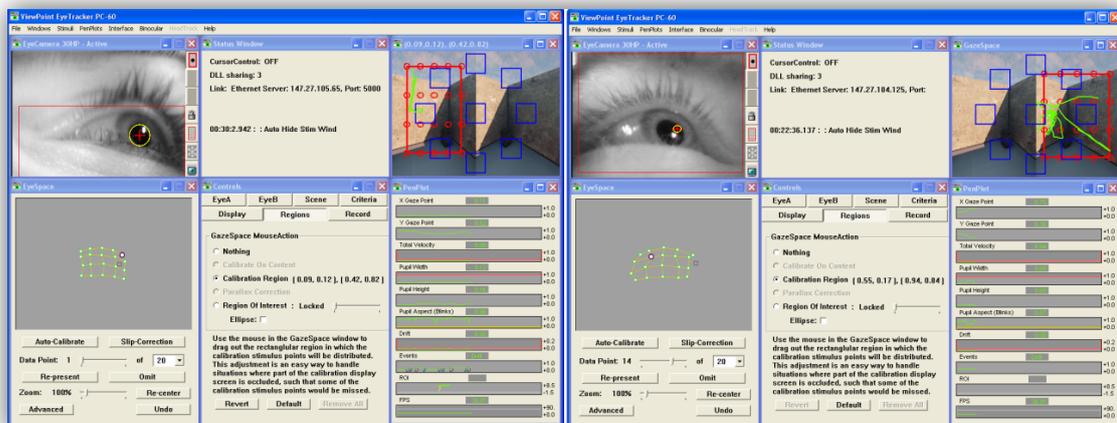


Figure 83: Calibration for eye-gaze on left dominant eye (left picture) and right dominant eye (right picture).

After all the adjustments, the application for the experiment was setup and a training VE was loaded on the HMD. Participants were instructed to freely navigate and look around the synthetic scene using the head-tracker, as much time as they wanted, in order to get familiar with the VE and the HMD. The participants viewed a virtual room, comprising of walls, floor, but without a ceiling, so the sky was visible. Inside the room, there were primitive objects such as box, sphere, cylinder etc. (Figure 84).

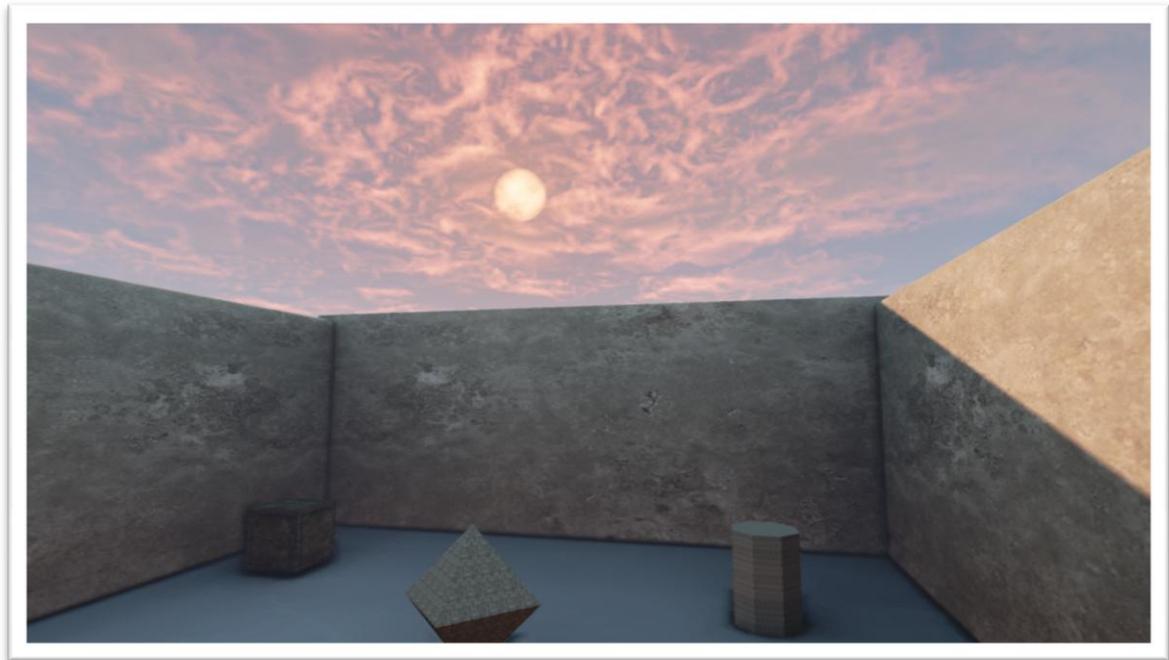


Figure 84: The training Virtual Environment.

Second stage – Main Experiment

When the participants felt comfortable viewing with the settings of the test scene, the next stage of the experiment was loaded on the HMD. Prior to that, participants were told that they didn't have to complete a specific task in the VE. They were instructed to just look around and explore all the zones of the house as they wished. Furthermore, participants were told that the purpose of the main experiment was to test the new technology devices that our lab had recently acquired. The scenario for the main experiment was attentively chosen in order to prevent participants suspect that after the end of the main experiment they have to complete a memory task.

Participants were exposed to an interactive simulation of a synthetic scene. The exposure time was one hundred and twenty seconds in each condition. The exposure time was defined after a series of pilot studies which aimed to identify the exposure time while ensuring that no floor or ceiling effects were observed, e.g. the task being too easy or too difficult. The final pilot studies aimed to finalize the experimental design and provide preliminary insight.

Third stage - Memory Recognition Task

After the end of the experiment, each participant was informed that they were about to complete a memory task by completing an on-line questionnaire. The participant was led in a quiet place without distractions and was given a four pages leaflet; each page containing one of the four zones of the house respectively. In the pictures of the leaflet the six objects of each zone were replaced by a red numbered mark as shown in Figures 85, 86, 87 and 88 respectively.

The on-line questionnaire comprised of four pages, each one representing the lounge, office, kitchen and dining zone respectively, as shown in the Appendix C. Participants were required to select which object they considered they saw during their previous exposure to the scene in each marked position, selecting objects from an on screen object recognition list as well as one out of five levels of confidence: No confidence, Low confidence, Moderate confidence, Confident, Certain, and two choices of awareness states: Remember and Know (Type A and Type B respectively). A recognition list was devised including a list of objects per scene zone. Each zone included in random order the six present objects as well as six absent objects (three inconsistent and three consistent) in each zone. The four lists include a total of 48 objects. The four pages of the leaflet are shown in Figures 85, 86, 87, and 88 in the next four pages respectively.

Prior to the memory recognition task, awareness states were explained to the participants in the following terms:

- TYPE A means that you can recall specific details. For example, you can visualize clearly the object in the room in your head, in that particular location. You virtually ‘see’ again elements of the room in your mind, or you recollect other specific information about when you saw it.
- TYPE B means that you just ‘know’ the correct answer and the alternative you have selected just ‘stood out’ from the choices available. In this case you can’t visualize the specific image or information in your mind.

Chapter 6 – Experiments

The first page of the leaflet contains the lounge area with the red marks numbered from one to six as shown in Figure 85.

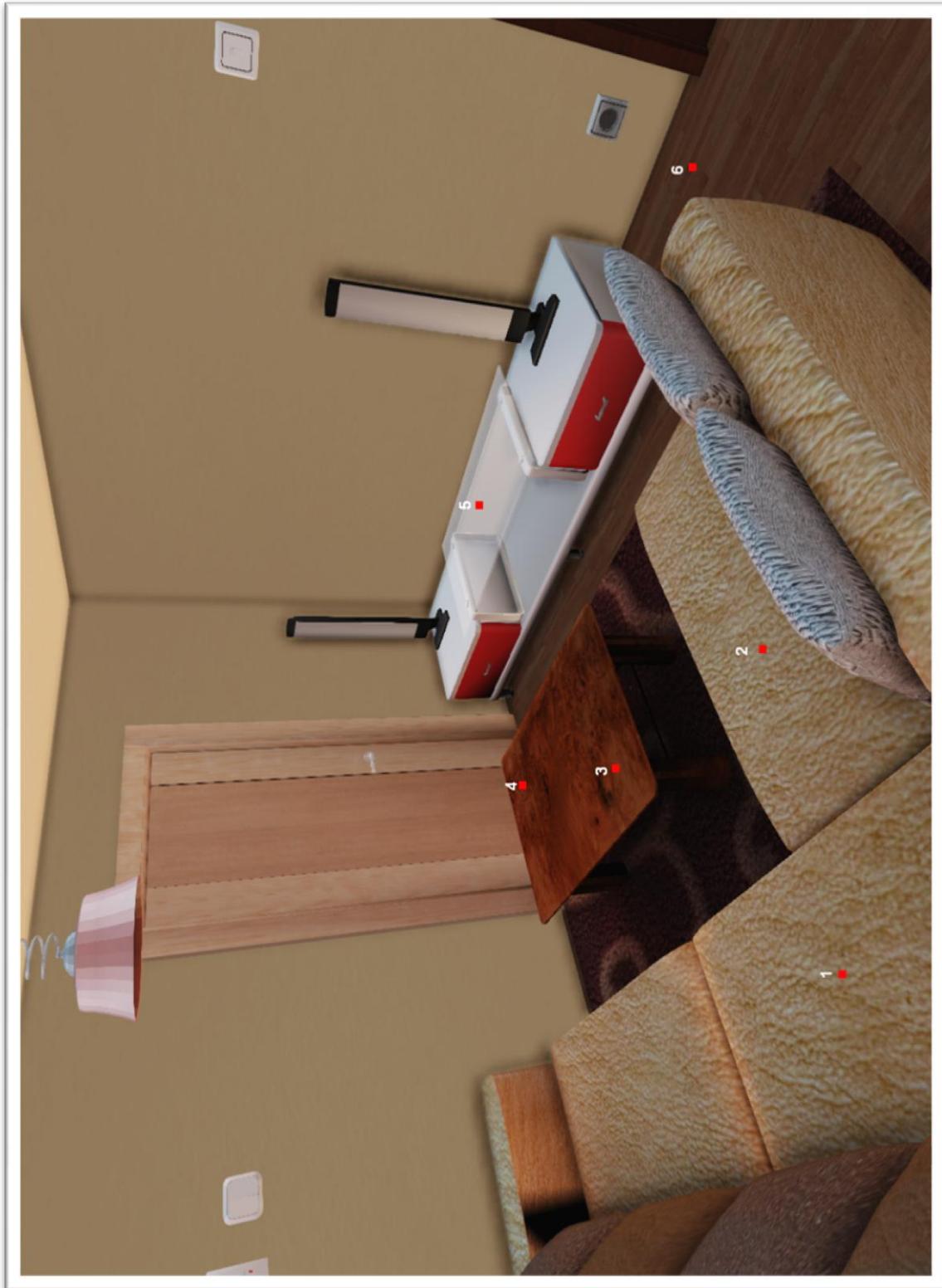


Figure 85: First page of the leaflet containing the Lounge Area.

Chapter 6 – Experiments

The second page of the leaflet contains the office area with the red marks numbered from seven to twelve as shown in Figure 86.



Figure 86: Second page of the leaflet containing the Office Area.

Chapter 6 – Experiments

The third page of the leaflet contains the kitchen area with the red marks numbered from thirteen to eighteen as shown in Figure 87.



Figure 87: Third page of the leaflet containing the Kitchen Area.

Chapter 6 – Experiments

The fourth page of the leaflet contains the dining area with the red marks numbered from nineteen to twenty-four as shown in Figure 88.



Figure 88: Fourth page of the leaflet containing the Dining Area.

6.1.4 Simulator Sickness

Despite the fact that the NVIS nVisor SX111 Head Mounted Display used in the experiments is currently state-of-the-art in the market, there still exist certain minor weaknesses related to its use. In particular, simulator sickness which is a potential side effect of all HMDs in general, has been observed during the experiment presented here too.

Participants reported several symptoms related to simulator sickness such as fatigue, headache, dizziness, visual discomfort, and nausea. There also exist other indirect effects of VEs on the visual system such as eyestrain, changes in binocular vision and visual acuity, balance, nausea, and motion sickness. Participants experienced the aforementioned symptoms to varying degrees. Various articles exist in related literature focusing on possible causes of simulator sickness such as system latency [DiZio and Lackner 1997, Cobb et al. 1999], limited Field-of-View [DiZio and Lackner 1997], Image scale factor [Draper et al. 2001], etc.

Apart from the previous factors that provoke simulator sickness, other aspects of the HMD that contribute to participant discomfort exist. The HMD itself weighted 1.3 Kg making participants uncomfortable during the experiment. Additionally, as a result of improper adjustment of the Interpupillary Distance (IPD) participants perceive dissimilar imagery from their eyes. Moreover, when immersed in artificial environments for example by using HMDs and particularly in partial overlap systems where the monocular fields of view are narrower than in natural viewing, image borders are accompanied by a perceptual phenomenon that in virtual reality literature has come to be known as luning. Luning refers to a peculiarity appearing in visual perception in the monocular areas of partial binocular overlap displays characterized by a subjective darkening of the visual field. Nevertheless, the experiments reported here were conducted without any participant interrupting the procedure because of simulator sickness.

6.2 Statistical Analysis

This section presents the basic statistical principles employed in order to analyze the acquired memory recognition self-report.

6.2.1 Analysis of Variance

ANalysis Of VAriance: ANOVA procedures are powerful parametric methods for testing the significance of the differences between sample means where more than two conditions are used, or even when several independent variables are involved [Coolican 1999]. ANOVA is used to compare the variance between the two groups with the variability within each of the groups. This comparison is in the form of a ratio known as the F test. A high value for F indicates a strong effect, i.e. the variance between groups is higher than the variance within the groups. The strength of the effect is given by the p value. The p value represents the probability that there is no between groups variance. This is called the **null hypothesis** and is disproved **if a value of p below 0.05 is returned.**

6.3 Results and Discussion

Memory recognition performance:

Memory recognition performance was measured by counting the number of correct positions of objects (out of a possible 24). Prior probabilities were obtained by calculating the proportions of correct answers falling in each of the two memory awareness categories for each participant.

	Males (n=40)		Females (n=29)	
	Consistent	Inconsistent	Consistent	Inconsistent
Total correct (out of 24)	5.15 (1.75)	7.10 (2.35)	6.97 (1.68)	8.41 (1.66)

Table 5: Number of correct responses and standard deviations.

Correct recognition scores of objects in each location were analyzed using a 2x2 mixed analysis of variance (ANOVA) with gender (male, female) entered as a between subjects variable and the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable (Table 1). A large main effect of gender was identified ($F(1,67)=16.65$, $p<0.001$, partial eta-squared=0.20). Female participants correctly recognized more objects in their locations (Mean = 7.69) compared to the male participants (Mean = 6.13). A large main effect of context consistency was also identified ($F(1,67)=40.77$, $p<0.001$, partial eta-squared=0.38). More inconsistent objects were correctly recognized in their locations (Mean = 7.76) than consistent objects (Mean = 6.05). No interaction between gender and context consistency revealed ($F(1,67)=0.89$, $p>.05$).

	Males (n=40)		Females (n=29)	
	Consistent	Inconsistent	Consistent	Inconsistent
Type A (remember)	.25 (.11)	.43 (.15)	.23 (.11)	.39 (.12)
Type B (know)	.17 (.13)	.14 (.15)	.22 (.10)	.16 (.13)

Table 6: Proportion of correct responses and standard deviations.

The proportion of correct responses (displayed in Table 2) was analyzed with separate 2x2 mixed ANOVAs for each awareness state (Table 2). Gender (male, female) was entered as a between subjects variable, with the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable. A minimum alpha level of .05 was used throughout the analyses. No reliable main effects of gender were found for either Memory A awareness states ($F(1,67)=1.81, p>.05$) or Memory B awareness states ($F(1,67)=1.49, p>.05$). Similarly, no interactions were found between gender and the context consistency of the objects for either Memory A awareness states ($F(1,67)=0.22, p>.05$) or Memory B awareness states ($F(1,67)=0.52, p>.05$). However, there was a large main effect of context consistency on Memory A awareness states ($F(1,67)=66.30, p<.001, \text{partial } \eta^2 = 0.50$) and a small main effect of context consistency on Memory B awareness states ($F(1,67)=4.81, p<.05, \text{partial } \eta^2 = 0.067$). When objects were correctly recognized in the correct location, a higher proportion of correct responses were reported as Memory A awareness state (remember) with inconsistent objects (Mean = .41) compared to consistent objects (Mean = .24). Conversely, participants reported a higher proportion of correct responses were reported as Memory B awareness states with consistent objects (Mean = .19) compared to inconsistent objects (Mean = .15). The analysis was repeated following the removal of 11 male participants tested, in order to create equal group sizes. The exact same pattern of results was found ($N=29$).

	Males (n=40)		Females (n=29)	
	Consistent	Inconsistent	Consistent	Inconsistent
Confidence (5-point scale)	3.74 (0.75)	4.19 (0.59)	3.55 (0.66)	4.14 (0.58)

Table 7: Mean confidence rating and standard deviation as a function of Gender (Males, Females) and context consistency (consistent, inconsistent).

The average confidence ratings for objects correctly identified in the correct location were analyzed using a 2x2 mixed ANOVA. Gender (male, female) was entered as a between subjects variable, with the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable. Means are displayed in table 2. No reliable main effect of gender was found ($F(1,67)=0.82, p>.05$), and there was no interaction between context consistency and gender ($F(1,67)=0.54, p>.05$). However, there was a reliable main effect of context consistency ($F(1,67)=32.12, p<.05, \text{partial } \eta^2 = 0.32$). On average, participants reported more confidence when correctly identifying objects that were inconsistent with the context ($M = 4.17, SD = 0.60$) compared to participants reported their confidence when correctly identifying the context consistent objects ($M = 3.65, SD = 0.72$).

The analyses highlighted two important effects. The first was a clear influence of the context consistency of objects on the type of memorial experience that participants had. Vivid and contextually detailed memorial experiences were reported more often for objects that

were inconsistent with the context of the scene (e.g. a toothbrush in the kitchen area). Conversely less contextually detailed feelings of knowing were reported more often for objects that were consistent with the context of the scene (e.g. a book in the office area). This indicates that context consistency has an important influence on the memorial experiences of users in such environments. Secondly, there was a clear influence of gender on the number of objects correctly recognized in their correct location. Female participants outperformed male participants through correctly recognizing more of the objects in the correct locations. The gender of users is therefore an important consideration in terms of memory for objects and their locations within such environments.

Eye-tracking Results:

Although the total number of participant's was seventy, certain participant's records were described as outliers and were not included in the analysis. More specifically eight participant's records were removed from the dataset due to corrupted or invalid data. Additionally, six participant's records were removed from the dataset in which zero fixations were made to either consistent or inconsistent objects. Fixation (visual) is the maintaining of the gaze in a constant direction. Fixation was considered the time greater than 100ms a participant looks at an object in the virtual scene. Kolmogorov-Smirnov tests indicated that the means were not normally distributed ($p < .05$). Although ANOVA assumes the data have a (normal) Gaussian distribution, for the purposes of these analyses it was assumed to be sufficiently robust to handle this lack of normality. To reflect this additional uncertainty a minimum alpha level of .01 was used throughout the analyses to judge a reliable difference, and to reduce the probability of a type I error.

Males (n=33)		Females (n=23)	
Consistent	Inconsistent	Consistent	Inconsistent
27.00 (13.26)	10.15 (8.98)	24.00 (11.61)	12.74 (12.07)

Table 8: Total Number of Fixations.

Table 8 shows the number of fixations on objects in each category. The total number of fixations, on objects within the scene, were analyzed using a 2x2 mixed ANOVA. Gender (male, female) was entered as a between subjects variable, with the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable. Means are displayed in table 1. No reliable main effects of gender were found ($F(1,54)=0.005$, $p > .01$), and there was no interaction between context consistency and gender ($F(1,54)=4.41$, $p > .01$). However, there was a reliable main effect of context consistency ($F(1,54)=111.67$, $p < .0001$, partial eta-squared = 0.67). On average, context consistent objects were fixated on more often ($M = 25.76$, $SD = 12.59$) than context inconsistent objects ($M = 11.21$, $SD = 10.34$).

Males (n=33)		Females (n=23)	
Consistent	Inconsistent	Consistent	Inconsistent
9.88 (5.58)	3.12 (4.13)	7.92 (4.12)	3.33 (3.57)

Table 9: Total Time Spent Fixating (seconds).

The total time spent fixating on objects was analyzed using a 2x2 mixed ANOVA. Gender (male, female) was entered as a between subjects variable, with the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable. Means are displayed in table 2. No reliable main effects of gender were found ($F(1,54)=0.684$, $p>.01$), and there was no interaction between context consistency and gender ($F(1,54)=3.16$, $p>.01$). However, there was a reliable main effect of context consistency ($F(1,54)=85.93$, $p<.0001$, partial eta-squared = 0.61). On average, context consistent objects were fixated on for longer overall ($M = 9.08$, $SD = 5.09$) than context inconsistent objects ($M = 3.21$, $SD = 3.88$).

Males (n=33)		Females (n=23)	
Consistent	Inconsistent	Consistent	Inconsistent
2.96 (0.75)	4.22 (1.71)	3.13 (0.68)	4.41 (1.28)

Table 10: Mean Time Spent Fixating (seconds).

The mean time spent fixating on each object in each category was analyzed using a 2x2 mixed ANOVA. Gender (male, female) was entered as a between subjects variable, with the context consistency of the objects (consistent, inconsistent) entered as a within subjects variable. Means are displayed in table 3. No reliable main effects of gender were found ($F(1,54)=0.620$, $p>.01$), and there was no interaction between context consistency and gender ($F(1,54)=0.003$, $p>.01$). However, there was a reliable main effect of context consistency ($F(1,54)=28.63$, $p<.0001$, partial eta-squared = 0.35). When fixated on, context inconsistent objects were fixated on for longer ($M = 4.30$, $SD = 1.54$) than context consistent objects ($M = 3.03$, $SD = 0.72$).

In summary, context consistent objects were fixated on more often, and the total time spent fixating on these objects was greater. However, when fixated on, inconsistent objects were fixated for longer. The gender of the participants did not influence this effect.

7 Chapter 7 – Conclusions

In this thesis, an interactive stereoscopic 3D application was developed and displayed on a HMD. The application was used to conduct an experiment examining gender differences in spatial navigation, memory performance and spatial awareness. In addition, the application was carefully designed and implemented in a way to be compatible with innovative Eye-tracking technology and employ a more modern approach to such psychophysical experiments. Our intention was to correlate responses from classic post-exposure memory task questionnaires that participants completed after the experiment with the data collected from an Eye-tracker during their navigation in the IVE. Based on this analysis we aimed to determine how gender affects memory performance and find the factors that contribute to such an effect.

From a technical point of view the actual setup of the experiment was a great challenge both because of the design requirements of the interactive 3D scene as well as the implementation of a formal experimental paradigm. The VE had to be of high fidelity, i.e. photorealistically rendered and interactive in order to communicate the spatial cues necessary to induce simulation of real-world spatial awareness. Another challenge was the use of state of the art Eye-tracking technology. As this technology was recently acquired and never before used in our University, technical expertise was inexistent. Few academic research groups have already successfully used it in an integrated set-up involving a HMD for conducting experiments and there is limited literature around its use. Therefore, there were several issues involving the HMD with embedded eye tracking that had to be resolved.

The Unreal Development Kit was used to build the application as it provides the necessary tools and programming framework (Unreal Script) in order to create an interactively manipulated photorealistically-rendered synthetic scene, as well as the infrastructure to link the 3D application with the Eye-tracker. The design of the stereoscopic 3D framework was adapted to address the challenges arising from the strict experimental protocol, including the creation of the synthetic scene and the User Interface implemented as an Adobe Flash application which controlled the phases of the experiment. Beyond the implementation of the UI, the application had to comply with all the technical constraints introduced by the partial overlap method system employed by the HMD and match certain stereoscopic parameters such as resolution and Field Of View in order for the rendering to be displayed correctly.

The strict experimental protocol imposed time limits that the framework had to meet, by specifying timers in order to generate events and corresponding actions as well as simultaneously logging every action taking place in the VE in a database. This was done in order to be able to understand and analyze the data gathered later on. A dynamic library was written to manage the communication with the Eye-tracker i.e. act as a messenger between the Unreal application and the Eye-tracking software. Finally the data from the online questionnaires that participants completed were acquired using an online web form created using Google services.

7.1 Main contributions

The effectiveness of VEs as training mediums has often been linked to successful spatial awareness which, in turn, induces the sense of presence reported by users of those VEs. Presence is defined as the subjective experience of being in one place or environment, even when one is physically situated in another. It is argued that VEs that generate a higher feeling of presence would result in transfer equivalent to real-world situations, which would be extremely useful for VEs such as training simulators. Immersive Virtual Environments provoke a higher feeling of presence as they allow users to have an experience that very much looks and feels "real."

Spatial awareness is significant for human performance efficiency of training tasks as they require spatial knowledge of an environment. A central research issue therefore for real-time VE applications for training is how participants mentally represent an interactive computer graphics world and how their recognition and memory of such worlds correspond to real world conditions.

The experimental methodology presented focuses upon exploring gender differences in object-location recognition memory and its associated awareness states while immersed in a radiosity-rendered synthetic simulation of a complex scene. Gender specific memory performance differences were studied from responses to a memory recognition task as well as from examining the eye's gaze data during the navigation in the IVE. This study allowed us to investigate the effect of gender on both the accuracy and the phenomenological aspects of object memories acquired in an IVE.

The results derived from the participants' questionnaires responses during the experiment indicated that clear differences exist amongst genders. Specifically, females were found to be more accurate in identifying objects' position in the house scene in comparison to males. It became evident that each gender had different memorial experience based on the context consistency of objects. Vivid and contextually detailed memorial experiences were reported more often for objects that were inconsistent with the context of the scene; on the other hand less contextually detailed feelings of knowing were reported more often for objects that were consistent with the context of the scene. This indicates –as mentioned above- that context consistency has an important effect on the memorial experiences of users in such environments.

Preliminary results derived from the data acquired from Eye-tracking showed that participants tend to look more often at context consistent objects rather than inconsistent objects. In addition, the total time spent fixating on these objects was greater. However, when fixated on, inconsistent objects were fixated for longer. Future work and more extensive analysis of the Eye-tracking data could identify potential gender differences of attentional patterns in relation to specific awareness states.

8 References – Bibliography

- Ashdown 2004. Ian Ashdown, *Radiosity Bibliography* [www], <http://tralvex.com/pub/rover/abs-ian0.htm> (Accessed March 2004)
- Baddeley 1997 Baddeley, A. (1997). *Human Memory, Theory and Practice*. Psychology Press.
- Birn 2000 Birn, J. 2000. *[digital] Lighting and Rendering*. New Riders.
- Bouknight 1970 Bouknight J. 1970. "A procedure for generation of three dimensional half-toned computer graphics presentations." *Communications of the ACM*, Vol. 13 (9) 527–536
- Brewer and Treyens 1981 Brewer W.F. and Treyens J.C. (1981). *Role of Schemata in Memory for places*.
- Chai et al. 2009 Chai, X., Jacobs, L. (2009). Sex Differences in Directional Cue Use in a Virtual Landscape. *Behavioral Neuroscience*, 123 (2) 276–283.
- Cobb et al. 1999 Cobb, S., Nichols, S., Ramsey, A., & Wilson, J. (1999). Virtual reality-induced symptoms and effects (VRISE). *Presence: Teleoperators & Virtual Environments*, 8, 169-186.
- Cohen and Greenberg 1985 Cohen M.F., and Greenberg D.P. 1985. "The Hemi-Cube: A radiosity solution for complex environments." In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 31-40.
- Cohen et al. 1988 Cohen M.F., Chen S.E., Wallace J.R., and Greenberg D.P. 1988. "A progressive refinement approach to fast radiosity image generation." In John Dill, editor, *Computer Graphics (SIGGRAPH 1988 Proceedings)*, volume 22, pages 75–84.
- Conway et al. 1997 CONWAY, M. A., et AL. 1997. Changes in Memory Awareness During Learning: The Acquisition of Knowledge by Psychology Undergraduates. *Journal of Experimental Psychology : General*, 126(4), 393 – 413.
- Coolican 1999 Coolican, H. 1999. *Research Methods and Statistics in Psychology*, Hodder & Stoughton Educational, U.K
- Cooper 1995 Cooper, L.A. (1995). Varieties of Visual Representations: How are we to analyse the concept of mental image? *Neuropsychologia*, Vol. 33(11), 1575-1582 Elsevier Science Ltd.
- Desney et al. 2003 Desney S. Tan, Mary Czerwinski and George Robertson (2003). *Women Go With the (Optical) Flow*. Microsoft Research.
- DiZio and Lackner 1997 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.
- Drapper et al. 2001 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 .
- Ferwerda 2003 Ferwerda, J.A. 2003. "Three varieties of realism in computer graphics." In IS&T/SPIE Conference on Human Vision and Electronic Imaging VIII, SPIE Proceedings Vol. 5007, 290-297
- Gardiner 2001 Gardiner, J.M. (2000). Remembering and Knowing. In the E. Tulving and F.I.M. Craik (Eds.) *Oxford Handbook on Memory*. Oxford University Press.
- Gibson and Hubbard 1997 Gibson, S., and Hubbard R.J. 1997. "Perceptually Driven Radiosity." *Computer Graphics Forum*, 16 (2): 129-140.
- Glassner 1984 Glassner, A.S. 1984. "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics & Applications*, Vol.4, No. 10, pp 15-22.

References – Bibliography

- Glassner 2000 Glassner, A.S. 2000. *An Introduction to Ray tracing.* Morgan Kaufmann.
- Goral et al. 1984. Goral C.M., Torrance K.E., Greenberg D.P., and Battaile B. 1984. "Modeling the interaction of light between diffuse surfaces." In proceedings of the *11th Annual Conference on Computer Graphics and Interactive Techniques*. Vol. 18 (3) 212-222
- Gouraud 1971 Gouraud H., "Continuous shading of curved surfaces", *IEEE Transactions on Computers*, 20(6): 623-628.
- Gregg and Gardiner 1994 Gregg, V.H. & Gardiner, J.M. (1994). Recognition Memory and Awareness: A Large Effect of Study-Test Modalities on 'Know' Responses Following a Highly Perceptual Orienting Task. *European Journal of Cognitive Psychology*, 6(2), 131-147.
- Hall and Greenberg 1983 Hall R.A., and Greenberg D.P. 1983. "A Testbed for Realistic Image Synthesis." *IEEE Computer Graphics and Applications*, Vol. 3, No. 8. pp. 10-19.
- Held and Durlach 1992 Held, R. and Durlach, N. (1992), *Telepresence, Presence: Teleoperators and Virtual Environments*, 1992 1(1), 109–112.
- Jacobs et al. 2003 Jacobs, L.F. & Schenk, F. (2003) Unpacking the cognitive map: the parallel map theory of hippocampal function. *Psychol. Rev.*, 110, 285–315.
- Jon Radoff 2008 Jon Radoff, Anatomy of an MMORPG, <http://radoff.com/blog/2008/08/22/anatomy-of-an-mmorpg>
- Kajiya 1986 Kajiya, J.T. 1986. "The Rendering Equation." *ACM SIGGRAPH 1986 Conference Proceedings*, volume 20,143-150.
- Kajiya 1990 Kajiya, J.T. (1990). Radiometry and Photometry for Computer Graphics. *Proc. of ACM SIGGRAPH 1990 Advanced Topics in Ray Tracing course notes*.
- Kim et al. 2007 Kim, B., Lee, S., Lee, J. (2007). Gender Differences in Spatial Navigation. *World Academy of Science, Engineering and Technology*.
- Koriat & Goldsmith 1994 KORIAS, A., GOLDSMITH, M. 1994. Memory in Naturalistic and Laboratory Contexts: Distinguishing the accuracy oriented and quantity oriented approaches to memory assessment. *Journal of Experimental Psychology: General*, 123, 297-315.
- Kosslyn et al. 1995 Kosslyn, S.M., Behrmann, M., Jeannerod, M. (1995) The Cognitive Neuroscience of Mental Imagery. *Neuropsychologia*, 33(11), 1335-1344. ElsevierScience Ltd.
- Kuipers 1975 KUIPERS, B.J. 1975. A frame for frames: Representing knowledge for recognition. In D.G. Bobrow & A. Collins (Eds.), *Representation and Understanding: Studies in Cognitive Science*. New York: Academic Press, 1975.
- Languénou et al. 1992. Languénou E., Bouatouch K., and Tellier P. 1992. "An adaptive Discretization Method for Radiosity." *Computer Graphics Forum* 11(3): 205-216.
- Lampinen et al. 2001 LAMPINEN, J., COPELAND, S., NEUSCHATZ, J. 2001. Recollections of things schematic: rooms schemas revisited. *Cognition*, 27, 1211-1222.
- Laria et al. Laria, G., Palermo, L., Committeri, G., Barton, J.J., (2009b). Age differences in the formation and use of cognitive maps. *Behavioral Brain Res.* 196, 187–191.
- Lauren et al. 2005 Lauren J. Levy, Robert S. Astur, Karyn M. Frick (2005). Men and Women Differ in Object Memory but Not Performance of a Virtual Radial Maze. *Behavioral* Vol. 119, No. 4, 853–862
- Lovden 2005 Lovden, M., Schellenbach, M., Grossman-Hutter, B., Kruger, A., Lindenberger, U., (2005). Environmental topography and postural

References – Bibliography

- control demands shape aging-associated decrements in spatial navigation performance. *Psychol. Aging* 20, 683–694.
- Lovden et al. 2007 Lövdén, M., Herlitz, A., Schellenbach, M., Grossman-Hutter, B., Krüger, A. & Lindenberger, U. (2007). Quantitative and qualitative sex differences in spatial navigation. *Scandinavian Journal of Psychology*, 48, 353–358.
- Lischinski 2004 Lischinski, D., Radiosity, (Lecture notes) <http://www.cs.huji.ac.il/~danix/advanced/notes3.pdf>
- Mania et al. 2003 Mania K., Troschianko T., Hawkes R., A. Chalmers 2003. Fidelity metrics for virtual environment simulations based on spatial memory awareness states. *Presence, Teleoperators and Virtual Environments*, 12(3), 296-310. MIT Press.
- Mania et al. 2005 The Effect of Memory Schemas on Object Recognition in Virtual Environments. *Presence Teleoperators and Virtual Environments*, MIT Press
- Mania et al. 2006 Mania K., Wooldridge D., Coxon M., Robinson A. 2006. The effect of visual and interaction fidelity on spatial cognition in immersive virtual environments. *IEEE Transactions on Visualization and Computer Graphics journal*, 12(3): 396-404.
- Mania et al. 2010 Mania K., Badariah S., Coxon M. 2010. Cognitive transfer of training from immersive virtual environments to reality. *ACM Transactions on Applied Perception*, ACM Press, 7(2), 9:1-9:14.
- McCabe et al. 2009 McCabe, D.P., & Geraci, L.D. (2009). The influence of instructions and terminology on the accuracy of remember-know judgements. *Consciousness and Cognition*, 18, 401-413.
- ME 2004 Molecular Expressions, *Physics of light And colour* [www],<http://micro.magnet.fsu.edu/primer/java/reflection/specular/>
- Minsky 1975 Minsky, M. 1975. A framework for representing knowledge. In P.H. Winston (Ed.), *The Psychology of Computer Vision*. McGraw-Hill.
- Moffat et al. Moffat, S.D., Resnick, S.M., (2002). Effects of age on virtual environment place navigation and allocentric cognitive mapping. *Behav. Neurosci.* 116, 851–859.
- Moss and Muth 2008 Simulator Sickness during Head Mounted Display (HMD) of Real World Video Captured Scenes *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* September 2008 52: 1631-1634,
- Muller et al. 2008 Mueller, Sven.C. Jackson , Carl.P.T. Skelton, Ron.W (2008). Sex differences in a virtual water maze: an eye tracking and pupillometry study. Sex differences in eye movements during spatial navigation.
- NightLase 2004 NightLase, *Light and optics theories and principles* [www], <http://www.nightlase.com.au/education/optics/diffraction.htm>
- Noah et al. 2003 Noah J. Sandstrom , Jordy Kaufman, Scott A. Huettel (1998) , Males and females use different distal cues in a virtual environment navigation task , *Cognitive Brain Research* 6 1998 351–360.
- Nusselt 1928 Nusselt, W. 1928. “Gräpische Bestimmung des Winkelverhältnisses bei der Warmestrahlung,” *Zeitschrift des Vereines Deutscher Ingenieure* 72(20):673.
- Palmer 1999 Palmer, S.E. 1999. *Vision Science – Photons to Phenomenology*. Massachusetts Institute of Technology Press
- Phong 1975 Phong, B.T. 1975. “Illumination for Computer-Generated Images.” *Communications of the ACM*, Vol 18 (6) 449— 455.
- PICHERT, J.W., ANDERSON, R.C. 1966. Taking a different perspectives on

References – Bibliography

- Pichert and Anderson 1966 a story. *Journal of Educational Psychology* 69, 309- 315. SCHANK, R.C. 1999. *Dynamic memory revisited*. Cambridge, UK. Cambridge University Press.
- Rajaram 1996 Rajaram, S. (1996). Perceptual Effects on Remebering: Recollective Processes in Picture Recognition Memory. *Journal of Experimental Psychology*, 22(2), 365-377.
- Riesberg 1997 Riesberg, D. (1997). *Cognition: Exploring the Science of the Mind*. W.W Norton and Company, London, UK.
- Rojahn and Pettigrew 1992 ROJAHN, K. PETTIGREW, T. 1992. Memory for schema relevant information: a meta-analytic resolution. *British Journal of Social Psychology* 31, 81-109.
- Ross et al. 2006 Ross, S., Skelton, R., Mueller, S. (2006). Gender differences in spatial navigation in virtual space: implications when using VEs in instruction and assessment. *Virtual Reality* 10:175–184
- Segal et al. 1970 Segal, S.J. & Fusella, V. (1970). Influence of Imaged Pictures and Sounds on Detection of Visual and Auditory Signals. *Journal of Experimental Psychology*, 83, 458-464.
- Shepard and Metzler 1971 Shepard, R.N. & Metzler, J. (1971). Mental Rotation of Three-Dimensional Objects. *Science*, 171, 701-703.
- Siegel and Howell 1992 Siegel R., and Howell J.R. 1992. *Thermal Radiation Heat Transfer*, 3rd Edition. Hemisphere Publishing Corporation, New York, NY.
- Smith 1992 Smith, M.E. (1992). Neurophysiological Manifestations of Recollective Memory Experience during Recognition Memory Judgements. *Journal of Cognitive Neuroscience*, 5, 1-13.
- Staffan Björk and Jussi Holopainen 2004 Björk, Staffan; Jussi Holopainen (2004). *Patterns In Game Design*. Charles River Media. p. 206. ISBN 1-58450-354-8.
- Steuer 1992 Steuer, J. (1992), Defining Virtual Reality: Dimensions Determining Telepresence, *Journal of Communication* 42, 73–93.
- Sutherland 1965 Sutherland, I. (1965), The Ultimate Display, in *Proceedings of the IFIP Congress*, pp.506–508.
- Tulving 1985 Tulving, E. (1985). Memory and Concioussness. *Canadian Psychologist*, 26, 1-12.
- Tulving 1992 Tulving, E. 1992. *Elements of Episodic Memory*, Oxford: Oxford Science Publications.
- Tulving 1993 Tulving, E. (1993). Varieties of Concioussness and Levels of Awareness in Memory. In A.D. Baddeley and L. Weiskrantz (Eds.), *Attention: Selection, Awareness and Control. A tribute to Donald Broadbent*, 283-299. London: Oxford University Press.
- Zotos et al. 2009 ZOTOS, A., MANIA, K., MOURKOSSIS, N. 2009. A Schema-based Selective Rendering Framework. *ACM Siggraph Symposium on Applied Perception in Graphics and Visualization*, 85-92, Chania, Crete, Greece.
- Zotos et al. 2010 Zotos, A., Mania, K., Mourkoussis, N. (2010). A Selective Rendering Algorithm based on Memory Schemas. Poster, *ACM Siggraph 2010*, New Orleans, USA.
- 3D Model Repositories <http://archive3d.net/>
<http://www.3dmodelfree.com/>
<http://artist-3d.com/>
<http://www.mr-cad.com/>
<http://www.3dvalley.com>
<http://www.archibase.net/>
- Epic Games <http://www.epicgames.com/>

References – Bibliography

Model Textures	http://cgtextures.com/ http://archivetextures.net/ http://www.crazy3dsmax.com
Ogre 3D	http://www.ogre3d.org/
Unity 3D	http://unity3d.com/
Unreal Development Kit	http://udk.com
Arrington Research Viewpoint User Manual	http://sbiwiki.cns.ki.se/mediawiki/images/Viewpoint_software_user_guide.pdf
Type of maps	http://www.reallusion.com/iclone/Help/3DXchange4/STD/Types_of_maps.htm
Texture Mapping	http://en.wikipedia.org/wiki/Texture_mapping
Immersion (virtual reality)	http://en.wikipedia.org/wiki/Immersion_(virtual_reality)
Eye tracking	http://en.wikipedia.org/wiki/Eye_tracking

APPENDIX A

First Pilot Study

Section A: Consent form

Project Title: *Gender differences in spatial navigation and spatial knowledge through a complex virtual environment*

Please read the following:

The research and experimental stage of this undergraduate work performed at the Technical University of Crete, Department of Electronic and Computer Engineering. You will be asked to fill out a questionnaire.

Responsible of this experiment is the undergraduate student Chris Paraskevas, doctoral student Giorgos Koulieris and the Assistant Professor Katerina Mania. Expected to keep you busy up to 20 minutes. We will use your data anonymously with other participants.

Remember that your participation is **voluntary**. You can choose not to participate in either or any of the individual stages of the questionnaire.

Please circle your answers to the following questions:

You understand the consent form?	YES	NO
Give your data for further analysis in the present work?	YES	NO
Confirmation that you are at least 18 years old?	YES	NO
Confirm that never before had signs or symptoms related to the disease of epilepsy?	YES	NO

Sign _____

Date _____

Name _____

APPENDIX A

Section B: Personal information

Please fill the following:

Name:

Age group: 18-22 23-27 28-32 33-37 38-42 above 43

Gender: Male /Female

Date and time: _____

(If you currently
studying choose

your educational rank): Undergraduate Postgraduate Doctoral

Current state: Student Researcher Academic

Other (report) _____

APPENDIX B

Experiment's scenario

Thank you for participating in the process of our experiments. The experiment consists **of 3 stages**. Below are details for each stage separately. Having carefully read the details of each stage, you are ready to start.

Briefly, the first step is to familiarize you with the equipment and with the three-dimensional graphics. The second stage is the main experiment. The last stage consists of a review section.

Step 1. Familiarity with the equipment and three-dimensional graphics

Are you ready to participate in the first stage. The aim is to familiarize yourself with the equipment (Head Mounted Display or HMD) and three-dimensional graphics. As long as you have to navigate in an open space. Use the equipment to turn your chair 360 degrees and look in all angles. Ask for free any questions or stop the process if you feel unwell. Once you feel comfortable with the equipment and navigate in three-dimensional space, ask the person responsible for the experiment to "load" the main stage of the experiment.

Step 2. The main experiment. Browse a three-dimensional scene.

The new scene describes a room with objects and dining area, kitchen, office and living room and are required to observe all places and objects of these sites. The time it will spend in each area of the home and what you look is yours concern. The aim of the experiment is to test the new technology laboratory equipment and to identify potential problems.

Step 3. Review

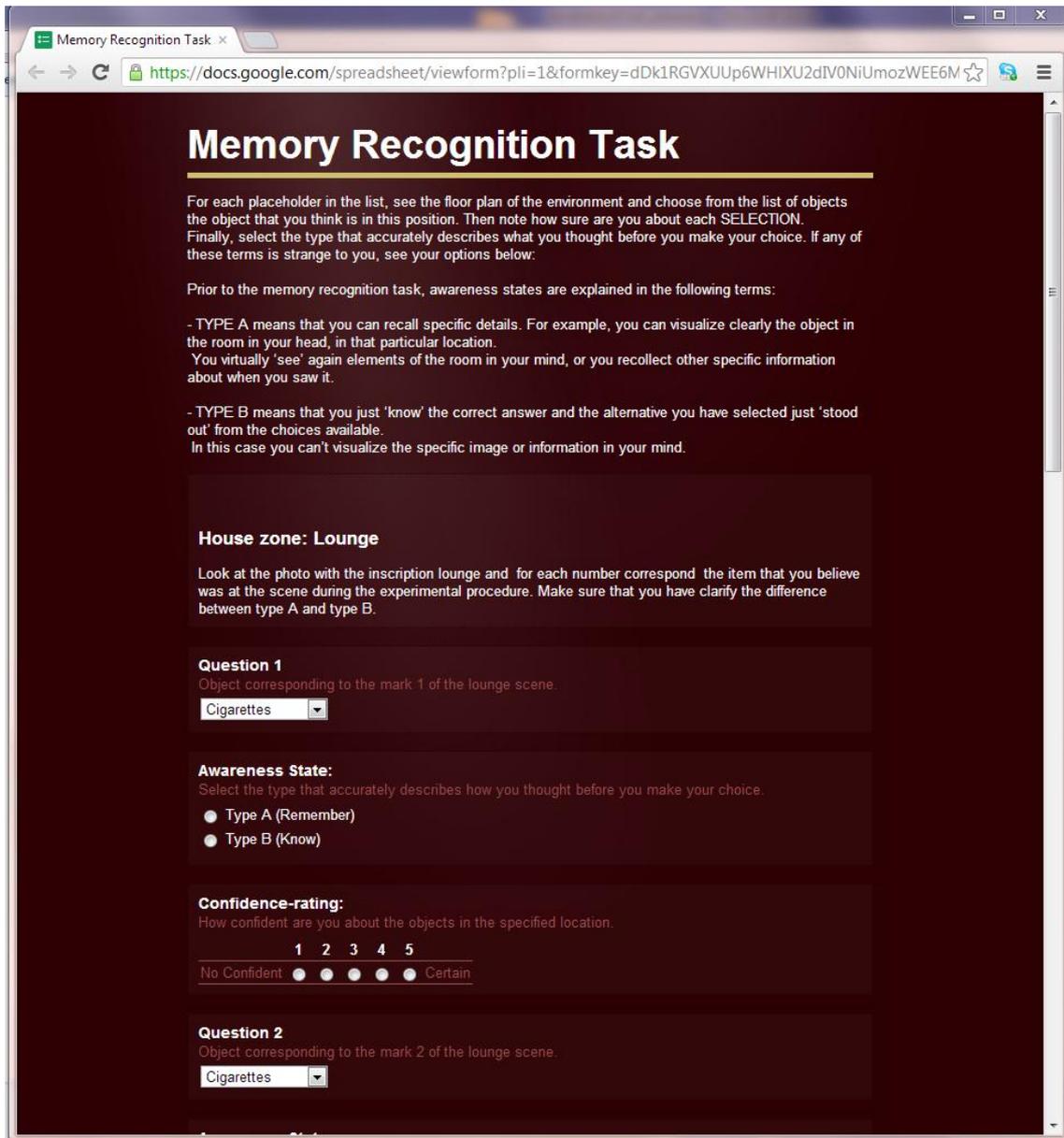
According to the experience gained from the last scene you saw (stage 2), answer in the review form below. You can have as much time you want to read the instructions and answer all the questions.

Please do not talk with other participants about your impressions of the navigation or stage of review. However, you can ask those in charge of the experiment for any questions you have.

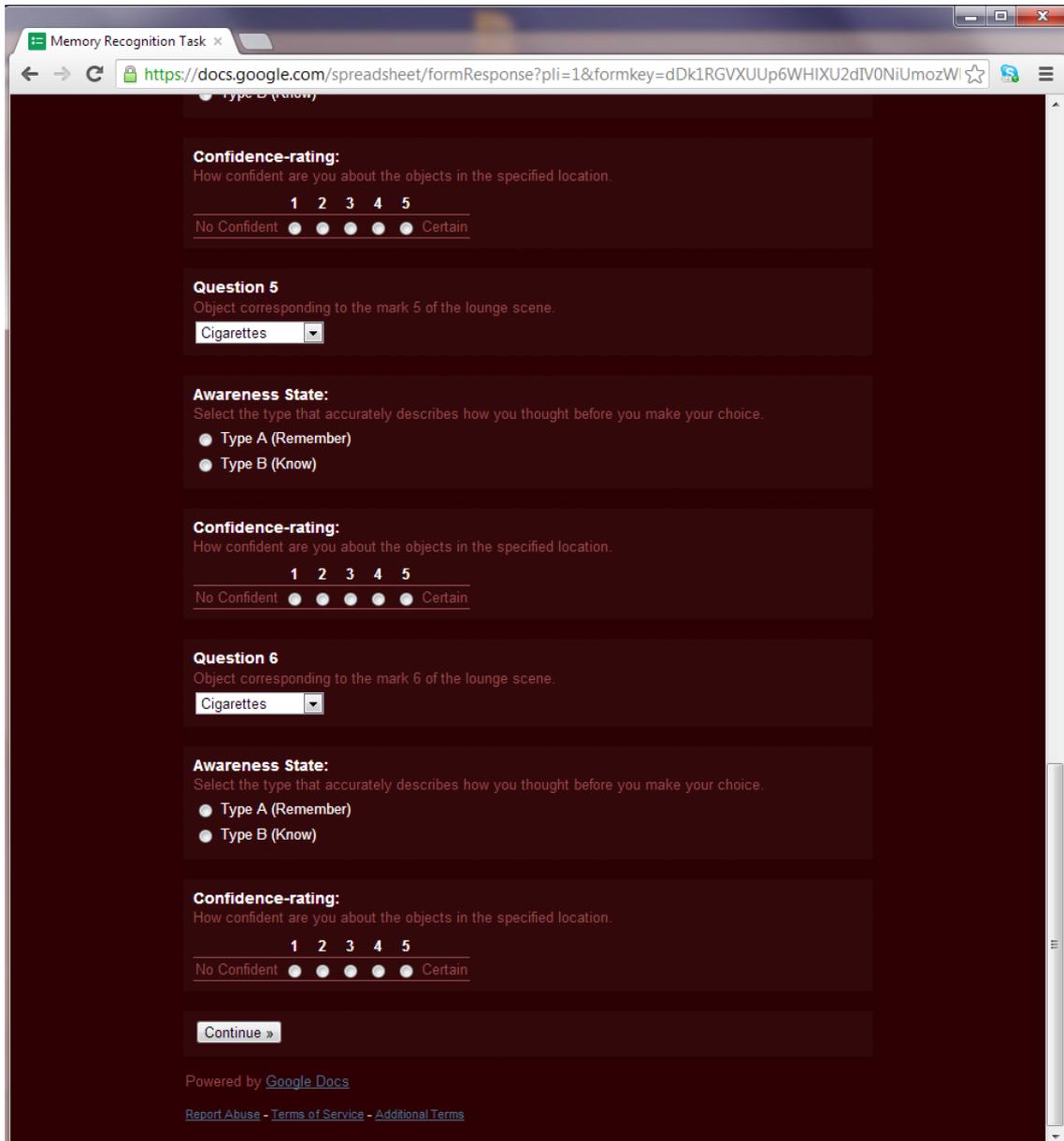
This is the final stage. Thanks for your participation and we hope to enjoy the ride in the Virtual Reality!!

APPENDIX C

Online Questionnaires: Screenshots



Online Questionnaires: Screenshots



Online Questionnaires: Screenshots

