NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF INFORMATION TRANSMISSION SYSTEMS AND MATERIAL TECHNOLOGY

Navigating a Virtual Environment using Eye Movements

DIPLOMA THESIS

Stefanos Apostolopoulos

Supervisors : Nikolaos Ouzounoglou, Professor, NTUA
Angelos Amditis, Researcher A, ICCS

Athens, October 2012

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF INFORMATION TRANSMISSION SYSTEMS AND MATERIAL TECHNOLOGY

Navigating a Virtual Environment using Eye Movements

DIPLOMA THESIS

Stefanos Apostolopoulos

Supervisors : Nikolaos Ouzounoglou, Professor NTUA

Angelos Amditis, Researcher A, ICCS

Approved by the three-member examination committee:

....	....	....
N. Ouzounoglou Professor, NTUA	A. Amditis Research A, ICCS	G. Matsopoulos Assist. Professor, NTUA

Athens, October 2012

....

Stefanos Apostolopoulos

Diploma in Electrical and Computer Engineer NTUA

Acknowledgements

I would like to express my gratitude to my supervising professor N. Ouzounoglou, for giving me the opportunity to work on a subject that is both interesting and useful and Dr. A. Amditis, Researcher A of ICCS, for his trust and advice.

I would like to give my warm regards to Dr. A. Tzelepi for her substantial and proactive help, guidance and advice along all parts of this journey.

I would also like to thank my family and my friends for being there when I needed them and Athina for her insight, patience and support. Without you none of this would have been possible!

Abstract

Virtual Environments have been used over the years in entertainment, educational and rehabilitation contexts. User immersion within such environments can be improved not only by optimizing the graphics and design of the environment per se, but also by allowing users to navigate them in a natural fashion. Hence, the purpose of this thesis is twofold: first, to develop a virtual environment that can explored via multiple Human-Computer Interaction methods and, second, to implement a navigation method using eye movements.

To that end, a realistic 3d labyrinth was constructed using state of the art graphics algorithms. The visual motif was inspired by Greek mythology, while the geometry of the environment was based on a pseudo-random labyrinth generator, providing a unique, but repeatable, experience to users. Simulation of light and shadows, water movement, light bloom, refraction and reflection were implemented to increase user immersion. All effects were rendered in real time and presented using stereographic projection on the Immersive Power Wall (CAVE) of the Virtual Reality Laboratory in the Institute of Communication and Computer System.

User navigation within the environment was achieved through a network communication protocol that can transfer input signals from diverse input sources. In the context of this thesis, the environment was tested with a computer keyboard, mouse and electrooculography (EOG). Ultimately, users were able to navigate the labyrinth solely through eye movements. This was achieved by recording and decoding EOG signals in real time using signal processing techniques.

Overall, this project represents a generic platform that is suitable for developing, testing and comparing different ways of navigation.

Keywords: virtual reality, human-computer interaction, eye movements, electrooculography

Περίληψη

Περιβάλλοντα εικονικής πραγματικότητας έχουν χρησιμοποιηθεί τα τελευταία χρόνια για σκοπούς ψυχαγωγικούς, εκπαιδευτικούς και αποκατάστασης. Η περιήγηση του χρήστη σε τέτοια περιβάλλοντα μπορεί να βελτιωθεί όχι μόνο με τελειοποιώντας τα γραφικά και το γενικότερο σχεδιασμό του περιβάλλοντος, αλλά επίσης επιτρέποντας στους χρήστες να πλοηγούνται με φυσικό τρόπο. Ως εκ τούτου, η παρούσα διπλωματική εργασία έχει δύο κυρίως στόχους: πρώτον, την ανάπτυξη ενός εικονικού περιβάλλοντος που μπορεί να εξερευνηθεί μέσω πολλαπλών μεθόδων αλληλεπίδρασης ανθρώπου-υπολογιστή και, δεύτερον, την υλοποίηση μιας συγκεκριμένης μεθόδου πλοήγησης βασισμένη σε κινήσεις των ματιών.

Για το σκοπό αυτό κατασκευάστηκε ένας ρεαλιστικός τρισδιάστατος λαβύρινθος, χρησιμοποιώντας αλγορίθμους γραφικών αιχμής. Το οπτικό μοτίβο του λαβυρίνθου είναι εμπνευσμένο από την ελληνική μυθολογία, ενώ η γεωμετρία του περιβάλλοντος βασίστηκε σε μια ψευδο-τυχαία γεννήτρια λαβύρινθων, προσφέροντας μια μοναδική, αλλά ελεγχόμενη, εμπειρία σε κάθε χρήστη. Προκειμένου να αυξηθεί η αίσθηση εμβύθυσης των χρηστών στο περιβάλλον αυτό, υλοποιήθηκαν αλγόριθμοι προσομοίωσης φωτός και σκιών, κίνησης νερού, αλλά και διάθλασης και περίθλασης του φωτός. Το περιβάλλον σχεδιάζεται σε πραγματικό χρόνο και παρουσιάζεται στους χρήστες σε στερεογραφική προβολή, στο Immersive Power Wall (CAVE) του Εργαστηρίου Εικονικής Πραγματικότητας, στο Ερευντηικό Πανεπιστημιακό Ινστιτούτο Συστημάτων Επικοινωνιών & Υπολογιστών.

Η πλοήγηση του χρήστη στο περιβάλλον επιτεύχθηκε μέσα από ένα πρωτόκολλο επικοινωνίας δικτύου που μπορεί να μεταφέρει σήματα εισόδου από διαφορετικές πηγές. Στο πλαίσιο της παρούσας εργασίας, το περιβάλλον ελέγχθηκε με πληκτρολόγιο υπολογιστή, ποντίκι και ηλεκτρο-οφθαλμογράφημα (ΗΟΓ). Οι χρήστες είχαν τελικά τη δυνατότητα να περιηγηθούν στο λαβύρινθο μέσω κινήσεων των ματιών τους. Αυτό επιτεύχθηκε με την καταγραφή και αποκωδικοποίηση σημάτων ΗΟΓ σε πραγματικό χρόνο, χρησιμοποιώντας τεχνικές επεξεργασίας σήματος.

Συνοπτικά, η παρούσα διπλωματική αποτελεί μια γενική πλατφόρμα που μπορεί να χρησιμοποιηθεί για ανάπτυξη, δοκιμή και σύγκριση διαφορετικών τρόποων πλοήγησης.

Λέξεις-κλειδιά: εικονική πραγματικότητα, αλληλεπίδραση ανθρώπου-υπολογιστή, κινήσεις ματιών, ηλεοκτρο-οφθαλμογράφημα.

Table of contents

Part A: Theoretical Part

Chapter 1: Introduction

Virtual Reality (VR) is the attempt to simulate physical presence in a computer-generated environment. These environments, commonly referred to as “Virtual Environments” (VEs), are typically composed of visual, aural and haptic elements that represent existing (e.g. the cockpit of an airplane) or novel settings (e.g. the view of a mars rover).

1.1 History

Even though VR is nowadays tied to computer simulations, its roots predate computers by far. The first appearances can be traced back to the 15th century, with the arrival of 360-degree panoramic art. On notable such example is the “Sala Delle Prospettive” by Baldassere Peruzzi, an Italian architect born in 1481 (figure 1.1).

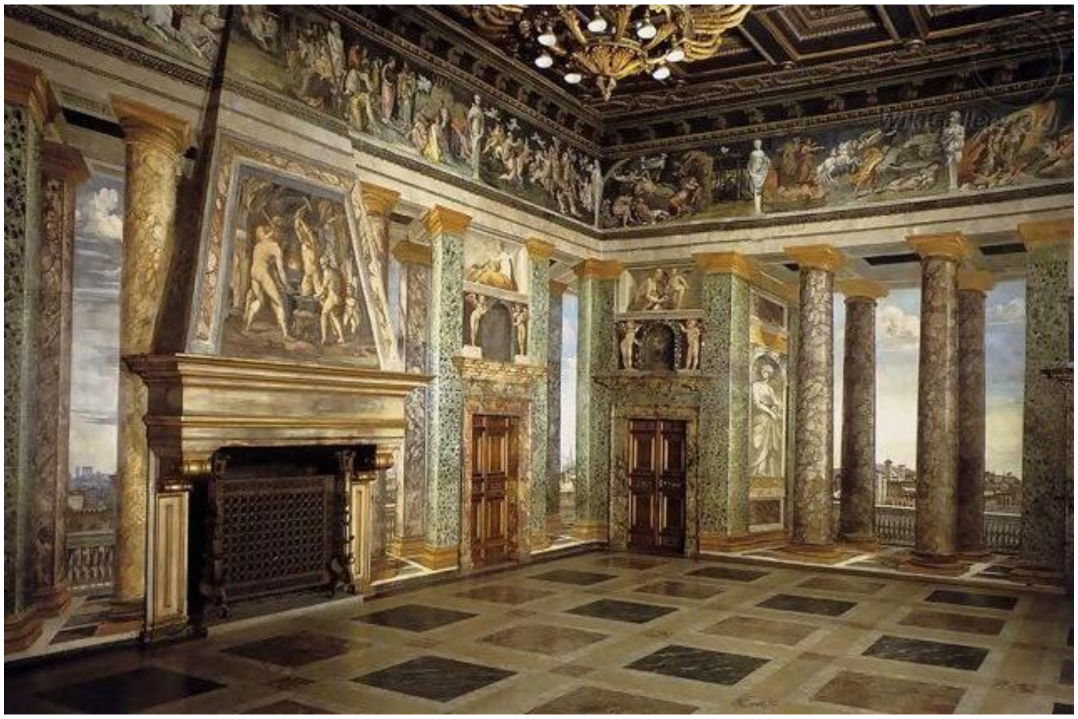


Figure 1.1 - Sala Delle Prospettive, Baldassere Peruzzi, 1518-1519

The walls of this room are painted using mathematically-accurate perspective to create the illusion of depth. Standing on the left side of the room, the projection lines of the painted walls align with the floor tiles to give the illusion of an open-air terrace. The real elements of the room, i.e. the doors and fireplace, blend in with the painted walls to enhance the illusion.

As time passed, novel uses appeared for VR. In 1920, the first vehicle simulators were invented. In 1962, Morton Heilig built Sensorama, a mechanical device that could display five short films, engaging multiple senses (sight, sound, smell, and touch). In 1968, Ivan Sutherland and Bob Sproull constructed the computer-based VR system, known as the "Sword of Damocles", which consisted of a stereoscopic head-mounted display (HMD) capable of tracking the head motions of its user. A more compact HMD device was created by Thomas Furness in 1982 for the U.S. Air Force, which augmented the pilots' view with flight path and targeting information.

The first commercially available VR devices, "DataGlove" and "EyePhone HMD" was developed by VPL Research Inc. in 1985 and 1989, respectively. The next important milestone would be achieved three years later, in 1992, when the University of Illinois introduced the first system to use stereoscopic video projection and shutter glasses, rather than static or head-mounted displays. This approach freed the user from wearing bulky equipment and allowed a wider range of movements, while maintaining the illusion of immersion in the virtual world. In fact, modern versions of this system, called CAVE (CAVE Automatic Virtual Environment, figure 1.2), are still used today in scientific and commercial visualizations.

Advancements in computing and display technology gradually helped reduce the cost and size of VR systems and created new markets for the technology. In 1995, Nintendo released the Virtual Boy, a portable VR system designed solely for video games. Despite its aspirations, the system was a commercial and critical failure. Criticism focused on its peculiar design, resembling a head-mounted display that had to be set on a desk, its uninspiring red-and-black monochrome LCD screens, and the eye and neck strain it caused after short periods of use.

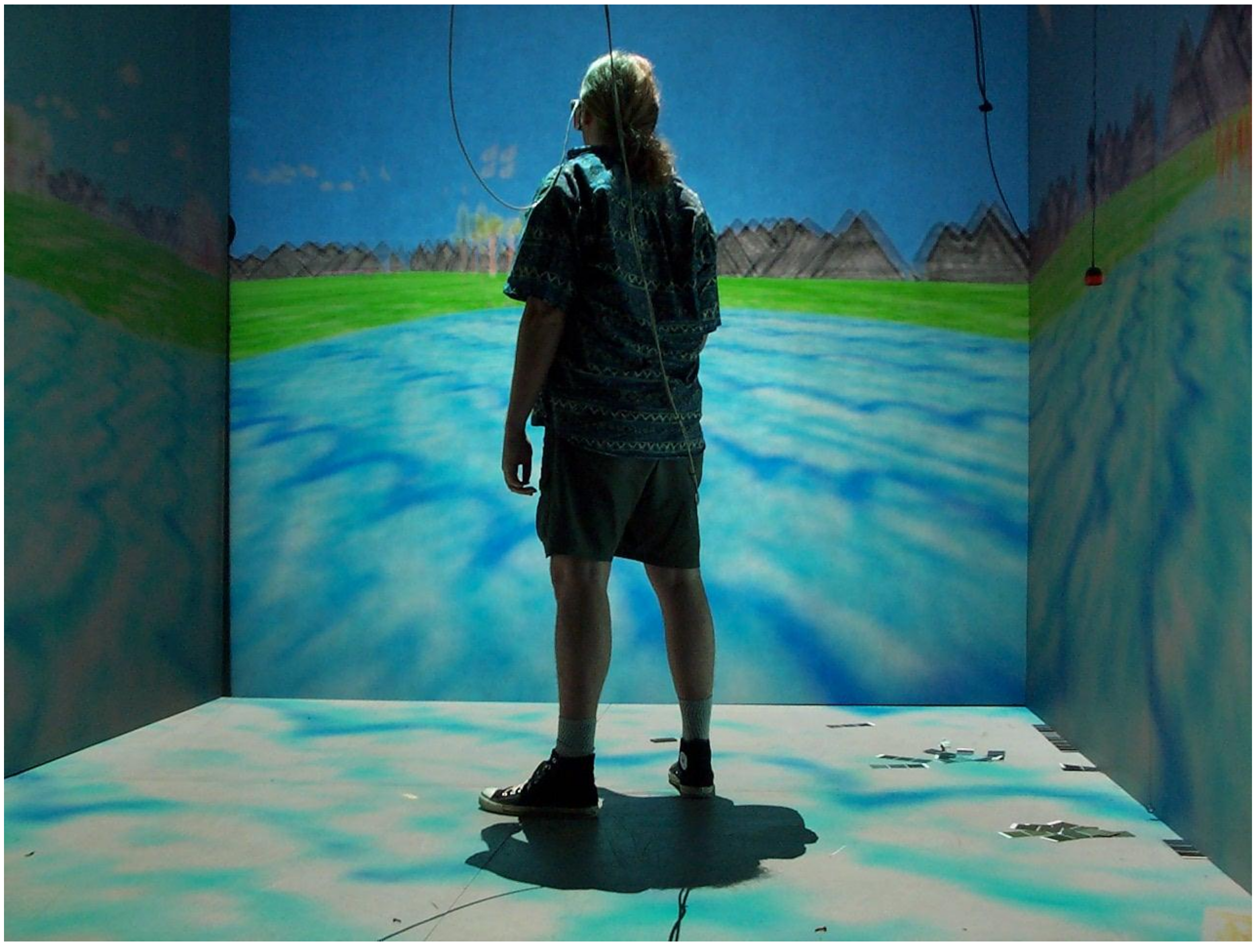


Figure 1.2 - Cave Automatic Virtual Environment at EVL, University of Illinois, 2001

While the first attempt to bring VR technology into the mainstream failed, a different approach was met with more success: the first online *3d virtual world*, Meridian 59, was released in 1996. Unlike previous 2d virtual worlds, Meridian 59 allowed its players to explore its environments through the eyes of their *Avatars*, i.e. their virtual representations inside the game world. Following this breakthrough, the number of free and commercial 3d virtual worlds exploded. Notable examples, such as Second Life, World of Warcraft and RuneScape, average 80K - 140K concurrent users and 2M - 10M active players (data as of August 2012).

The success of these virtual worlds partly stems from their low barrier to entry for casual users, (i.e. users outside the research or professional communities). Unlike previous VR systems, virtual worlds such as Second Life can be accessed through a typical personal computer (PC) either for free or for a small monthly fee. No specialized equipment is required to enter and interact with the virtual environment - a user can explore it from the comfort of her home.

However, this very fact that drove the popularity of online virtual worlds also placed limits on the immersion of the user inside the virtual environment. The lack of specialized input devices confines human-computer interactions to indirect control interfaces based on PC mice and keyboards, where the mouse rotates the user avatar inside the world and the keyboard controls his movement. The lack of specialized output devices both limits the view of the world to a flat monoscopic projection with low field of vision and reduces the ability to provide haptic feedback to user actions (more details on chapters 2 and 3).

In response to the proliferation of virtual worlds, hardware manufacturers have developed new technologies that bridge the gap between high-cost, industrial / research VR systems, such as CAVE, and low-cost, commoditized computer hardware. Multi-display systems, stereoscopic monitors that rely on shutter glasses and, more recently, autostereoscopic displays dramatically improve user immersion; infrared and optical motion-tracking systems allow direct control of the avatar inside the environment; finally, haptic devices with motion sensing and force-feedback technology provide instant feedback to user actions (e.g. by simulating the effects of acceleration on a flight stick or car wheel).

As of 2012, new technologies and decreasing costs have led to a VR renaissance, with VR systems appearing in entertainment, training, research, industrial design and medical applications.

1.2 Applications of VR technology

With the evolution and on-going commoditization of VR technology, new applications continuously appear. Even so, several major segments can be identified historically:

1.2.1 Entertainment

The film industry and the computer game industry have enjoyed significant success in their adoption and use of VR technology.

Films employ VR in two respects: (a) an ever-increasing number of films is being recorded and released in stereoscopic 3d format, which can increase viewer immersion; (b) several major films have employed virtual actors and virtual environments either in part (to replicate stunts and sets that would be expensive or even impossible to create using physical methods) or in whole (as a stylistic choice), to various degrees of success. As of 2012, the most expensive and highest-grossing film of all time, Avatar, utilizes both techniques in its entirety (the actors and sets are completely computer-generated - actors were only used for motion capture and voice acting).

Computer games, and especially so-called AAA computer games, use techniques directly inspired by the film industry, in an effort to increase realism and player immersion. Stereoscopic rendering, motion tracking and haptic feedback are employed by an increasing amount of games, while efforts to achieve visual photorealism play a significant role in the advancement of computer hardware and software used in VR systems.

1.2.2 Training

Vehicle and flight simulation has historically been one of the most active research topics in the field of VR technology. The goal of vehicle simulation is to provide pilots and drivers with hands-on training on vehicle instruments and behavior, under both normal and abnormal conditions that could otherwise be expensive or even dangerous to recreate (electrical and hydraulic system failures, engine damage or extreme weather conditions). Modern flight training schools employ generic “Flight and Navigation Procedures Trainers” (FNPT) for initial training and more advanced “Full Flight Simulators” (FFS) for aircraft-specific training, to complement in-aircraft flight exercises.

More recently, VR has been used to enhance of industrial safety procedures. VR technologies have been used: (a) to create realistic training conditions and help trainees develop skills that can be transferred from the virtual world to the real world; (b) to assist in risk assessment through the identification of risk sources and improve the familiarity of the personnel with the layout of an industrial plant and its equipment; and (c) to allow the evaluation of “what-if” scenarios and the performance of emergency procedures under different conditions. Commercial VR products for industrial safety are now available (Loupos et al., 2008).

1.2.3 Research and medicine

VR systems have been developed to simulate and research human behavior under different conditions. One notable example of such a system is the virtual agent “Jack” that was developed at the Center for Computer Graphics Research at the University of Pennsylvania. Jack uses advanced kinematics and behavioral models to react to his environment, and can be used to evaluate the effect of human factors on potential building designs.

Other interesting applications of VR technology in research include Computational Neuroscience and Molecular Chemistry, for the visualization of neural network and molecular simulations, respectively. Significant efforts are also being made to introduce VR systems in the field of medicine, and particularly in surgeries, ultrasound examinations and patient rehabilitation.

1.2.4 Industrial design

The architectural design of buildings and vehicles are areas with significant investment in VR technologies. Notable examples include Vicker’s Shipbuilding and Engineering Ltd., a U.S. company that has designed a VR system that simulates the behavior of underwater vehicles; BMW, which uses VR to investigate the results of vehicle collision testing; Fraunhofer IGD and Wismut GmbH, which have co-developed a VR system to visualize the results of a mining operation.

1.3 Categories of VR systems

A complete categorization of VR systems is difficult, due to the sheer number of different implementations available. However, modern VR configurations can be ranked according to the degree of immersion the can provide. *Immersion* or *presence* is the ability of a VR system to provide a user with a feeling of being “inside” the virtual environment or, in other words, the ability to suspend disbelief on the part of the user.

Three distinct categories of VR systems can be identified:

1.3.1 Non-immersive systems

Non-immersive or desktop systems offer the least amount of immersion. In such systems, the virtual environment is presented to the user via a standard computer monitor, offering a so-called “Window on World”. Users usually interact with the environment through standard input devices, such as keyboards and mice, sometimes enhanced with low-cost head- or motion-tracking equipment.

Such systems use commoditized computer hardware, available in typical computer systems. The lack of specialized hardware lowers the cost of implementation, but also limits user immersion to the minimum amount that can be achieved through standard 2d monoscopic monitors.

1.3.2 Semi-immersive systems

Semi-immersive systems are the intermediate step between non-immersive and fully immersive VR systems. They use technology that was initially developed for flight simulators, and usually comprise high-performance graphic cards and multiple monitors or projection systems, often with stereoscopic capabilities.

The higher field-of-view increases user immersion and improves the sense of scale compared to non-immersive systems. The cost is higher than desktop systems but still considerably lower than specialized fully-immersive technology, covered below.

1.3.3 Fully-immersive systems

Fully-immersive systems provide the highest-quality experience and sense of immersion. They use specialized equipment for both interaction and visual / aural representation, thus increasing the total cost of implementation.

Typical technologies include head-mounted displays (figure 1.3) or 6-wall CAVEs with stereoscopic capabilities; surround sound systems with head-related transfer functions to improve sound localization; motion tracking sensors and force feedback to achieve tactile realism. Depending on the application, such systems may be combined with physical equipment, such as airplane cockpits, to provide a highly realistic experience.

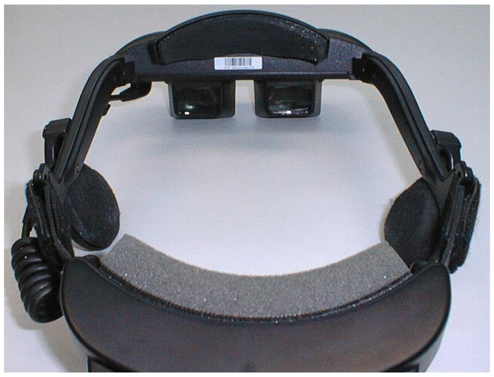


Figure 1.3 - Head-mounted display with stereoscopic capabilties
(Image courtesy of VISERG, Loughborough University)

Chapter 2: Computer Graphics

Computer graphics, in their most abstract sense, refer to images created, or *rendered*, using computers. *Interactive computer graphics* refer to computer graphics that are react to user input in real-time, typically in 15 milliseconds or less. This is in contrast to non-interactive or *offline* computer graphics that can often take hours to render (Akenine-Moller et al., 2002).

The history of computer graphics is intertwined with the history and evolution of computing technology itself. The first known example of computer-generated graphics is the vector-scope display of the Whirlwind computer in 1951. As computer hardware and software evolved, more complex algorithms could be developed to improve the simulation of reality and, in fact, such algorithms have often driven the development of more powerful hardware. In that sense, the maximum quality or realism that can be achieved at any point time is limited by two main factors: first, the computing power available at that time; and, second, by the sophistication of the graphics algorithms that will take advantage of this computing power (Kent and Smith, 1980).

In the following pages, I will describe the main computer graphics concepts first from a software, then from a hardware perspective. Finally, I will present the main components of a modern graphics pipeline.

2.1 Software

The rendering of computer graphics is an area of active scientific research. New algorithms are continuously being invented, each with different performance characteristics and quality tradeoffs. Most graphics algorithms can be categorized in one of three distinct categories: ray casting, ray tracing and rasterization. Hybrid algorithms that combine the advantages of more than one categories also exist.

The main categories and their key characteristics are presented below:

2.1.1 Ray casting

Ray casting is an image order algorithm that can be used to render three dimensional scenes on computer monitors. It works by tracing one ray of light per screen pixel, from the eye of the viewer towards a light source, until it intersects with a solid surface. The main assumption behind this algorithm is that if a surface faces a light, the light will reach that surface and not be blocked or in shadow.

The advantage of this method is that it can be implemented using simple calculations and has therefore been used in early real-time 3d computer graphics. However, the simplicity of this algorithm precludes the accurate rendering of reflections, refractions and shadows. These visual effects must be faked using texture maps or other methods.

The first ray casting algorithm used for rendering was presented by Arthur Appel in 1968 (Macey, University of Bournemouth lecture notes). Ray casting for producing computer graphics was first used by scientists at Mathematical Applications Group, Inc., (MAGI) of Elmsford, New York (Goldstein and Nagel, 1971).

2.1.2 Ray tracing

Ray tracing is an evolved version of ray casting. Whereas ray casting traces light rays from the eye into the scene until they hit an object, ray tracing continues this process by generating additional rays after they hit an object. In the initial incarnation of this algorithm, invented by Turner Whitted in 1979, three types of secondary rays were defined: reflection, refraction and shadow (Whitted, 1979; Nikodym, 2010). More recent implementations incorporate additional types to account for scattering and dispersion phenomena, such as chromatic aberration.

In ray tracing, primary rays are traced either from the eye towards each light or from each light towards the eye. In either case, a reflection ray appears when a primary ray hits a mirror-like object and is traced in the direction of the mirror-reflection. If an object is translucent, a refraction ray is also spawned to travel through its mass, in a direction defined by its refraction index using Snell's law (see also figure 2.1). In order to reduce the computational load, shadow rays are used to test whether an object is visible to a light source, and avoid tracing additional rays if it is not.

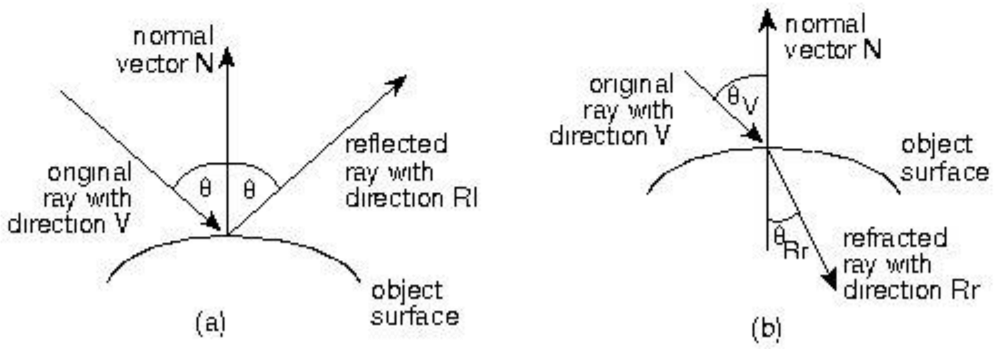


Figure 2.1 - (a) Reflected ray. (b) Refracted ray

Ray tracing offers the most realistic simulation of lighting, compared to ray casting and rasterization (see figure 2.2). Effects such as reflections and shadows are a natural result of the ray tracing algorithm. Even so, ray tracing is surprisingly simple to implement and is very suitable to parallel computations, due to the independence of the light rays (Chalmers et al., 2002).

Despite the simplicity of its implementation, ray tracing has a high computational cost and is best suited for non-real-time applications. Where other algorithms use data coherence to share computations between pixels, ray traces restarts the tracing process for every single light ray. Specialized data structures, such as k-trees, have been developed to accelerate the ray tracing process, even though they can only be applied to the initial rays and for static scenes. For these reasons, ray tracing is mainly in the film industry, where high quality is essential and rendering time is not a key consideration.

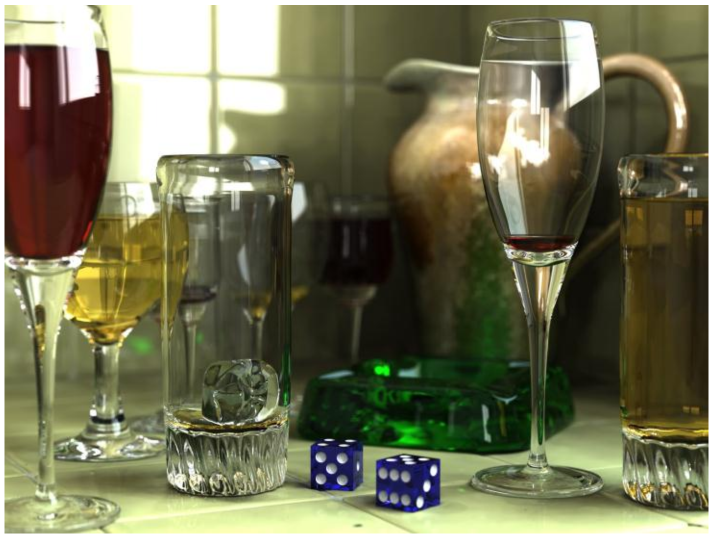


Figure 2.2 - Photorealistic scene traced with POV-Ray. (Image courtesy of Gilles Tran)

2.1.3 Rasterization

Rasterization is the most popular rendering algorithm for real-time 3d applications. On an abstract level, it represents the process of computing the projection of 3d geometry to a 2d plane for display. The 3d geometry consists of polygons, typically triangles, which are represented by their vertices. The rasterization algorithm receives a stream of vertices and projects them onto a 2d plane, usually a computer monitor, for display.

In that sense, rasterization merely defines the mapping of 3d geometry to a 2d plane; the color of the final display must be specified by additional algorithms, such as texture and shadow mapping, physical or non-physical light transport functions (e.g. gouraud or phong shading) and more. These algorithms are usually picked depending on artistic intent and available computing power.

Rasterization offers several advantages over other rendering techniques. Not only is it computationally efficient, but it is also amenable to implementation in dedicated computing hardware. Modern GPUs are highly tuned to the requirements of rasterization and can process massive amounts of data, in the order of hundreds of *gigaflops* (i.e. billion floating-point operations per second, see figure 2.3).

However, effects that are produced naturally in ray tracing, such as shadows and reflections, are harder to implement in a rasterization pipeline. Several algorithms have been developed to implement these effects, with different tradeoffs between quality and performance. Recently, hybrid algorithms have appeared to combine the performance of rasterization with the superior quality of ray tracing (Cabeleira J).

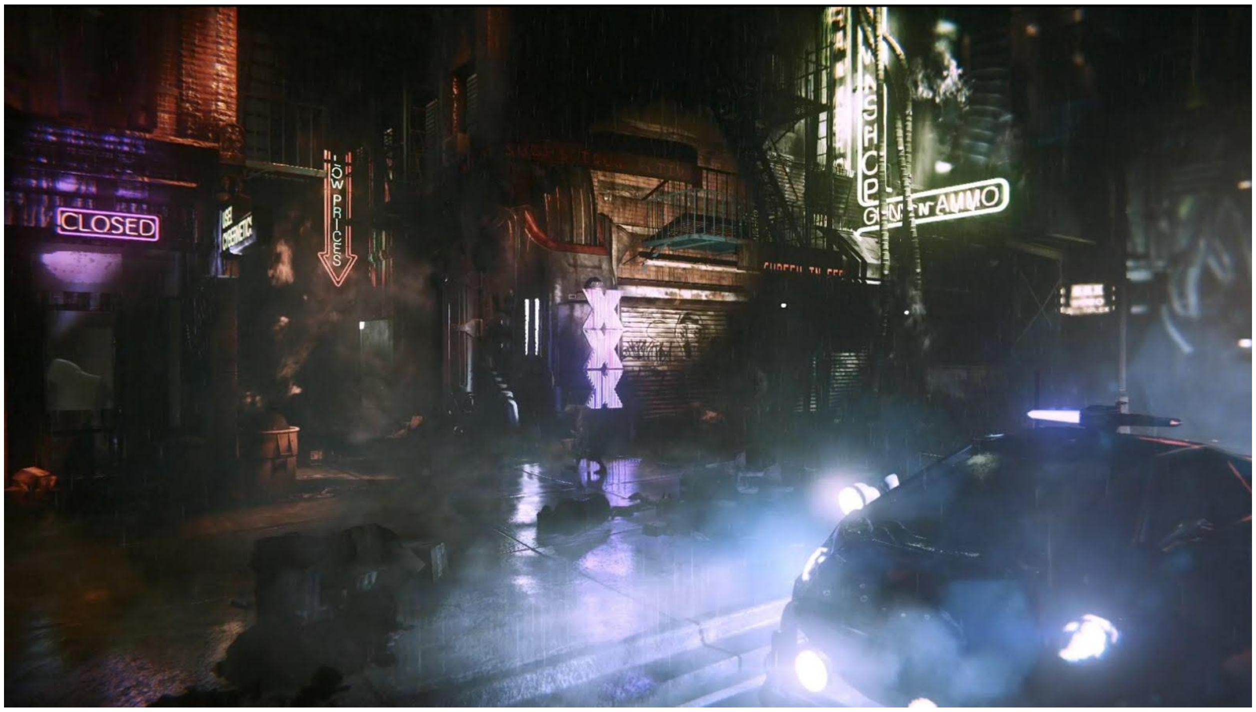


Figure 2.3 - Samaritan demo showing next-generation rasterization effects.
(Image courtesy of Epic Games.)

2.2 Hardware

As graphics algorithms became more sophisticated, their computational requirements soon surpassed the capabilities of early general-purpose *Central Processing Units* (CPUs). Specialized processors, called *Graphics Processing Units* (GPUs), were designed to fill this gap. GPUs work by implementing specific graphics algorithms in hardware, thus relieving the CPU from the relevant computations. This process is called *acceleration*, due to the significant performance gain that it brings.

The first GPU, iSBX 275, was released by Intel in 1983 and contained hardware that could accelerate the drawing of 2d lines, rectangles, arcs and text. Two years later, the Commodore Amiga GPU incorporated a *blitter*, a new piece of hardware that could accelerate the movement and manipulation of arbitrary memory bitmaps (Swaine, 1983). By 1995, every GPU could accelerate 2d drawing and the first 3d accelerators were starting to appear.

Like 2d GPUs, 3d GPUs contain dedicated hardware to accelerate parts of graphics computations. All commercial 3d accelerators rely on the rasterization technique[1], where the main graphic elements, known as *primitives*, are points, lines or triangles. Early 3d accelerators, such as Matrox Mystique, 3dfx Voodoo and Rendition Verite, could typically accelerate the projection of vertices from 3d space to a 2d plane and apply texture maps to polygons. As time passed, accelerators would gain additional capabilities for texture filtering, polygon transformation and lighting (these techniques will be analyzed in more detail in the next section).

The next evolutionary step came in 2000, when Nvidia introduced the Geforce 3 processor. Unlike previous processors, this GPU offered a limited form of programmability, i.e. applications could alter specific parts of the rasterization process. In 2002, the Radeon 9700 GPU by Ati followed this process to its logical conclusion, by replacing the core parts of the rasterization process parts by programmable *vertex* and *fragment* processors. This was a significant development, not only because it marked the birth of the modern *general-purpose GPU* (GPGPU), but also because it allowed programmers unprecedented amounts of flexibility in implementing new graphics algorithms (Rege, 2008).

As of 2012, modern GPUs have continued this trend of increased programmability. A modern GPU consists of hundreds of programmable *stream processors* that can execute either graphics programs, called *shaders*, or general-purpose computations. Tim Sweeney, founder of Epic Games, predicts that GPUs will gradually be integrated back into CPUs, as the programmability of the former and the performance of the latter increases. He also expects that “life-like” graphics would require a 2000x increase in performance over current hardware[2], and the invention of new physically-accurate graphics algorithms. His first prediction has for the most part been validated: both AMD and Intel now integrate fully-functional GPUs inside their CPUs. His second prediction remains to be seen (Arstechnica.com; Venturebeats.com).

[1] Prototypes for ray tracing accelerators exist, but are not available in the mass market. The most famous one is Intel’s “Larabee” project, which was ultimately cancelled after a long - and costly - period of development.

[2] This is roughly the increase in performance requirements between Unreal 1 (1998) and Unreal 4 (expected in 2013). Unreal 1 required around 1 gigaflop of GPU power, while the “Samaritan demo” of Unreal 4 requires around 2.5 teraflops.

2.3 Graphics pipeline

This section will present the key components of a modern *graphics pipeline*. A graphics pipeline defines the process by which a 3d scene is transformed into the 2d raster image that is displayed to the user.

The concepts below are not tied to a specific graphics programming interface (graphics API) or programming language. However, to avoid confusion, the terminology follows the conventions of the Open Graphics Library (OpenGL, which is covered in the next chapter).

2.3.1 The 3d scene

A 3d *scene*, also known as *world* or *virtual environment*, forms the input of the graphics pipeline. In the most abstract level, a scene is composed by its geometry, materials and light sources. (Additional information is commonly included for the audio pipeline and for the world simulation - this falls outside the scope of this discussion.)

The scene *geometry* is usually defined in terms of a graph, with graph vertices representing the vertices of the geometry and edges connecting these vertices to form primitives (points, lines or triangles). Aside from its position in *world coordinates*, each vertex may also contain additional *attributes*, such as a normal (defining the perpendicular direction of the graphics primitive), a set of texture coordinates (used to map materials onto complex surfaces) and more. Rarely, mathematical equations, voxels or distance fields are also used to define geometry, to take advantage of domain-specific advantages (e.g. a sphere can be represented in a very compact fashion by its center and its radius. A scene comprising thousands of spheres can be described very efficiently using this representation, compared to the millions of triangle vertices that would be required.) However, such representations are not directly usable by modern GPUs and have to be converted to triangle vertices before rasterization, either on the CPU or using a GPU shader.

Materials influence the appearance of the scene geometry. Imagine two identical spheres, one made of wood the other made of silver. The underlying geometry may be identical, but the appearance of the spheres is completely different, due to their materials. Materials in modern computer graphics are described using multiple parameters: a *texture map* that defines the color (texture) at each point of the geometry; a *translucency map*, often combined with the texture map, that defines the translucency of the material; a *specular map*, that defines its glossiness; a *normal map*, sometimes accompanied by a *height map*, that describes its bumpiness, in the case of uneven surfaces.

Light sources provide the illumination sources for the scene. Real-time computer graphics, as examined here, model lights using significant simplifications compared to reality. Unlike reality, lights in computer graphics have a *type*, that define their behavior. Common types include *directional* lights, which are thought of as being infinitely far away from the scene and thus their rays[3] are parallel and in a single direction (e.g. the sun or moon); point lights, which cast rays towards all directions equally; *spotlights*, which are similar to point lights, but cast rays in a coned direction; *area lights* that are similar to directional but originate from a cutoff plane inside the scene; *volumetric lights* that cast rays inwards their enclosed space, instead of outwards.

2.3.2 The graphics pipeline

The graphics pipeline defines *how* the 3d scene is presented to the user. It consists of a set of GPU programs, called shaders, which are executed one after the other, in order to produce a visible image from the abstract 3d scene representation. Each shader belongs to a specific shader type, or *stage*, and process a specific portion of the data that comprise the 3d scene. A complete execution of all shader stages is called a *pass*; modern scenes often require multiple passes, each using different shaders, to be fully processed. A complete pass consists of the following stages:

1. The first stage executes a *vertex shader* for each vertex in the scene. The vertex shader controls the transformation of the scene geometry from its *world coordinates* into *window coordinates* plus a *depth value* (also known as *z-value*), where the window coordinates correspond to what the user sees on her screen and the depth value defines how far away that vertex is from the user.

[3] Ray tracing and rasterization use the same terminology for light sources. However, please note that rasterization algorithms, unlike ray tracing algorithms, do not actually trace the rays originating from the light source. Rather, the influence of a light source on the scene is calculated through closed mathematical equations (approximations).

2. The second stage executes the *geometry shader*. This is an optional stage, whose purpose is to generate new graphics primitives or suppress existing ones. It can be used to simplify or improve their performance of specific graphics algorithms that would otherwise require multiple passes to implement. If this stage is missing, the results of the previous stage remain unmodified.

3. The third stage executes the *fragment shader* for each fragment (pixel) visible on the application window. It controls the final color of that fragment, by *sampling* the various parameters of the corresponding material, as well as the nearby light sources.

If the complexity of the scene is high, it may not be possible to calculate the final color of each fragment in a single pass. In that case, intermediate results are stored and used as input to later passes. In the case of this project, up to 16 consecutive passes were required to determine the final image that is displayed to the user!

Apart from these programmable stages, modern graphics pipelines include additional non-programmable stages that are required for rasterization (see figure 2.4). Epigrammatically, the most notable stages include:

- *Primitive clipping*, wherein primitives are clipped to the boundaries of the defined window *viewport*. This stage occurs right after geometry shading.

- *Polygon culling*, wherein the *back faces* of polygons are removed from the graphics pipeline. Back faces are normally invisible in continuous, convex shapes, and their removal improves performance without impact on visual quality. However, this optimization must be disabled for scenes that contain concave or non-continuous shapes. This stage runs after primitive clipping.

- *Rasterization*, wherein polygon faces are finally rasterized. This is the last stage before fragment shading.

- *Scissor, alpha* and *stencil testing*, wherein the result of a fragment shader is potentially discarded, depending on specific conditions. The scissor test checks whether the fragment falls within a specified rectangular region (if not, it discards the fragment). The alpha test compares the *alpha* (i.e. translucency) value of the fragment to a threshold and discards the fragment if the comparison fails. The stencil test performs the same test using a *stencil buffer*. If a fragment passes all three conditions, then it is *accepted*.

- *Depth testing*, wherein the depth-value of a fragment is compared to the contents of the *depth buffer*. By default, if an incoming fragment at coordinates (x, y) has a higher depth-value than the contents of the depth buffer at coordinates (x, y), then the incoming fragment is discarded (because, conceptually, it belongs to an object that is obscured by an object that was rendered previously). Conversely, if the depth-value of the incoming fragment is lower than the depth buffer, then it is accepted and the depth buffer is updated with the new value.
  As an optimization, some GPUs perform depth testing before actually running a (potentially computationally complex) fragment shader, if they can conclusively determine that this fragment will fail depth testing.

- *Blending*, which controls how an accepted fragment affects the color of the previous fragment at the same coordinates. By default, no blending is performed and the accepted fragment fully replaces the previous fragment. By enabling blending, the new fragment can be mixed with the old fragment in order to approximate the effects of lights, shadows and translucent materials (e.g. a thin curtain or a colored glass; Porter and Duff, 1984).

- *sRGB conversion*. This stage, if enabled, will transform the fragment from a *linear color-space* to the *sRGB color-space*. Applications that require high degrees of color accuracy, for instance photorealistic Computer-Aided Design (CAD) applications, must perform careful color management on each part of the graphics pipeline. sRGB is a standardized non-linear color-space designed for use on monitors, printers and the internet. By default, GPUs operate in a linear color-space, which is easier to work with (operations such as color addition are not defined in non-linear color-spaces) - by converting to sRGB as a final step, applications can ensure their images look correct regardless of the output device (Stokes et al., 1996).

- *Render operations* (ROPs). The final stage stores the results of a complete pass to *video memory* (VRAM) for display or for use in later passes.

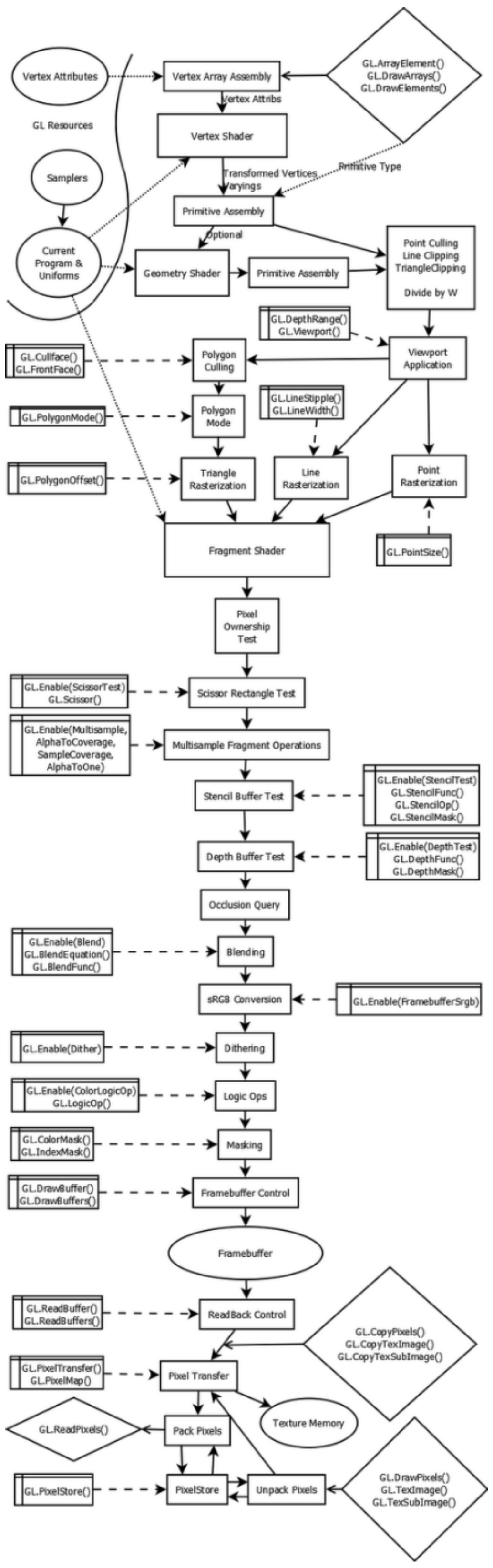


Figure 2.4 - The complete OpenGL pipeline. (Image courtesy of OpenTK)

Modern rendering techniques have been developed to improve the performance of the graphics pipeline on complex scenes. The classical approach, called *forward rendering*, orders scene elements by their materials and influencing lights, and feeds each *batch* of objects in the pipeline. The pipeline is re-configured between each batch to execute different shaders - a costly operation that adversely impacts the performance of complex scenes with many different materials.

In contrast to forward rendering, *deferred rendering* splits the rendering process into two parts: first, it visits every scene object using specific shaders to build the so-called *g-buffers*, intermediate buffers that store information for each object in *screen space* (i.e. after the perspective projection). The pipeline is then reconfigured and these g-buffers are used as input sources to calculate the final result.

Deferred rendering is higher overhead than forward rendering for simple scenes. However, as the light and material count increase, forward rendering takes the advantage. Almost all modern 3d games use a deferred approach.

2.3.3 Image output

The final step in the graphics pipeline is the presentation of the calculated image to the user. The image that is visible on the display device of the user is stored in a specific region of VRAM that is called the *front buffer*. In order to avoid displaying a half-finished image to the user, the results of the graphics pipeline are written to a different, invisible VRAM region that is called the *back buffer*. Once a new frame has been completely rendered, the front buffer and the back buffer are *flipped*, and the new frame is presented to the user. The old frame is then *cleared* and the process starts anew.

The key parameters of the displayed image are its resolution, its color depth, its refresh rate and its latency. Depending on the choice of output device, the user may see a monoscopic or stereoscopic image, and a different field of view.

The *resolution* of the image defines how many distinct pixels (fragments) it comprises. It is typically defined as width x height, where width and height are specified in pixels. Higher resolutions allow for higher levels of detail and fewer aliasing artifacts in computer graphics.

The *color depth* of the image defines how many distinct colors each pixel can display (Keith, 2007). It is usually specified in bits-per-pixel (bpp), where for a given n bpp, 2^n distinct colors can be displayed. Modern computer graphics typically use 24 or 30 bpp for 16.7 million or 1 billion colors, respectively. Images that are meant for post-processing (for instance, the results of intermediate passes in the graphics pipeline, as described above) can sometimes use higher such as 48, 64 or even 128 bpp. The human eye can discern at least 10 million colors (Judd and Wyszecki, 1975).

The *refresh rate* of the image refers to the number of times the image is updated in a single second (Marsh, 2001; msdn.microsoft.com). The most common refresh rate for computer equipment is 60 Hz, or 60 updates per second. Modern stereoscopic monitors are typically rated for 120 Hz (60 Hz per eye). Real-time computer graphics in the context of VR must maintain a constant or nearly constant refresh rate in order to maintain user immersion and avoid inducing motion sickness.

The *latency* of the image, sometimes referred to as input lag, describes the length of time between a user action and its reflection on screen. Depending on the application, latencies higher than 100 milliseconds (around 6 updates at 60 Hz) can adversely affect user immersion and her ability to perform tasks within the virtual environment (MacKenzie and Ware, 1993; Ware and Balakrishnan, 1994).

The field of view (FOV) defines the extent of the visible virtual world that can be displayed onto a display device (Jeng-Weei Lin et al., 2002). It is measured in degrees and depends on the size of the display device and the distance between the display device and the user. HMDs have FOVs that range between 30° - 150°. Computer monitors lie in the range between 30° - 75° and CAVE systems can reach up to 360° (fully immersive FOV).

Stereoscopy is a technique that mimics binocular vision in order to provide depth perception. It works by presenting two different 2d images to the viewer, one for each eye. The brain combines these two images and to gain the perception of 3d depth. This can be achieved in several different ways. Common ones include the use of shutter glasses that shutter each consecutively, accompanied by a computer monitor that alternates between the two images in sync with the glasses (this is also known as active stereoscopy); the use of two distinct monitors or projectors, each with different polarization, and a pair of polarized glasses to distinguish between the images (this is also known as passive stereoscopy); or the use of an autostereoscopic monitor that can direct each image to a different eye without additional equipment.

A stereoscopic display requires two distinct front buffers and two back buffers (usually called front-left, front-right, back-left and back-right).

Chapter 3: Human - Computer Interaction

Optimizing the communication between humans and machines is the main goal in the field of Human - Computer Interaction (HCI). HCI is a highly interdisciplinary field as it takes into account human factors, like psychology or linguistics in order to design machines or interfaces that can be used by an optimal way. Various aspects of HCI involve the design of computer programs and interfaces, the development and optimization of new techniques and devices for interacting with users and aim in general to allow humans to use computers or machines in general in the most ‘natural’ way, without having to adapt their cognitive functions to them.

In order to design such an optimal device one needs to take into account the way this device will be used. Factors about the environment of use, or the type and limitations of the machine are combined with information about the number and particular needs of users. For example, it should be examined whether an application is designed for clinical populations with specific impairments or requirements or for the general public.

A particular field of HCI that is mostly relevant to the present thesis involves virtual and augmented reality environments (see also chapter 1). HCI research in this context aims at allowing humans to optimally navigate within such environments and interact with them for achieving pre-defined goals. Recent advances that aim at a more natural interaction with augmented reality environments allow users to provide directly mental commands for interaction (Brain - Computer Interfaces). Such interfaces were examined in the context of this thesis and will be presented in the following.

3.1 Brain - Computer Interfaces

Brain - Computer Interfaces (BCI) mainly aims at providing an alternative pathway to mental commands and/or movement intentions and to use brain activity for controlling, instead of body muscles, external devices (Wolpaw et al., 2002). Such devices span from small robots and wheelchairs with some degrees of artificial intelligence (Allison et al., 2012; Friedman et al., 2010) to computer applications, specifically designed for this type of control. Research in the field of BCI started already since 1970, but the first BCI that worked in real time with humans only appeared in 1990. Ever since there has been a tremendous increase in research on this domain (figure 3.1).

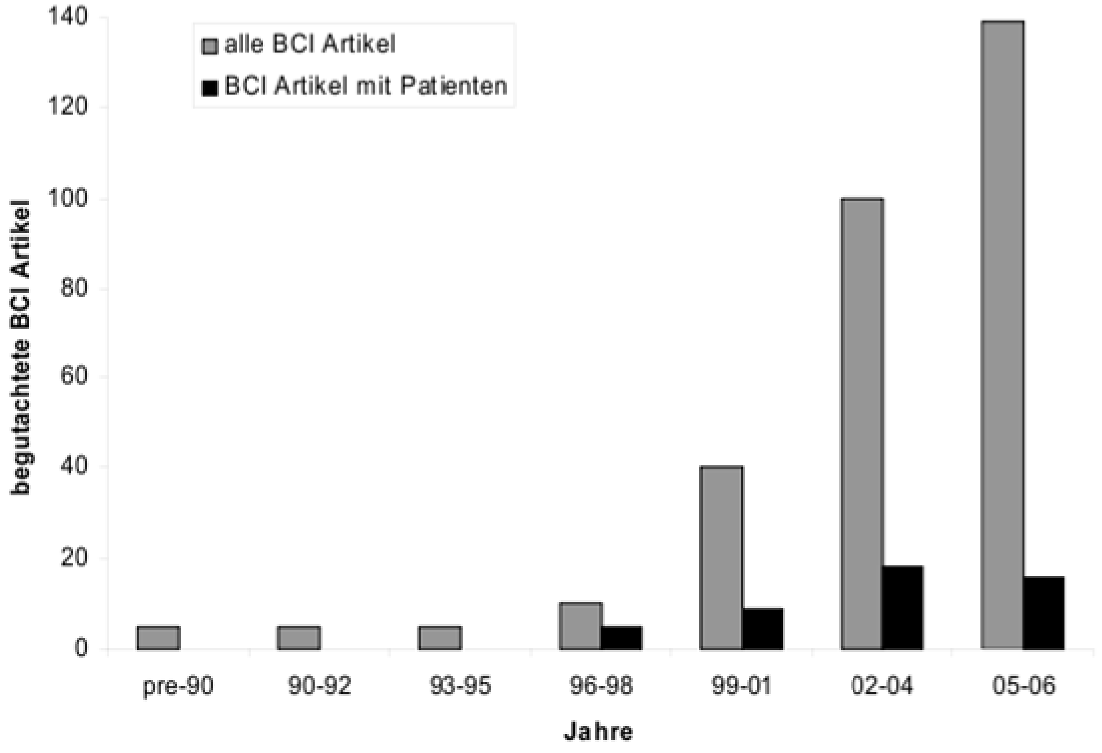


Figure 3.1 - Number of publications in the field of BCI.

One major aim of BCIs in general is to assist people with severe disabilities who have limited or no control over their body muscles, by providing alternative ways of movement or communication. For example, it is now possible to use brain activity, as measured with electroencephalography (EEG) for controlling wheelchairs (Galán et al., 2008; Pires et al., 2008; Long et al., 2012) or small robots (Perrin et al., 2010; Anderson et al., 2012; Ron-Angevin et al., 2012). Other applications of BCIs emphasize more on communication and therefore use brain signals for composing text (Belitski et al., 2011; Ortner et al., 2011; Jin et al., 2012; Hwang et al., 2012) or browsing the internet (Yu et al., 2012). Finally, a large field of BCI research focuses on controlling virtual reality environments (Leeb et al., 2007; Renaud et al., 2011; Zhu et al., 2011; Cheron et al., 2012; Ortner et al., 2012). The use of BCI in virtual environments has several applications, spanning from pure entertainment to rehabilitation of patients with motor deficiencies (Leeb et al., 2007; Hashimoto et al., 2011; Cheron et al., 2012) or psychiatric patients (Renaud et al., 2011; Zhu et al., 2011).

3.1.1 The composing parts of a BCI

Like any communication system, a BCI roughly consists of an input, an output and parts/protocols allowing to transform the received input to the desired output (figure 3.2). The typical input in a BCI system in neural activity, as it can be recorded by means of EEG, functional Magnetic Resonance Imaging (Renaud et al., 2011; Sorger et al., 2012; Weiskopf 2012) or intracranial recordings (Schalk and Leuthardt, 2011; Thongpang et al., 2011). Alternatively, physiological signals of non-neural origin, such as eye movement activity can also be used as or assist BCI input (Usakli et al., 2009; Usakli et al., 2010). The use of such signals will be presented in more details in the following.

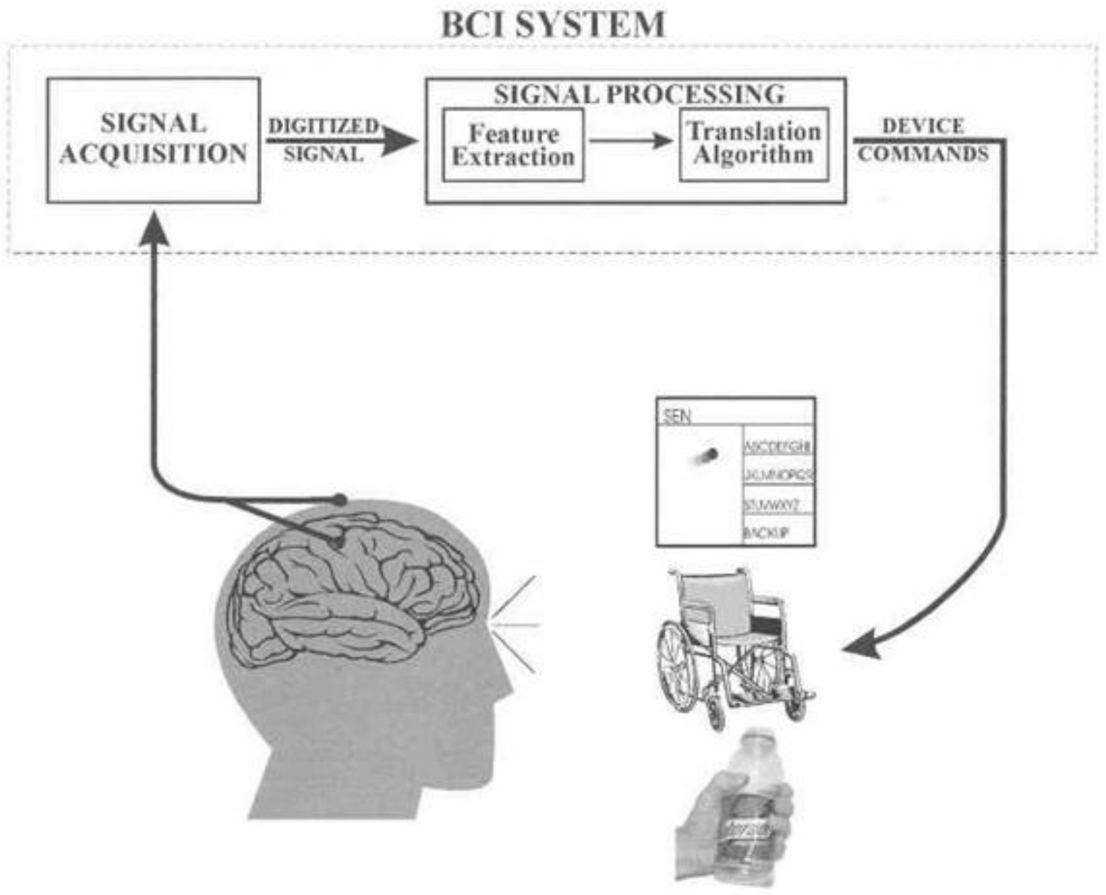


Figure 3.2 - Schematic representation of a typical BCI system (Wolpaw et al., 2002).

The recorded input signals from the user are amplified and digitized and then processed by signal-processing algorithms that are specially developed and allow online communication (Wolpaw et al., 2002 for a review). This signal processing aims at decoding the user's intentions from his/hers neurophysiological activity and usually consists of two-steps: first, a feature extraction step and second a classification algorithm.

Often neurophysiological signals are embedded in noise or contain multiple dimensions, which might not be fully relevant to the task. The feature extraction step aims at eliminating such noise which can be generated by the user (for example movement artifacts or muscular noise in the case of EEG) or can be emitted by other sources in the environment (for example electric noise). After this pre-processing and before classification the most relevant features of the signal are extracted, often by dimensionality reduction methods, like Independent Component Analysis or Principal Component Analysis which aim at reducing complexity in the data while keeping a maximum of relevant information. In general, signal processing can be done in the temporal (which is typically the case for evoked potentials, as recorded by EEG) or the spatial domain (for example electrodes' location as in Yang et al., 2012), or even a combination of the two (Dias et al., 2010; Higashi and Tanaka, 2011).

The next step in a typical BCI consists of translating the extracted features to plain machine commands (figure 3.2) and can be easily done by applying various classification techniques (Tomioka and Müller 2010; Manyakov et al., 2011; Trad et al., 2011; Saa and Çetin 2012). These classification techniques can be linear or non-linear, but in any case emphasize on speed of computations, for being able to communicate on real-time. The output of the classification algorithm provides an indication about the user's intentions and can be immediately executed as a command.

Neurophysiological signals naturally share some common features across subjects but have also a certain degree of variability in the population, which depends on the type of signal and/or task. In general, neural responses to high-level cognitive tasks, such as decision-making represent high degrees of variability across subjects (Tzovara et al., 2012). This is the reason that more and more signal-processing approaches take into account this variability and aim at optimizing and adapting the classification and feature extraction algorithms for each user separately.

Another important factor that needs to be taken into account by signal-processing algorithms is that of re-usability. When a subject attempts to use a BCI system for the first time various parameters of the system are adapted to his/her responses through a training phase that extracts individualized features from neurophysiological signals, which can then be used in a testing phase for actually controlling a device (Iturrate et al., 2011). However, if the subject aims at re-using this device, it is very possible that its neural signals will be different. In fact, it is well known that they change depending on the time of the day, hormonal levels, fatigue etc (Wolpaw et al., 2002). Moreover, effects of neural plasticity might induce a change in neurophysiological responses, either at short-term (from the beginning to the end of a session) or at short-term (across different sessions/days). To address this issue, the stability and reliability of any BCI algorithm should be tested over prolonged sessions and many days.

3.1.2 Hybrid BCIs

Typically, BCI research is performed in scientific environments, where variability in sensory inputs is limited to the necessary minimum and well-controlled. The reason for this is that neural signals, as they are recorded for instance with EEG are very subtle and volatile. Even a minor experimental manipulation might therefore change these neural signals and diminish BCI performance. For example, when a BCI is controlled by neural responses to visual stimuli, such as in the case of flashing letter pallets (Belitski et al., 2011; Ortner et al., 2011), even a minor change of attention or visual fixation might result to a distortion of the neural response and therefore to an erroneous output of the BCI. This issue is particularly prominent in the case of BCIs that aim at controlling VR environments.

The aim of VR environments is to simulate, as realistically as possible a real-world setup, in which we constantly receive visual and auditory inputs from a variety of external sources. Of course the experimenter can set strict rules to their subjects, but attention or gaze can be easily diverted from the main focus and thus EEG signals are prone to distortions. One way to address this issue is through the use of hybrid systems, which rely on a combination of physiological signals for communication and not just brain activity alone (Usakli et al., 2009; Punsawad et al., 2010; Usakli et al., 2010; Ianez et al., 2011). Such signals may involve electromyogram, which records voltage potentials of various muscles, which can be used to detect preparation for a movement, measures of conductance of the skin, in order to detect sweating which signals stressful situations, heart activity via electrocardiogram or eye movements. Among all these different signals and techniques, eye movements are of particular interest for the present thesis and the principles of various methods for recording them will be presented in detail in the following.

3.2 Saccadic eye movements

Humans and most animals constantly move their eyes in their surrounding space in order to detect and focus on points of interest. The reason for this constant movement is to bring such points or objects of interest in the fovea (figure 3.3), which is located in the central part of the retina and has a very high spatial resolution. Such eye movements thus allow stabilize a visual input in the retina even in the case of moving objects.

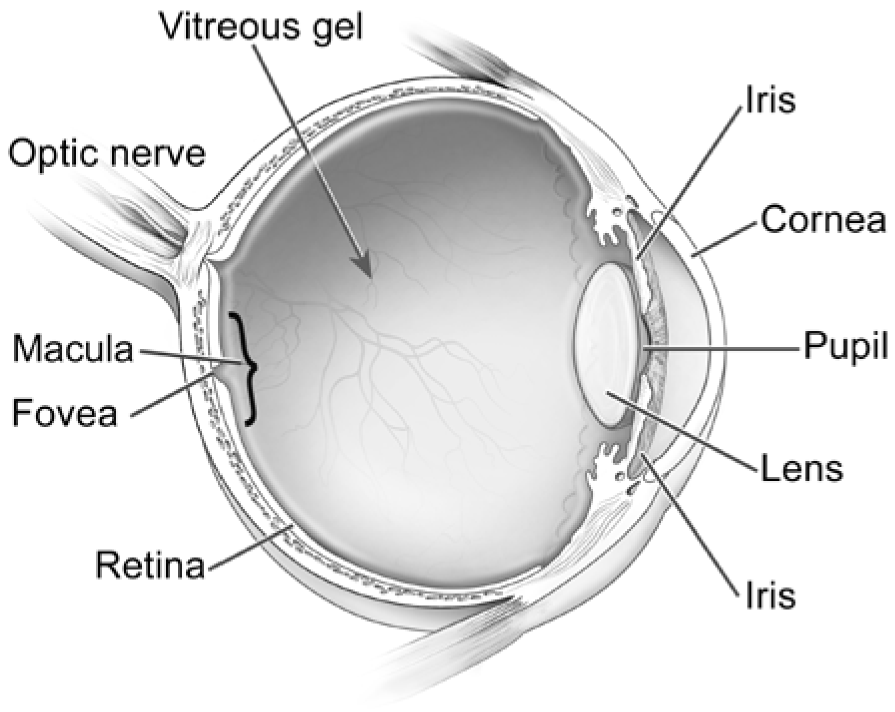


Figure 3.3 - Anatomy of the eye (Figure from freedomscientific.com).

One particular type of eye movements that is of interest for the present thesis are the so-called saccadic eye movements or saccades. Saccades were first observed in 1880 by a french Ophthalmologist, Émile Javal. Javal studied eye movements during silent reading by placing a mirror at the edge of a text page. This simple technique revealed that eyes do not continuously move but rather execute a series of single movements (Javal, 1878).

Saccades can be divided in two general categories, voluntary and reflexive saccades (Pierrot-Deseilligny et al., 2003). On the one hand, voluntary saccades are a response to a specific command and are initiated after a conscious decision of the subject to move his/her eyes. They are executed either towards a target that exists in the surrounding space, towards a target that subjects expect to see, or towards a target that was in space but is not there anymore (memory saccades). This type of saccades involve, prior to their execution, a conscious decision, which is why they are relatively slow. In this same category of voluntary saccades also belong the anti-saccades, which are eye movements that are executed on the opposite direction of a certain visual stimulus. When such a stimulus - target appears subjects tend to automatically move their eyes towards it, which is why antisaccades require high levels of attention in order to inhibit this automatic response towards the target.

On the other hand, reflexive saccades are caused by the sudden appearance of an external stimulus; subjects are not specifically instructed to execute them. Such a stimulus could be a visual target that appears unexpectedly in the surrounding environment. Reflexive saccades are much faster than voluntary ones since they do not require a conscious decision to be made prior to their execution.

In order to study saccades, researchers typically measure their amplitude and latency. The amplitude of a saccade is in fact the angle along which the center of eye moves for executing the saccade. The saccade latency is the temporal interval between the appearance of an external stimulus that triggers a saccade and the saccade's onset. Such behavioral measures can be easily assessed when recording saccadic eye movements with various methods that will be described in the following.

3.3 Recording saccades

Ever since 1880 and the observation of the first saccades through their reflection on a mirror (Javal, 1878), a series of more sensitive recording methods have been developed and are now easy to use. Such methods include the Electrooculogram (EOG), Infrared Reflection Devices and video oculography. The advantages and limitations of each of these methods will be presented in the following.

3.3.1 Electrooculogram

EOG takes advantage of the fact that the human eye acts like a dipole parallel to its optical axon. Neurons and light receptors in the retina result in its being more negative that the cornea (figure 3.3), by approximately 6mV (corneo-fundal potential; Marmor et al., 2011). This results in a dipole which follows the rotation of the eye, as the eye moves. This movement thus creates small voltage differences (in the order of magnitude of few decades of μVs) on the surface of the skin. In this sense, an eye movement towards the left will increase the potential on the external corner of the left eye (left canthus) and will decrease the potential on the right canthus. This voltage difference can be measured by simply placing two electrodes bilaterally, on the two canthi (Figure 4).

EOG consists of measuring exactly this difference in voltage potentials through a pair of electrodes (Marmor et al., 1993; Malmivuo and Plonsey, 1995; Marmor et al., 2011). By only using these two electrodes one can record horizontal eye movements, but it is also possible to place additional electrodes below and above of one or the two eyes and to also record vertical movements. This type of recording is however more susceptible to be contaminated by artifacts (muscle activity) during eye blinks.

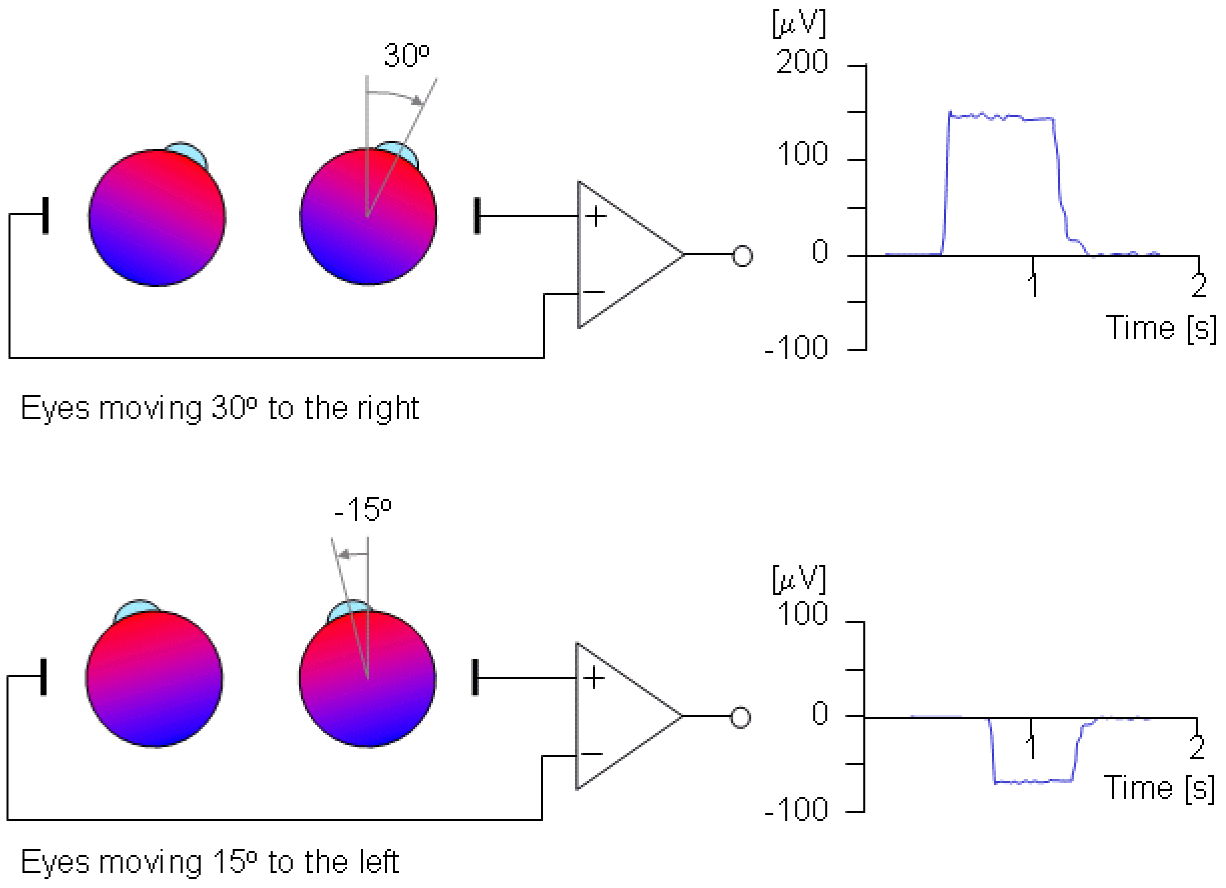


Figure 3.4 - Schematic example of EOG recording.
(Figure from Malmivuo and Plonsey, 1995).

Before placing the electrodes, the skin needs to be locally cleaned so that any dead cells are removed. This reduces the impedance between an electrode and the skin, which according to instructions for EOG in the clinics, should not exceed 5kΩ (Marmor et al., 2011). A reference electrode is also placed on the subject's forehead or earlobes. The signal recorded from the electrodes is then amplified (figure 3.4). A typical sensitivity of EOG measurements can be around ±2°, with a maximum recorded angle of ±70° (Malmivuo and Plonsey, 1995).

Overall, the EOG represents an inexpensive, easy to setup and portable way to record eye movements that is not affected by lighting conditions and has low requirements in computing power.

3.3.2 Infrared Reflection Devices

An alternative way to record eye movements is through Infrared Reflection Devices (IRD). Such devices employ light-sensitive receptors with high spatial resolution and measure the intensity of infrared light that is reflected by the eye. IRDs rely on the fact that the iris and pupil of the eye (figure 3.3) reflect less light than the sclera (the white part of the eye). It is therefore possible to measure this difference in reflection by placing the light-sensitive receptors of the IRD close to the eye (few cm apart). Usually an IRD system is wearable and can be attached on the user's head. This also makes it less sensitive to head movements and gives a first advantage compared to EOG, where the head needs to be well stabilized during the whole recording. Moreover, IRD recordings allow a better resolution than EOG but are more susceptible to eye blinks (Eggert, 2007).

3.3.3 Video Oculography

Video Oculography (VOG) detects the pupil of the eye (figure 3.3) through a digital video camera (Gans 2001). In order to achieve a high precision with VOG it is important for the camera to be stable relatively to the head and to not be influenced by head movements. For this reason VOGs are often mounted on goggles that subjects can wear and still freely move their heads. In the first VOG devices, the camera was mounted on one goggle and allowed subjects to only see through the other eye. This approach was limited as it did not allow record eye movements in realistic situations. More modern devices use dichotic filters on both eyes, which allow the subject to see normally but also reflect infrared light, allowing the camera to record the position of their eyes (Gans, 2001). In summary, VOG allows a high spatial resolution (~0.05°) with minimal requirements for subjects' preparation. However, the cost of such a camera is significantly higher than the cost for an EOG and eye tracking algorithms tend to be more complex.

Part B: Experimental Part

Chapter 4: Experimental Part A - Virtual Environment

This chapter discusses the design and development of the OpenBCI Labyrinth. It focuses on the virtual environment from a simulation and rendering standpoint.

4.1 Concept

This project had two goals: first, to realize a virtual environment that would be suitable for navigation through eye movements and, second, to implement at navigation using electrooculography. As we saw in chapter 3, several methods to track eye movements exist, each with different advantages and disadvantages. Therefore, the virtual environment should take a form that is suitable not only for a particular method, but that can be meaningfully navigated using different eye tracking methods.

One environment that fulfills these requirements is a labyrinth. Labyrinths offer several advantages over other environments: they are relatively simple to construct and visualize using a computer; their concept is both timeless and easy to explain (i.e. “find the exit!”); while they still allow for meaningful user interactions, by exercising user memory, navigation and situational awareness skills. These advantages explain their popularity not only in research, but also in games and media.

For the purposes of this project, labyrinths present an additional advantage: an input method can be devised to allow navigation using a single axis of freedom. Additional axes can provide additional capabilities for navigation, but only a single axis is required to solve a planar labyrinth (i.e. turn left, turn right or maintain current direction), making this environment especially suitable for electrooculography-based navigation (more on this in chapter 5).

In order to increase user immersion, special attention has been paid to the visual appearance of the environment. The visual motif is inspired by the Theseus and the Minotaur mythos of greek mythology, where the courageous hero, Theseus, navigated a labyrinth to defeat the Minotaur at its heart (see figure 4.1). The walls of the labyrinth are decorated with frescoes depicting running men and meanders, interspersed with golden marble pillars in the Doric order. The floor is made of worked red sandstone, while the ceiling is open to reveal a view of a clear night sky and the moon. The labyrinth is placed in the middle of a calm lake, which makes for an interesting interplay of reflections and light. The scene is illuminated by the moonlight and by a "fairy" that helps the user navigate the labyrinth. The viewpoint is that of a 180 cm tall human.

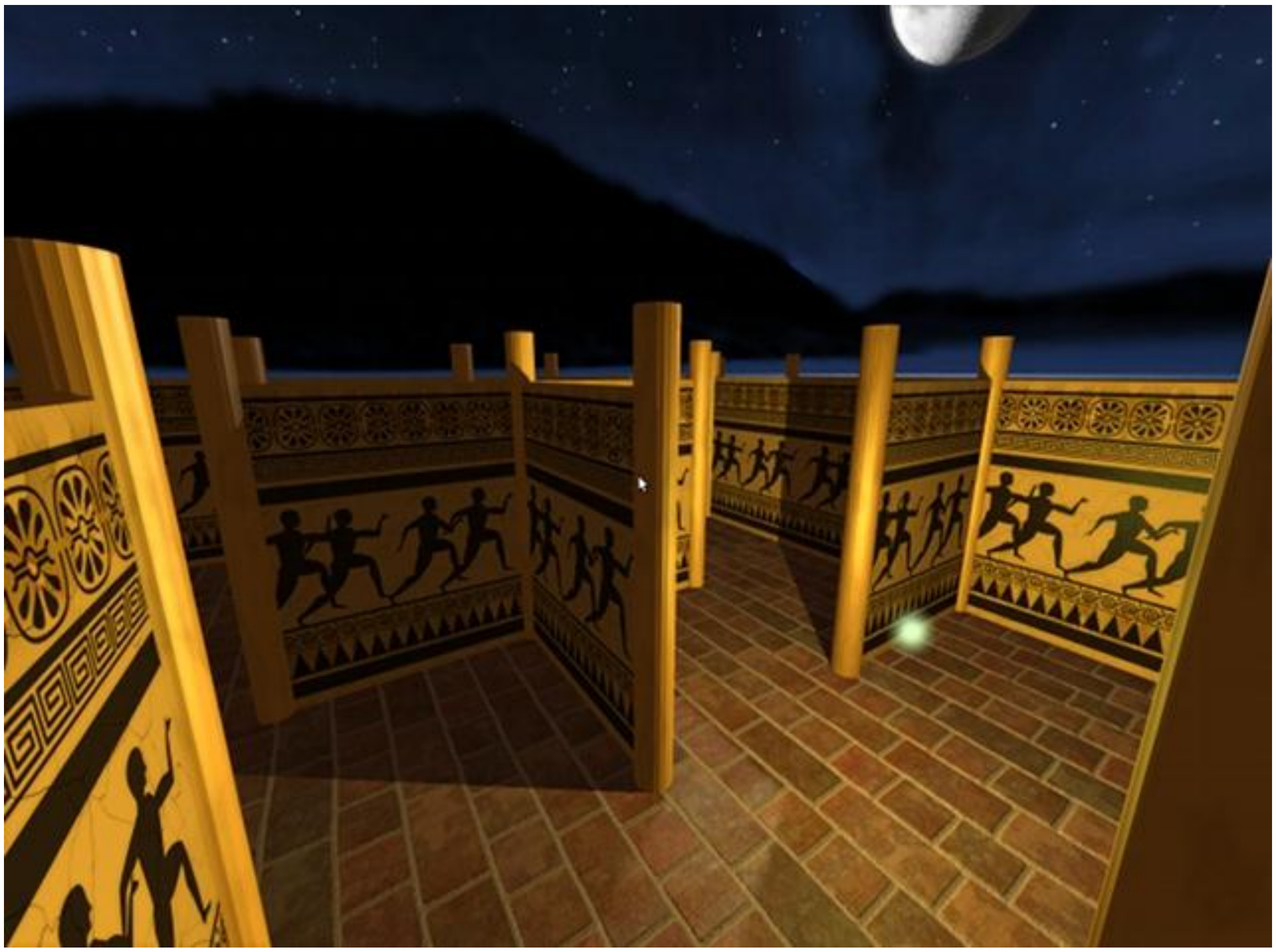


Figure 4.1 - The realized concept of the labyrinth.

4.2 Technologies

This project was built from the ground up using the C# programming language and OpenGL. The Open Toolkit library (OpenTK) provided the binding between C# and OpenGL.

OpenGL (Open Graphics Library) is a cross-platform programming interface for computer graphics. The first versioned was developed by Silicon Graphics in 1992. As of August 2012, it has reached version 4.3 and is being managed by the Khronos Group, a non-profit consortium comprising major computer graphics companies. OpenGL is commonly used in virtual reality, scientific visualization, CAD software and cross-platform computer games. A lighter version, OpenGL for Embedded Systems (OpenGL ES) can also be found in modern mobile phones.

C# is a modern, general-purpose programming language that provides a good balance of performance, safety and ease of use. It was designed by Anders Hejlsberg for Microsoft and, as of 2006, is an ECMA and ISO standard (ECMA-334; ISO/IEC 23270:2006).

The Open Toolkit library is an open-source project that allows C# applications to access the OpenGL and OpenAL (Open Audio Library) APIs. It also provides a *platform abstraction layer* that allows applications to construct OpenGL windows and read input from the keyboard, mouse or other input devices in a cross-platform manner. It was designed by Stefanos Apostolopoulos in 2006 and is, as of 2012, available for Windows, Linux and Mac OS X, as well as for Android and iPhone smartphones (Opentk.com).

4.3 Design of the environment

The first step towards the realization of this project is the construction of the geometry of the 3d virtual environment. This is usually achieved through specialized 3d modeling packages, such as 3d Studio Max, Maya or Blender3d. More rarely, virtual environments are constructed from mathematical functions, simulations or other data sources (e.g. from the results of computer-assisted tomography).

4.3.1 Geometry

Labyrinths lend themselves well to generation through computer algorithms. Different algorithms result in different labyrinth *texture*, which is defined in terms of the following factors (Foltin 2011):

- *Bias*, which refers to the labyrinth being easier to navigate in one direction versus another. For instance, a labyrinth with long, uninterrupted horizontal passages but short, interrupted vertical passages would indicate the existence of horizontal bias.

- *Run*, which indicates the average length of straight passages. A labyrinth with a low run factor would require frequent turns to navigate, while a labyrinth with a high run factor would consist of lengthy straight passages broken by infrequent turns.

- *Elitism*, which compares the length of the solution to the size of the labyrinth. An elitist labyrinth will have a short and direct solution, compared to its size, and may be harder to solve than a non-elitist labyrinth (i.e. it is more difficult to come across the short "correct path").

- *Symmetry*, which represents the existence of horizontal, vertical or rotational symmetry between the passages of the labyrinth.

- *River*, which defines the average length of passages leading to a dead end. A low river factor indicates many short dead ends. Conversely, a high river factor indicates few, but long, dead ends.

In this project, the "recursive backtracker" algorithm was chosen for the generation of the labyrinth. This is a *perfect* algorithm, meaning that every point of the labyrinth is reachable from any other point (i.e. the labyrinth forms a connected graph - there are no cut off regions), which offers a texture with a high river factor, low elitism and no bias. The solution of a recursive backtracker labyrinth is usually quite long and has a good proportion of straight passages and turns. It is also very fast to compute, making it ideal for real-time evaluation.

The recursive backtracker algorithm works as follows:

1. It divides the area of the labyrinth into square *cells*. One cell is marked as the starting point and one cell is marked as the ending point. The algorithm begins from the starting cell, with all labyrinth cells “walled off” (i.e. no pathways exist between cells in the beginning).

2. At each step, it randomly selects one of the four cells that are adjacent to the current cell. If that cell is walled off, it “carves” the wall between the current and the selected cells, creating a new pathway, and moves repeats step 2 using the selected cell as a starting point. If the selected cell is not walled off, it randomly tries a different adjacent cell. If no adjacent cell is walled off, it “backtracks” to the previous cell and tries step 2 again.

3. The algorithm ends when it backtracks back to the starting position and no walled off cells exist.

4. Given a random number generator (necessary for step 2 of the algorithm), any recursive backtracker labyrinth can be represented using exactly three numbers: its width and height, in number of cells, and the *seed* of the random number generator. Given these three numbers, the layout of a labyrinth can be reconstructed fully. This ensures the repeatability of experiments on specific labyrinth layouts, but also allows for a very simple way to generate new layouts.

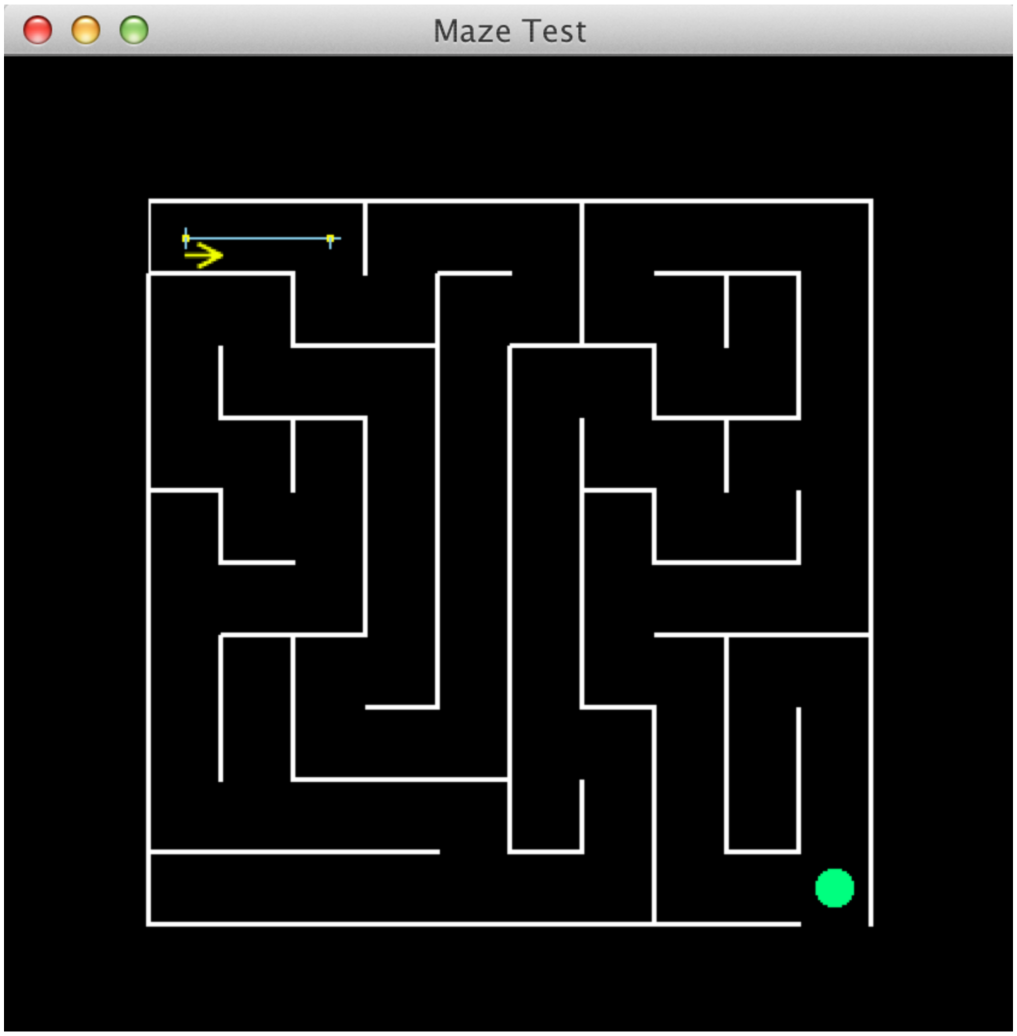
5. 

Figure 4.2 - Overhead view of an 8x8 recursive backtracker labyrinth, as implemented in this project.

Once the layout of a labyrinth is constructed, it must be turned into a proper 3d geometric representation made of graphics primitives (triangles). This is a simple matter of visiting each cell in the labyrinth and placing two orthogonal triangles, connected at their hypotenuse and placed vertically in respect to the floor plane, at the location of each remaining wall. The floor of each cell is filled by another two orthogonal, connected triangles, placed along the floor plane.

In order to make the labyrinth more visually appealing, the floor extends one cell outside the area of the labyrinth, and columns are placed at each point where two or more intersect. The columns are constructed of thin, vertical triangles, forming a cylinder around the intersecting line of the walls.

The sky over the labyrinth is represented by a *skydome*, a hemisphere made of hundreds of connected triangles. The lake is represented by a large plain, made of dozens of connected triangles. These elements, as well as the labyrinth columns, are constructed by solving mathematical equations; the whole world is generated by the computer as soon as the OpenBCI Labyrinth starts up.

4.3.2 Materials

The virtual environment is composed of seven different materials: the floor, the walls, the columns, the sky, the water, the moon and the fairy (see figure 4.3).

Each material is composed of up to 5 different parameters that control its appearance: a texture map, a specular map, a gloss map, a normal map and a height map. In order to conserve memory and improve performance, these parameters are packed into three maps and compressed using DirectX Texture Compression (DXT).

- Texture maps are stored standalone and compressed using the DXT1 format, achieving a 6:1 compression rate. Sizes range from 256x256 pixels (the fairy texture) up to 8192x4096 pixels (the sky texture). Most materials are 2048x2048 pixels.
- Specular maps and gloss maps are packed together and compressed using the DXT5 format for a 4:1 compression rate. Sizes are 1024x1024 and 2048x2048.
- Normal maps and height maps are packed together in a manner identical to specular and gloss maps.

The packing is achieved by storing specular and normal maps into the color (red, green, blue) channels of the compressed images, and the gloss and height maps into the alpha (translucency) channels, respectively.

Textures maps, normal maps and height maps were generated procedurally through Genetica Viewer™ for the wall, floor and column textures. The moon and the sky were based on high-resolution stock photos were used instead, and the water and fairy textures were created manually in the GNU Image Manipulation Program (GIMP). GIMP was also used to post-process all textures for color balance, contrast improvement and noise reduction. Specular and gloss maps were also created through GIMP, using the texture maps as a base.

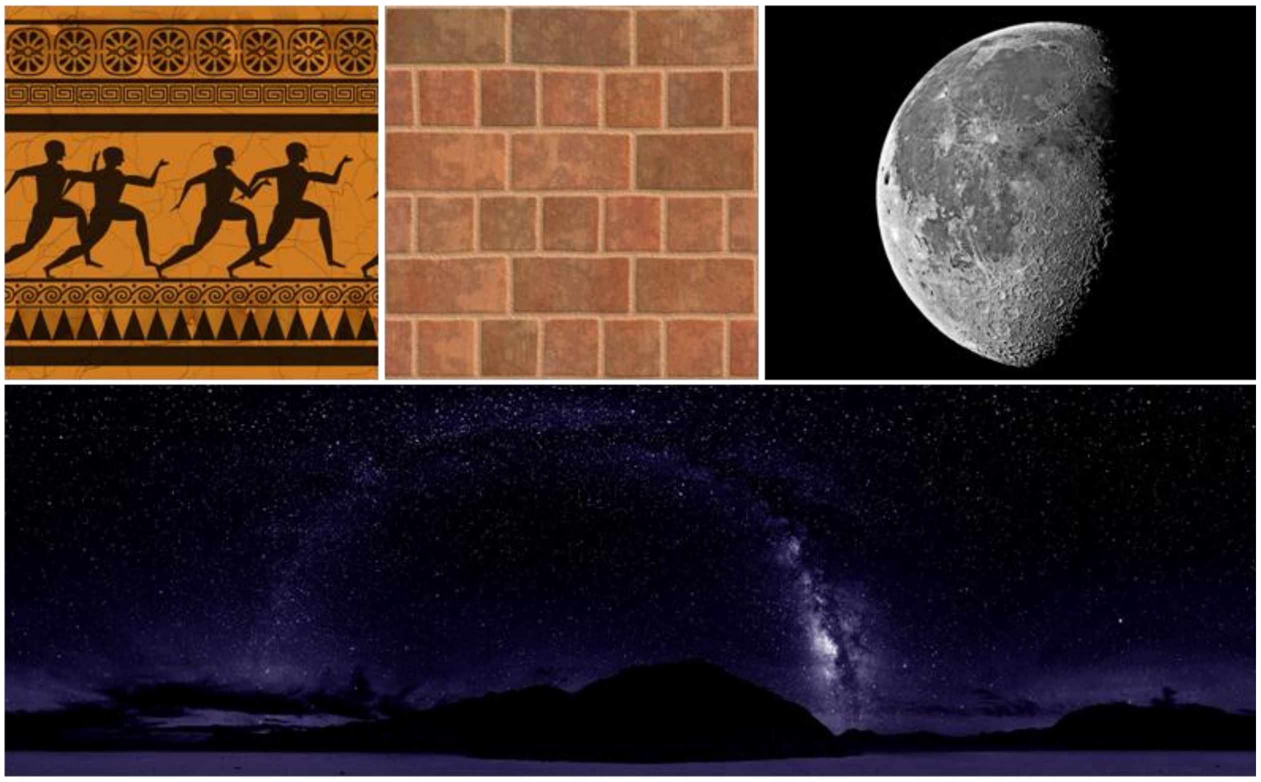


Figure 4.3 - Texture maps used in the virtual environment. ("Death Valley Night Sky" image courtesy of Dan Duriscoe.)

4.3.3 Lights

The environment is lit by two lights: the moon and a small light-emitting fairy that accompanies the user inside the labyrinth.

The moon provides low-level illumination which is mostly visible in dark areas outside the influence of the fairy, on shiny spots on the walls, as well as in far-away parts of the lake. It is represented by a directional light with a silvery color and moderate intensity.

The fairy is the main source of illumination in the scene. It is represented by a point light that follows the user's intention (more details on chapter 5), and has a bright warm cream color and high intensity.

Additionally, a low-intensity ambient factor is added to the scene, to avoid pitch black colors in shadowed areas. This approximates global illumination by starlight and indirect light bouncing off walls and the floor (see figure 4.4).

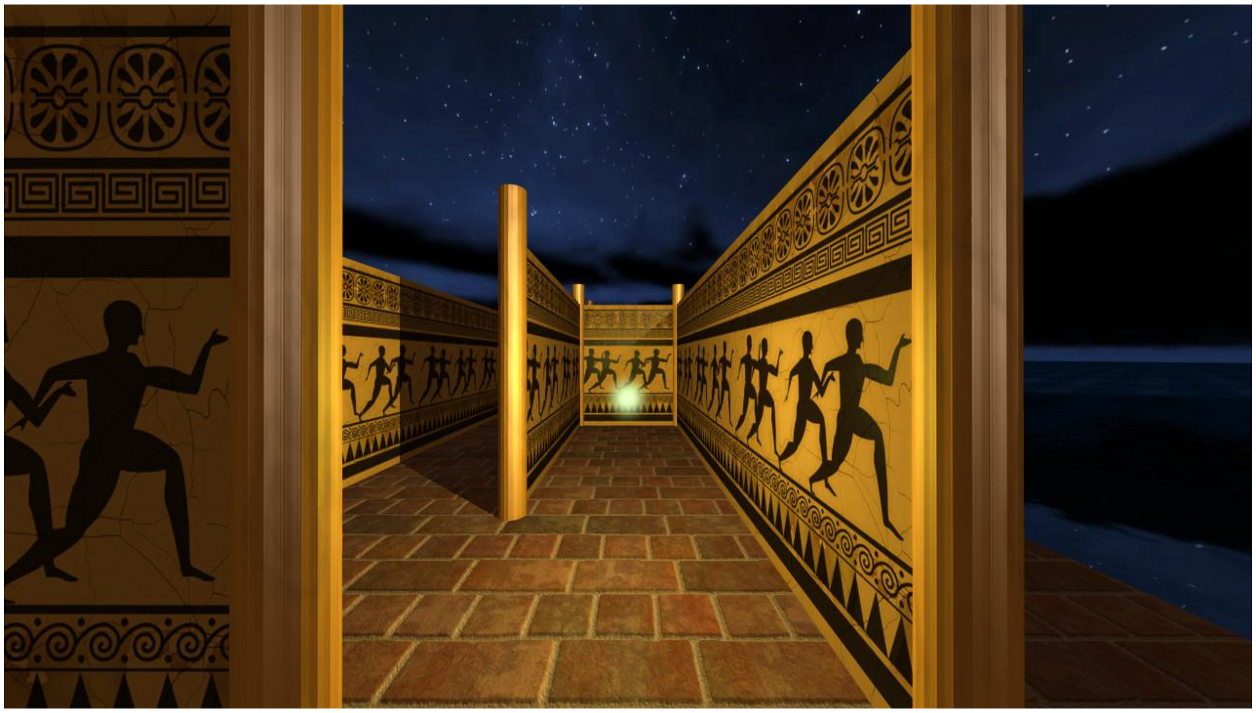


Figure 4.4 - The result of the lighting model used on the labyrinth. Moonlight is visible on the front side of the columns, while the fairy is the main source of illumination in the scene.

4.4 Implementation of the labyrinth

The OpenBCI Labyrinth is modular and consists of a VR engine and the actual VR application that utilizes the engine.

The VR engine was built from scratch using OpenGL and C#, through the Open Toolkit library. It provides a modern, high-level system to manage the world geometry, materials and lights, record user input and communicate with display devices, and is capable of stereoscopic rendering using shutter glasses or dual projections. A thin abstraction layer was developed to improve portability to different APIs, such as Microsoft's DirectX, should that be desirable in the future. The engine weighs at roughly 6000 lines of code.

The actual VR application sits on top of the VR engine. It provides the code that creates the labyrinth, interprets user input and simulates the virtual environment (i.e. water movement, walking, etc). It weights at roughly 4000 lines of code.

For each frame, OpenBCI Labyrinth performs three distinct steps:

1. User input processing
2. World updates (world simulation)
3. Rendering

Rendering occurs in the rendering subsystem, at a rate that is synced with the refresh rate of the display (typically 60 or 120 updates per second), using a “best effort” approach: if the GPU cannot keep up with the workload, the rendering rate will be reduced at an integer subdivision of the refresh rate (60, 30, 20, ...) World updates occur in the simulation subsystem, at a fixed rate of 60 updates per second regardless of the refresh rate, to ensure a constant simulation speed. In other words, an item moving at 1 m/s will cover a distance of 60 meters at exactly 60 second, regardless of the computing power of the underlying hardware[4]. A more powerful GPU will be able to render this movement more “smoothly” than a less powerful GPU - but the simulation time will be the same in either case.

User input is processed in the input processing subsystem, which runs in parallel to the simulation and rendering subsystems. This has two advantages: first, it minimizes the latency between the recording and the processing of user input; second, it improves the performance on modern multi-core CPUs. User input processing will be covered in more detail on chapter 5.

On a system with 2.6 GHz dual-core processor (Intel Core 2) and an Ati 4850 GPU, roughly 15% of the available computing power is spent in the input processing subsystem, 5% in world simulation and 80% in rendering.

[4] This assumes the availability of some minimum amount of CPU power (usually called “minimum requirements”) that is capable of running 60 world updates per second. For the purposes of this project, a 1.2 GHz or faster CPU is enough.

4.5 World simulation subsystem

World simulation takes place in the UpdateFrame event of an OpenTK.GameWindow. During a single UpdateFrame, the following actions take place:

1. The latest user action is retrieved from the input processing subsystem.

2. The desired world position of the user and the fairy are calculated according to the action from step 1. The new positions will either be identical to the current positions, or lie at a new cross-roads in the labyrinth.

3. Using the current and the new positions, a trajectory value is calculated for each movable entity in the world.

4. A Verlet numerical integrator is executed to calculate the correct position on this trajectory for the next simulation step (Press et al., 2007). Each item is advanced to its new position.

5. The simulation moves forward to the next time step (1/60th of a second, or 16.6 ms). If it is time to render a frame, the rendering subsystem is invoked; otherwise, we resume simulation from step 1.

This simulation is not physically accurate, as it does not model the effects of friction, mass or movement forces. However, by tweaking the trajectory generation algorithm in step 3, an aesthetically pleasing result can be achieved with significantly lower computing requirements. A physically accurate movement model is outside the goals of this project.

4.6 Rendering subsystem

The rendering subsystem is driven through the RenderFrame event of OpenTK.GameWindow. It implements a classical forward rendering pipeline with multiple passes.

The first step is to clear the back buffer. Afterwards, the following passes are executed:

1. Depth-only pass. The purpose of this pass is to prefill the depth buffer with the geometry of the labyrinth, in order to improve the performance of later passes. Color output is disabled and the labyrinth is rendered using a very simple depth-only fragment shader.

2. Moon shadow rendering pass. The labyrinth is rendered in depth-only mode from the viewpoint of the moon. This pass uses an orthographic project (since the moon is infinitely far away) and the results are stored for later use.

3. Fairy shadow rendering passes. The labyrinth is rendered in depth-only mode from the viewpoint of the fairy. This is identical to the 2nd pass, with the exception that it uses a perspective projection and is executed six times, once for each Cartesian direction around the fairy. The results are stored for later use.

4. Main rendering pass. This pass uses complex shaders to draw the labyrinth geometry (floor and walls). The fragment shader samples the scene materials and the results of the first three passes to implement the following effects:

    - Phong lighting model with normal mapping (Phong, 1975; Ernst et al., 1995).
    - Variance shadow mapping for shadows with soft penumbras (Donnelly and Lauritzen, 2006).
    - Steep parallax mapping with self-shadows via localized ray tracing to implement the unevenness of the floor stonework (Tatarchuk, 2006).
    - Specular mapping to control the shininess and specular intensity of light reflections on the different materials comprising the wall frescos and floor.

5. Secondary rendering pass. This pass renders the sky, moon and fairy using very simple shaders (plain texture material, not affected by lights).

6. Reflection pass. The main and secondary rendering passes are repeated using a mirrored, planar perspective projection. The results are saved for the next pass.

7. Water pass. This pass uses the results of the reflection pass, in order to render water reflections. It uses a moderately complex shader to implement the reflection and re-

fraction effects of light on water. The water surface is animated using moving low and high frequency sine waves, for a more realistic effect (Greene, 1986; Tessendorf, 2001).

8. Post-processing passes. The results so far are post-processed using the image processing filters, to improve visual quality:

    - Light bloom approximation, by applying a threshold filter to isolate bright spots on the original picture, and using a separable gaussian filter to turn these bright spots into soft halos.

    - Screen-space ambient occlusion (SSAO), by applying an edge detection filter to the depth buffer of the scene and overlaying its results on the original image.

9. Overlay pass (optional). The final pass overlays debugging information on the picture, such as the tracked eye location, the current and average frame rate, as well as the elapsed time.

If stereoscopic rendering is enabled, these passes are repeated twice, each for each eye.

Once rendering is complete, the back buffer (or back buffers, in the case of stereoscopic rendering) is flipped and the result is displayed to the user. Optionally, the rendered frames can be saved as uncompressed bitmap files on disk for further processing (video capture).

Chapter 5: Experimental Part B - Eye Movements

This chapter discusses the design and implementation of the input subsystem of the OpenBCI Labyrinth.

5.1 Concept

In order to achieve independence of the VR platform from the HCI system driving the simulation, an abstraction layer was designed based on network communication protocols. This approach improves reusability, because different HCI systems can use a common interface to communicate with the simulation.

The abstraction layer defines what user actions are available in the simulation and the network protocol that can be used to trigger them. In the case of the OpenBCI Labyrinth, the following actions are available:

- Turn 90° left
- Turn 90° right
- Move forward

This is the minimum[5] set of actions that are required to solve a 2d labyrinth.

The network protocol defines how user input is delivered from the input device to the VR simulation. The most important advantage of this solution is the ability to dissociate the actual input device from the computer running the simulation. This is quite useful for specialized input devices, such as EEG devices, which require specialized software and connectors that might not be available on a VR system.

[5] This is not strictly true, as the labyrinth can be solved using only "turn right" and "move forward" or "turn left" and "move forward". However, a HCI where the user can turn only right or left (but not both) would be rather uncomfortable - not to mention illogical - hence this set of three movements is the minimum *useful* set for solving a labyrinth.

OpenBCI Labyrinth defines a network protocol based on *UDP multicast*. UDP (user datagram protocol) multicast is a network communication technique for one-to-many data transmission. It is part of Internet Protocol, version 4 (*IPv4*), and its defining characteristic is that the sender does not need require prior knowledge about the existence and number of data receivers. UDP is a connectionless unreliable protocol, meaning that data packets that are lost during transmission are not re-transmitted; on the upside, transmission latency is minimal, due to the lack of connection (re-)establishment and data acknowledgment processes.

The use of UDP in this project confers the ability to connect the HCI device either to the VR simulation system directly or to a different system in the local network. In the first case, the input data is broadcast using the *loopback* device (i.e. a virtual network interface that broadcasts only to the system it belongs to, meaning that the data never enter the real network), but only devices compatible with the VR system can be used. In the second case, the input data is broadcast and can be captured by any listening device on the local network. The latency difference between the two approaches was measured to less than 1 ms, which did not cause any degradation in user experience.

Three different HCI methods were developed for use in this project. Different systems can be implemented using the same network protocol and basic approach described below.

5.2 Keyboard implementation

The first, and simplest, interaction method utilizes a typical computer keyboard. The *raw input* in this case, i.e. the unprocessed user input recorded by the device, is formed by key presses. A computer keyboard generates *key press events* whenever a key is first depressed by the user, and *key release events* once that key is released.

The input module works by directly mapping key press events to user actions: the left arrow key triggers the turn left event; the right arrow key triggers the turn right event; and the up arrow key triggers the move forward event.

5.3 Mouse implementation

The second interaction method relies on a typical computer mouse. The raw input in this case is formed by the position of the *mouse cursor* in the application window. The position of the mouse cursor is denoted by two numbers, representing the x and y coordinates of the cursor in Cartesian coordinates. The origin of the coordinate system is defined as the center of the application window, with (1, 1) defining the top-right corner of the window and (-1, -1) the bottom-left corner.

In this case, the coordinates of the mouse cursor simulate the offset of the eye from its resting position. This input interface receives a continuous stream of mouse movement events and feeds them to a decoding module that maps them to user actions. User actions are triggered when the mouse pointer is *fixated* at specific locations for more than a time threshold.

It is worth noting that the mouse interaction method is quite close to the electrooculography, described below, in terms of implementation. More details on the implementation of fixation will be covered below.

5.4 Electrooculography (EOG)

The final, and most interesting, interaction method implemented for this thesis was based on real-time EOG. This method provides some unique implementation challenges, as well as benefits, compared to other HCI approaches.

As mentioned in chapter 3, EOG consists of measuring the difference in voltage potential between the left and right canthus. In this thesis, this voltage differential was recorded using 2 EEG electrodes, placed bilaterally on the two canthi. A third electrode placed on the left earlobe to provide a ground voltage reading.

The EEG device in use was a BioSemi™ "ActiveTwo", equipped with 16 passive and 8 active electrodes. Samples can be captured at a rate of up to 16 KHz per channel. The analogue-to-digital converter (DAC) of this device quantizes signals using 24 bits per sample, which corresponds to a maximum digital resolution of 31 nV. This is called the least-significant bit

(LSB) resolution, which denotes the minimum voltage difference that can be captured directly by the DAC[6].

5.5 Communication with EEG

ActiveTwo communicates with a personal computer over a USB (universal serial bus) cable. The communication and data capture is mediated through *ActiveView*, an open-source data acquisition program written in LabVIEW, a programming environment designed for scientists and engineers.

ActiveView was configured to capture and broadcast samples at a 8 KHz sample rate. The minimum number of 8 electrodes was used: electrodes 1 and 2 carried the signal from the left and right canthus, respectively; electrode 3 carried the ground signal; electrodes 5-8 were ignored. At 24 bits per sample, the total data rate was 192 kilobytes per second.

5.6 Data analysis

The purpose of data analysis is the mapping to the acquired raw data into the three user actions defined by the virtual environment.

The first step was the calculation of voltage differences between each electrode and the ground electrode. The signal of the ground electrode (electrode 3) was subtracted from the signal electrodes 1 and 2, and further calculations were based on these new signals.

Several approaches were tested for the interpretation of the EOG signals. The most promising results were provided by the last, and simplest, approach tested: the voltage difference was mapped directly to a horizontal offset on the screen. An offset of -1 would correspond to the left edge of the screen; an offset of +1 to the right; and an offset of 0 to the center. A cali-

[6] Due to the repeatability of signals evoked by a stimulus, a higher signal-to-noise can be achieved by sweeping an evoked signal multiple times and summing the results.

bration step was added to the beginning of the simulation to determine the voltages corresponding to the each offset.

The screen was then divided into three regions, left, center and right. By default, an offset ranging between -1.0 and -0.3 corresponds to the left region; an offset between -0.3 and +0.3 corresponds to the center region; and tan offset from +0.3 to +1.0 corresponds to the right region. The thresholds corresponding to each region are configurable and should be modified to fit the size of the screen (a small screen would require a larger center region).

The three actions in the environment were mapped one to one with the three regions: turn left was mapped to the left region, turn right to the right region and move forward to the center region. The actions were triggered if the user was fixated on one region for longer than a specified *dwell time* (by default set to 1 second).

5.7 Problems encountered

During the development of the eye movement HCI, a number of issues were encountered that merited solutions. These issues are inherent in all HCIs based on pure eye tracking and can be partly alleviated using multimodal approaches that combine the benefits of each method.

In order of importance, the following four issues were encountered:

- The *Midas Touch* problem, which refers to the difficulty of distinguishing between intentional and unintentional actions from the user. If eye tracking is the sole input modality in use, every single eye movement can potentially trigger an action inside the virtual environment, resulting in an endless stream of actions that may not be desired by the user. The typical solution to this issue is the use of longer dwell times for command triggering. However, longer dwell times require higher effort on the part of the user, which can lead to unnatural, fatiguing interactions (Ware and Mikaelian, 1986; Ashmore et al., 2005).

  In this project, the problem is mitigated in three ways:

    1. first, there is a clear distinction between the triggers of the three actions (turn left / right, move forward), requiring deliberate action by the user;
    2. second, the user will typically wish to keep moving forward until an obstacle is encountered (i.e. move forward is the most common action and it requires minimal effort on the part of the user. Effort is only required when this action is no longer an option, which minimizes the chance of unintentional activation);
    3. third, a dwell time of 1 second is used, which is long enough to avoid unintentional activation, but short enough to feel natural.

- Calibration loss, which occurs when the user moves his head significantly from the calibrated position. Head movements overwhelm the voltage potentials of eye movements and throw off the calibration of the system. Given enough deviation, a head movement can alter the calculated offsets to the point where a fixation on the center of the screen triggers a turn left or turn right actions, leading to an endless spinning state or death spiral.

  Two methods were implemented to mitigate this issue:

    1. An automatic *sliding window* recalibration, which monitors incoming voltages and tries to compensate for deviations from the original calibration. If a voltage outside the calibrated limits is encountered, the limits are adjusted to maintain offset symmetry (from -1.0 to +1.0).
    2. If a death spiral is detected, movement is halted and a new calibration is required before resuming.

- *Input latency*, which represents the time difference between an eye movement being performed by the user and the result of that movement being reflected in the virtual environment (MacKenzie and Ware, 1993; Ware and Balakrishnan, 1994). Input latency is contributed by the flow of data from the EEG device to ActiView for initial processing, the transmission from ActiView to the input interface for further processing, the final transmission over the network (loopback or local), and the rendering of virtual environment.

The total amount of latency was measured to lie in the region[7] of 300-500 ms. The rendering of the virtual environment on the 1-wall CAVE required 8.3 ms for each frame (120 Hz), while the network transfer of the processed input from the input interface to the VR platform required less than 1 ms (measured using high-resolution timestamps). This leaves the transfer from the EEG to ActiView and the processing in ActiView as the largest contributing factors

In either case, the impact of this latency on the implemented HCI was far from debilitating. Indeed, the difference between mouse input (which did not suffer from this latency) and EOG was visible as a slight increase in the dwell time required for the triggering of an action (1.0 seconds for mouse input, 1.3-1.5 seconds for EOG). Things would have been very different if eye movements were used to control the orientation of the user *directly,* where input latency would create a noticeable and uncomfortable disconnection between user intent and environment reaction. The fact that an *indirect* mapping was used (i.e. look right for 1 second to turn 90° right) all but eliminated this issue.

- *Fixation jitter*. Unlike a mouse or a keyboard, eyes do not stop moving during fixation. Small involuntary movements, called microsaccades, still occur, with amplitudes between 2 and 120 arc minutes (Zuber et al., 1964). Microsaccades cause measurable jitter on the recorded voltage potential and may hinder user input based on a dwell time approach (Ashmore et al., 2005).

  Jitter was mitigated by avoiding small active elements in the user interface: eye movements were directed to three large regions on the screen (left, center and right), which were significantly larger than the amplitude of fixation jitter. The application of a smoothing filter, as suggested by Ashmore, was not deemed necessary.

[7] An exact measurement would require knowledge first of the exact start of the eye movement, then of the exact display of that movement on the screen. This is typically achieved either by using an additional trigger signal that records the exact timestamp of the eye movement event, or using one or more synchronized cameras that record both the user and the monitor.

Chapter 6: Conclusions and Future Directions

In the context of the present thesis a 3d virtual environment was constructed using state of the art graphics algorithms. Three HCI modalities, keyboard, mouse and eye movements were implemented for navigation, with the ability for expansion to implement additional methods in the future.

A labyrinth was chosen as the layout of the environment. The reason for that is that labyrinths are easy to explain, allow for meaningful user interactions and, equally importantly, can be navigated using a single axis of freedom, making them particularly suitable for EOG-based navigation. Moreover, such a layout can be used to test memory and spatial orientation in clinical populations with deficiencies.

The virtual environment utilizes the power of modern GPU hardware to create an attractive visual appearance, which includes dynamic light and shadow calculations, water reflections and refractions and ambient occlusion effects. The image is displayed in stereographic 3d on compatible hardware (3d monitors with shutter glasses, or video projection / CAVE with polarized glasses). Recent improvements in graphics algorithms, such as dynamic indirect lighting, can be incorporated to further improve the visual appearance of the environment (Crassin et al., 2011). Moreover, a fully color-managed graphics pipeline, based on the sRGB model, should be used, in order to ensure correct visual appearance on different monitors.

Even though the developed platform has only been tested with conventional input methods, like the keyboard and EOG, it is readily expandable. In fact, a major future direction of research would be to examine optimal ways to navigate this environment via mental commands. For this, we could profit from recent advances in the field of BCI: signal acquisition and processing techniques allow thoughts and mental commands to be translated to machine commands (Wolpaw et al., 2002). For example, one could imagine input coming from an EEG machine to be translated to actions in the VR environment. Eventually, more advanced scenarios with increasing degrees of complexity and interest for the user could be implemented. For this purpose, the current input methods (i.e. EOG) could be used for navigation and mental commands, recorded with EEG could be used for interaction (for example for grasping or exploring objects).

This type of hybrid platform, currently inspired by greek mythology, could be used for educational purposes, for example for rendering history lessons to young pupils more interactive. It could also be used by children with learning disabilities, as an alternative to more classical teaching methods. Other applications involve clinical populations, such as patients with psychiatric disorders, who could for example profit from a virtual environment for overcoming phobias in the real world. Finally, patients with motor disabilities, who have limited control over their body muscles (for example after a stroke, or patients with ALS), could use this platform for rehabilitation purposes. Such applications should of course be implemented with the collaboration of psychologists and clinicians who are aware of patients' special needs.

From a more technical point of view, the correlation between labyrinth size and user performance should be studied, in order to determine an optimal labyrinth size. It should be studied whether the requirement for continuous calibration and an immobile head could be lifted through improved head motion tracking and compensation, potentially using additional input signals alongside EOG. For EOG itself, vertical motions of the eye could also be added in order to allow additional axes of freedom and enrich the user experience. Finally, an optimal dwell time should be identified, which would both avoid the Midas Touch issue and increase user immersion.

Chapter 7: References

Abibullaev B, An J. 2012. Classification of frontal cortex haemodynamic responses during cognitive tasks using wavelet transforms and machine learning algorithms.Med Eng Phys. in press

Akenine-Möller T, Haines E, and Hoffman N. 2002. Real-time rendering. AK Peters.

Allison BZ, Leeb R, Brunner C, Müller-Putz GR, Bauernfeind G, Kelly JW, Neuper C. 2012. Toward smarter BCIs: extending BCIs through hybridization and intelligent control. J Neural Eng. 2012 Feb;9(1):013001.

Andersson P, Pluim JP, Viergever MA, Ramsey NF. 2012 Navigation of a Telepresence Robot via Covert Visuospatial Attention and Real-Time fMRI.Brain Topogr. 2012 Sep 11. in press.

Belitski A, Farquhar J, Desain P. 2011. P300 audio-visual speller. J Neural Eng. 2011 Apr;8(2):025022.

Cabeleira J, Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time, Instituto Superior Técnico.

Chalmers A, Davis T, Reinhard E. 2002. Practical parallel rendering, ISBN 1-56881-179-9. AK Peters, Ltd.,

Cheron G, Duvinage M, De Saedeleer C, Castermans T, Bengoetxea A, Petieau M, Seetharaman K, Hoellinger T, Dan B, Dutoit T, Sylos Labini F, Lacquaniti F, Ivanenko Y. 2012. From spinal central pattern generators to cortical network: integrated BCI for walking rehabilitation.Neural Plast. 2012;2012:375148.

Dias NS, Kamrunnahar M, Mendes PM, Schiff SJ, Correia JH. 2010. Feature selection on movement imagery discrimination and attention detection. Med Biol Eng Comput. 2010 Apr;48(4):331-41.

Eggert T. 2007. Eye Movement Recordings: Methods. Dev Ophthalmol. Basel, Karger, vol 40, pp 15–34

Flamary R, Rakotomamonjy A. 2012. Decoding Finger Movements from ECoG Signals Using Switching Linear Models. Front Neurosci. 2012;6:29.

D. Friedman, R. Leeb, G. Pfurtscheller and M. Slater. 2010. Human-Computer Interface Issues in Controlling Virtual Reality With Brain-Computer Interface.Human-Computer Interaction, vol. 25, p. 67-94.

F. Galán, M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips and J. d. R. Millán. 2008. A Brain-Actuated Wheelchair: Asynchronous and Non-Invasive Brain-Computer Interfaces for Continuous Control of Robots.

in Clinical Neurophysiology, vol. 119, num. 9, p. 2159-2169.

Gans RE. 2001. Video-oculography: A new diagnostic technology for vestibular patients. Hearing Journal.54 (5) pp 40,42.

Goldstein, R. A., and Nagel R. 1971. 3-D visual simulation. Simulation 16(1), pp. 25–31.

Hashimoto Y, Ushiba J, Kimura A, Liu M, Tomita Y. 2010. Change in brain activity through virtual reality-based brain-machine communication in a chronic tetraplegic subject with muscular dystrophy. BMC Neurosci. 2010 Sep 16;11:117.

Higashi H, Tanaka T. 2011. Optimal design of a bank of spatio-temporal filters for EEG signal classification. Conf Proc IEEE Eng Med Biol Soc. 2011;2011:6100-3.

Hwang HJ, Lim JH, Jung YJ, Choi H, Lee SW, Im CH. 2012. Development of an SSVEP-based BCI spelling system adopting a QWERTY-style LED keyboard.J Neurosci Methods. 2012 Jun 30;208(1):59-65.

Iáñez E, Ùbeda A, Azorín JM. 2011. Multimodal human-machine interface based on a brain-computer interface and an electrooculography interface. Conf Proc IEEE Eng Med Biol Soc. 2011;2011:4572-5.

Iturrate I, Montesano L, Chavarriaga R, del R Millán J, Minguez J. 2011. Minimizing calibration time using inter-subject information of single-trial recognition of error potentials in brain-computer interfaces. Conf Proc IEEE Eng Med Biol Soc.2011:6369-72.

Javal, É (1878). "Essai sur la physiologie de la lecture". Annales d'Oculistique 80: 61–73.

Jeng-Weei Lin J, Duh HBL, Abi-Rached H, Parker DE, and Furness TA. 2002. Effects of Field of View on Presence, Enjoyment, Memory, and Simulator Sickness in a Virtual Environment. In Proceedings of the IEEE Virtual Reality Conference 2002 (VR '02). IEEE Computer Society, Washington, DC, USA, 164.

Jin J, Sellers EW, Wang X. 2012. Targeting an efficient target-to-target interval for P300 speller brain-computer interfaces.Med Biol Eng Comput. 2012 Mar;50(3):289-96.

Judd DB, Wyszecki G. 1975. Color in Business, Science and Industry. Wiley Series in Pure and Applied Optics (third ed.). New York: Wiley-Interscience. pp. 388. ISBN 0-471-45212-2.

Keith J. 2007. Video demystified: a handbook for the digital engineer (5th ed.). Newnes. p. 168. ISBN 978-0-7506-8395-1.

Kent C R, Smith TM. 1980. Project Whirlwind: The History of a Pioneer Computer p. 216.Bedford, MA: Digital Press. ISBN 0-932376-09-6.

Leeb R, Friedman D, Müller-Putz GR, Scherer R, Slater M, Pfurtscheller G. 2007. Self-paced (asynchronous) BCI control of a wheelchair in virtual environments: a case study with a tetraplegic.Comput Intell Neurosci. 2007:79642.

Long J, Li Y, Wang H, Yu T, Pan J, Li F. 2012. A hybrid brain computer interface to control the direction and speed of a simulated or real wheelchair. IEEE Trans Neural Syst Rehabil Eng. 2012 Sep;20(5):720-9.

Macey J.Ray-tracing and other Rendering Approaches, lecture notes, MSc Computer Animation and Visual Effects University of Bournemouth.

MacKenzie IS, Ware C. 1993. Lag as a determinant of human performance in interactive systems. In Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems (CHI '93). ACM, New York, NY, USA, 488-493.

Malmivuo J and Plonsey R 1995. Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields, Oxford University Press, New York.

Manyakov NV, Chumerin N, Combaz A, Van Hulle MM. 2011. Comparison of classification methods for P300 brain-computer interface on disabled subjects. Comput Intell Neurosci. 2011;2011:519868.

Marmor MF, Zrenner E. 1993. Standard for clinical electro-oculography. International Society for Clinical Electrophysiology of Vision. Arch Ophthalmol.111(5):601-4.

Marmor MF, Brigell MG, McCulloch DL, Westall CA, Bach M; International Society for Clinical Electrophysiology of Vision. 2011. ISCEV standard for clinical electro-oculography (2010 update). Doc Ophthalmol. 122(1):1-7.

Marsh D, 2001 "Temporal Rate Conversion", Microsoft.

Nikodym T. 2010. "Ray Tracing Algorithm For Interactive Applications". Czech Technical University, FEE.

Ortner R, Aloise F, Prückl R, Schettini F, Putz V, Scharinger J, Opisso E, Costa U, Guger C. 2011. Accuracy of a P300 speller for people with motor impairments: a comparison.Clin EEG Neurosci. 2011 Oct;42(4):214-8.

X. Perrin, R. Chavarriaga, F. Colas, R. Siegwart and J. d. R. Millán.2010 Brain-coupled Interaction for Semi-autonomous Navigation of an Assistive Robot in Robotics and Autonomous Systems, vol. 58, num. 12, p. 1246-1255, 2010.

Pierrot-Deseilligny JC, Muri RM, Ploner. Gaymard B, Rivaud-Pechoux S. 2003. Cortical control of ocular saccades in humans: a model for mortricity. Progress in Brain Research, Vol. 142.

Pires G, Castelo-Branco M, Nunes U. 2008. Visual P300-based BCI to steer a wheelchair: a Bayesian approach.Conf Proc IEEE Eng Med Biol Soc.2008:658-6

Porter T, Duff T. 1984. Compositing Digital Images. Computer Graphics 18 (3): 253–259.

Punsawad Y, Wongsawat Y, Parnichkun M. 2010. Hybrid EEG-EOG brain-computer interface system for practical machine control. Conf Proc IEEE Eng Med Biol Soc.2010:1360-3.

Rege A, 2008. "An Introduction to Modern GPU Architecture", p. 4-13

Renaud P, Joyal C, Stoleru S, Goyette M, Weiskopf N, Birbaumer N. 2011. Real-time functional magnetic imaging-brain-computer interface and virtual reality promising tools for the treatment of pedophilia. Prog Brain Res. 2011;192:263-72.

Ron-Angevin R, Velasco-Alvarez F, Sancha-Ros S, da Silva-Sauer L. 2012. A two-class self-paced BCI to control a robot in four directions.IEEE Int Conf Rehabil Robot. 2011;2011:5975486.

Saa JF, Çetin M. 2012. A latent discriminative model-based approach for classification of imaginary motor tasks from EEG data.J Neural Eng. 2012 Apr;9(2):026020.

Schalk G, Leuthardt EC. 2011. Brain-computer interfaces using electrocorticographic signals.IEEE Rev Biomed Eng. 2011

Sorger B, Reithler J, Dahmen B, Goebel R. 2012. A real-time fMRI-based spelling device immediately enabling robust motor-independent communication. Curr Biol. 2012 Jul 24;22(14):1333-8.

Stokes M, Anderson M, Chandrasekar S, Motta R. 1996. A Standard Default Color Space for the Internet – sRGB, Version 1.10.

Swaine M.1983. New Chip from Intel Gives High-Quality Displays, p.16

Thongpang S, Richner TJ, Brodnick SK, Schendel A, Kim J, Wilson JA, Hippensteel J, Krugner-Higby L, Moran D, Ahmed AS, Neimann D, Sillay K, Williams JC. 2011. A micro-electrocorticography platform and deployment strategies for chronic BCI applications.Clin EEG Neurosci.42(4):259-65.

Tomioka R, Müller KR. 2010. A regularized discriminative framework for EEG analysis with application to brain-computer interface. Neuroimage. 49(1):415-32.

Trad D, Al-ani T, Monacelli E, Jemni M. 2011. Nonlinear and nonstationary framework for feature extraction and classification of motor imagery.IEEE Int Conf Rehabil Robot. 2011;2011:5975488.

Tsui CS, Gan JQ, Hu H. 2012. A self-paced motor imagery based brain-computer interface for robotic wheelchair control. Clin EEG Neurosci. 242(4):225-9.

Tzovara A, Murray MM, Bourdaud N, Chavarriaga R, Millán Jdel R, De Lucia M. 2012. The timing of exploratory decision-making revealed by single-trial topographic EEGanalyses. Neuroimage. 60(4):1959-69.

Usakli AB, Gurkan S, Aloise F, Vecchiato G, Babiloni F. 2009. A hybrid platform based on EOG and EEG signals to restore communication for patients afflicted with progressive motor neuron diseases. Conf Proc IEEE Eng Med Biol Soc. 2009;2009:543-6.

Usakli AB, Gurkan S, Aloise F, Vecchiato G, Babiloni F. 2010. On the use of electrooculogram for efficient human computer interfaces.Comput Intell Neurosci. 2010:135629.

Ware C and Balakrishnan R. 1994. Reaching for objects in VR displays: lag and frame rate. ACM Trans. Comput.-Hum. Interact. 1, 4, 331-356.

Weiskopf N. 2012. Real-time fMRI and its application to neurofeedback. Neuroimage. 2012 Aug 15;62(2):682-92.

Whitted T. 1979. An improved illumination model for shaded display. Proceedings of the 6th annual conference on Computer graphics and interactive techniques

Wolpaw JR, Birbaumer N, McFarland DJ, Pfurtscheller G, Vaughan TM. 2002. Brain-computer interfaces for communication and control. Clin Neurophysiol.113(6):767-91.

Yang J, Singh H, Hines EL, Schlaghecken F, Iliescu DD, Leeson MS, Stocks NG. 2012.Channel selection and classification of electroencephalogram signals: an artificial neural network and genetic algorithm-based approach. Artif Intell Med. 2012 Jun;55(2):117-26.

Yu T, Li Y, Long J, Gu Z. 2012. Surfing the internet with a BCI mouse.J Neural Eng. 2012 Jun;9(3):036012.

Zhu H, Sun Y, Zeng J, Sun H. 2011. Mirror neural training induced by virtual reality in brain-computer interfaces may provide a promising approach for the autism therapy.Med Hypotheses.76(5):646-7.

Zuber BL, Crider A, Stark L. 1964. Saccadic suppression associated with microsaccades. Quarterly Progress Report, 7:244–9.

Ashmore M, Duchowski A T, Shoemaker G. 2005. Efficient eye pointing with a fisheye lens. In Proceedings of Graphics Interface 2005 (GI '05). Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 203-210.

Ware C, Mikaelian HH. 1986. An evaluation of an eye tracker as a device for computer input. SIGCHI Bull. 18, 4 (May 1986), 183-188.

Crassin C, Neyret F, Sainz M, Green S, Eisemann E. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing: An Insight. 2011 Aug; "Siggraph 2011 Talk".

Press WH, Teukolsky SA, Vetterling WT, Flannery BP. 2007.Section 17.4. Second-Order Conservative Equations. Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York: Cambridge University Press.

Phong BT. 1975. Illumination for computer generated pictures. Commun. ACM 18, 6 (June 1975), 311-317.

Greene N. 1986. Environment Mapping and Other Applications of World Projections. Computer Graphics and Applications, IEEE (Nov 1986), vol.6, no.11, pp.21-29.

Emst I, Jackel D, Riisseler H, Wittig O. 1995. Hardware Supported Bump Mapping: A Step towards Ingber Quality Real-Time Rendering. Tenth Eurographics Workshop on Graphics Hardware, pp. 63-70.

Tessendorf J. 2001. Simulating Ocean Water [online].

Tatarchuk N. 2006. Dynamic parallax occlusion mapping with approximate soft shadows. In Proceedings of the 2006 symposium on Interactive 3D graphics and games (I3D '06). ACM, New York, NY, USA, 63-69.

Donnelly W and Lauritzen A. 2006. Variance shadow maps. In Proceedings of the 2006 symposium on Interactive 3D graphics and games (I3D '06). ACM, New York, NY, USA, 161-165.

Loupos K, Psonis P, Amditis A. 2008. Virtual Reality: The Way Ahead In Industrial Safety. 22nd European Conference on Modelling and Simulation (ECMS 2008).

Fortin M. 2011. Automated Maze Generation and Human Interaction [online]. Diplomová práce. Masarykova univerzita, Fakulta informatiky.

C# Language Specification (4th ed.). Ecma International. June 2006.

[Websites]

http://www.opentk.com

http://arstechnica.com/gaming/2008/09/gpu-sweeney-interview/

http://venturebeat.com/2012/02/09/epics-tim-sweeney-predicts-the-next-20-years-in-gaming-technology/

http://msdn.microsoft.com/en-us/windows/hardware/gg463407.aspx