Vision-Based Human Interaction Devices in a 3D environment

Using Nintendo Wiimote, Webcam and DirectX

Thomas Luel
Florent Mazzone

Vision, Graphics and Interactive Systems

Aalborg University

June 1, 2009

AALBORG UNIVERSITY
DEPARTMENT OF ELECTRONIC SYSTEMS

Niels Jernes Vej 12 – DK-9220 Aalborg East Phone 99 40 86 86

TITLE: Vision-Based Human Interaction Devices in a 3D Environment

PROJECT PERIOD: 10th Semester, February 2009 to June 2009

PROJECT GROUP: 1021

PARTICIPANTS:

THOMAS LUEL

FLORENT MAZZONE

SUPERVISOR:
ZHENG-HUA TAN

PUBLICATIONS:	3
NUMBER OF PAGES:	99
FINISHED:	JUNE 1, 2009

ABSTRACT

This report documents the development of new Human Interface Devices (HID) and their use in a 3D environment to enhance the immersion factor of a modeling tool such as an architecture designer software. The study focuses on studying and comparing two common devices used in vision-based motion tracking: the wiimote developed by Nintendo and a webcam. New means of interaction between the user and software, going beyond the mouse and keyboard bounds, are explored and described to interact efficiently and intuitively with desktop objects.

In this system, the user's fingers and head are infrared lighted by adapted gloves and frames fit with infrared diodes. The wiimote, developed by Nintendo, provides a high accuracy camera sensor whereas the webcam has a good detection range. Both are adapted answers to the application requirements. A 3D DirectX architect software, Master Builder, has been implemented, allowing the user to create his own buildings using his fingers and to view the result in 3D by moving his head.

The resulting projects and the users tests revealed that infrared vision-based tracking fits perfectly to the tracking requirements and that handling objects by using both hands is much more interactive for the user than using common human device interfaces such as the mouse.

Preface

Report purpose

This report is part of the 10th semester project of the "Vision, Graphics and Interactive Systems" Master program. It has been written in 2009 at the Aalborg University Department of Electronic Systems and presents the Master thesis of the group VGIS 09gr1021.

The project aim is to design and implement new means of interaction such as fingers and head tracking between the user and a 3D architecture modeling software, in order to improve the ease of use of this type of application. The overall project has been documented in this report, which describes the design choices, implementation, testing and improvements of the solution.

Report overview

Seven main sections compose this report. The Pre-Analysis describes common hardware and software technologies used to improve the user's immersion in an application and to track his motions. The Analysis part defines more precisely the technologies used in the project and presents the specification of requirements for the ideal system. The Design section describes the different modules developed for this project. A first sub-section presents the hardware components developed for the fingers and head tracking. The tracking software is then described in a second part. Finally, the development of the three dimensions game and of the user interface using DirectX9 is detailed. The Implementation part describes the development of the different modules designed, the problems encountered during this phase and some modifications of the initial design. The Tests part describes how the usability tests have been made to validate the application design choices and the results obtained. The Tests section then presents all the modifications of the application made to enhance the interaction between the user and the software and to correct some bugs discovered during the test period. Finally, the Conclusion will explain the project achievement and the improvement perspectives.

A list of the figures included is available; each figure caption is preceded by his chapter number separated by a dot from its order number in the section. The appendix contains the references and technical documents. A number in parenthesis marks each reference quoted.

Technology notice

The CD-Rom includes the binaries with documented source codes of MasterBuilder.exe, the 3D environment, and of MotionTracking.dll, the motion-tracking library. The CD-ROM also contains a

MotionTracking.dll sample application, the webcam driver, a tool to handle the camera properties, a documentation of the MotionTracking.dll external methods (.HTM and Microsoft Compressed HTML) detailed installation instructions and the electronic components datasheets.

The source code has been developed in C# using .Net framework 2 using Microsoft Visual Studio 2005 development environment. The report has been written with Microsoft Office 2007, the schematic and diagrams have been drawn with Microsoft Visio 2003 and the quality of the screen or hardware pictures has been improved using Paint.NET.

Acknowledgements

We would like to thank all the people who helped us to achieve our project, and more particularly:

- Ben Krøyer and Peter Boie Jensen, Electronic Systems Department assistant engineers, for their help and good advice during the electronic components development.
- Lars Bo Larsen, our semester coordinator for his support.
- Mette Billeskov, semester secretary, for her support regarding the administrative issues.
- Our supervisor Zeng-Hua Tan, our project supervisor, for his help, management and advice.
- All the testers who helped to check, validate and improve the design choices.
- The computer sciences workshop for their help regarding the computers and network issues.
- The components workshop that helped us to order all the needed components.

Table of contents

List of figures

Part I Introduction

This chapter presents the origins of the project. The existing technologies, the needs, the problem statement and the delimitations of the project are described in the following parts. The final sub-section motivates the motions that led us to choose this project.

Presentation

Interactions between a user and personal computer software are a main issue for developers since the invention of the first personal computers in the seventies. The basic command-line interface provided by the first operating systems such as Microsoft MS-DOS revealed rapidly their lack of intuitiveness for the user. For that reason, graphical operating systems such as Mac OS, IBM OS/2 or Microsoft Windows were designed in the eighties, allowing an occasional user to interact efficiently with a computer. The graphical operating systems soon revealed the limits of the keyboard unable to handle intuitively graphical objects. The widespread use of the graphical user interface led human interfaces devices (HID) designers to introduce the mouse (invented by Douglas Engelbart in 1964 [1]) which is much more adapted to the needs. This HID was so successful that it hid the development of others interfaces devices such as fingers tracking devices. Nevertheless, it appeared recently that the situation changed, now that the mouse device shows its limits especially in some applications as it does not provide a human logic way of interacting. When the application needs to have a good immersion factor, motion tracking devices such as fingers or head tracking are much more adapted. They provide an easy and intuitive way for the user to handle objects or to move in a three dimensions environment. Moreover, this spares the users the need and time to learn the necessary handling techniques before using the software.

Motion tracking has been used for years in computer sciences since it can be implemented using numerous well-known technologies: accelerometers, inertial, magnetic or vision-based tracking. However, these technologies were so expensive to implement that the use of motion tracking was limited during decades to major companies or laboratories. Fortunately, the improvement of the computer processing power and, above all, decreasing prices of accurate sensors enabled user interaction designers to implement motion tracking in generalized applications such as games or immersion applications. Many different cheap devices such as the wiimote (Nintendo: camera & accelerometer), Play Station 3 (Sony: camera [2]) or other specific cameras have been brought with success to the mass market for dedicated applications. To date the commercialized programs focus on tracking the head, the hand or the fingers position but could be easily modified to track other features and be used in others applications.

To date numerous cheap motion-tracking solutions have been designed and commercialized. Many devices rely on the vision-based tracking such as the wiimote implemented by Nintendo. However, it seems that most of them are used for gaming purpose and that their capabilities are not often used in others software. Moreover, no comparison of the tracking capabilities of these devices has been made. This lack of information can be disturbing for the developer improving the immersion of software using these tools.

Problem statement

The purpose of this project is to compare and implement motion tracking using two generalized devices, the wiimote and a webcam, in order to demonstrate the advantages of vision-based tracking in an architecture modeling software.

The wiimote and webcam capabilities for tracking will be compared. The advantages and drawbacks of each solution will be presented to allow developers to use the best components in their project. Moreover, a new architecture modeling software in three dimensions will be implemented to show the intuitiveness and efficiency of fingers tracking for this type of application and the need of head tracking to visualize in details a scene.

Fundamental requirements

- Track simultaneously at least two fingers and the head position of the user.
- Provide a good accuracy to execute all the application features at distances of 2 to 4 meters from the screen.
- Allow the user to run the application without using a keyboard and a mouse.
- Present to the user a three dimensions environment.
- Perform a user testing with neophyte testers to validate the intuitiveness and efficiency of the fingers object handling.

Limitations

- The solution language is English.
- The number of available different objects in the architecting software is limited as the aim of the project is to experiment new means of objects handling and not to provide a fully functional architecting software.
- The vision-based tracking used is limited to a room with controlled light such as a living room or classroom.

Part I Pre-Analysis

This part describes the main concepts of the project. A study of the importance of the immersion in games and other kind of software is done, and then an insight of motion tracking techniques will be presented.

1

“Immersion” factor in software and games

There are many factors, which make games interesting to the player: purpose of the game, interesting story line, etc. In some games, the purpose is to draw the player into another world, to make him “believe” that the game is a world itself, with its own rules and own goals. The factor responsible for this is called “Immersion”.

1.1 How could we define Immersion?

Immersion is defined as the “suspension of disbelief” [3]. To achieve this state, several criteria that will enhance the “reality” of the game have to be achieved. A game has a good “immersion factor” if the graphics are good, the movements of the characters are realistic, the physics effects (such as an object that would fall on the ground) match the reality (or at least the reality of the game), and other criteria as well [4]. When those criteria are met, the player can focus more easily on the goal of the game, and the gaming experience is much better. We will now list the most important criteria [5].

- **Good background and foreground graphics** will make the game looks beautiful and, more important, realistic. The user will easily identify the elements of the game, recognizing them by associating them with elements of the real world.
- **Realistic graphic effects** are also important. Small details can make the game more realistic: movement of the dust, interaction of the weather (rain, thunder, snow).
- **Sound and music** play also an important part in the immersion. They allow the user to “feel” the game world with something more than just vision. The music can create the atmosphere or the feeling that the user should experience. The more detailed is the sound, the better the immersion of the game is.
- **The means of interaction** with the game are also a very important point. For example, a joystick is better than a mouse for a flight simulator, because it seems nearer to the real commands of a plane, and more intuitive.

When those criteria are met, the immersion into the game is easier, and the game experience is better.

1.2 Why is immersion so important in games?

When a game has a good immersion factor, the player feels drawn into the game world, and the actions performed in the game seem real [6]. The illusion created is strong and the user will have a

stronger experience. With a good immersion, it is easier for the player to interact with the elements of the game as he identifies them more easily and can use his intuition to interact with them. That is why, since the beginning of video games, developers have tried to improve all those areas described above, pushing the limits further and further.

1.3 What about other kinds of software?

Other kinds of software can also profit from a good immersion. Software application where the user needs to look at something, or needs to get the feeling of something, can be used more efficiently if it creates a good illusion of reality.

Applications that simulate an environment would be part of this category of software. If the application is immersive, the user will identify objects and know their characteristics with more accuracy. Such applications could be any kind of software helping the user to design an object or a set of objects. Those applications belong to the “computer aided design” type of software [7].

1.4 What are the new ways to improve the immersion factor?

They are many ways to improve the immersion in a game or in software. In this report, we will give only a few examples [8][9].

- The Wii console: the main idea of this console is to associate real movements of the player to movements or actions in the game. For example, a tennis game will associate movements of the Wii remote to movements of the racket. The player will find the game nearer to a real tennis match than if it was played with a keyboard. This creates a kind of immersion.

Figure I-1: Picture showing how to use the wiimote[10].

- 3D sound: by using special sound effect, 3D sound gives the user the feeling that he can locate the place where the sound comes from [11].
- 3D glasses: There are several ways to use 3D glasses. Most of the time, two images are projected on the same screen, and the 3D glasses force the user to see only one image at a time, switching very quickly between the two images. This creates the impression of 3D.

- 3D headsets: They must be worn on the head. They create images right in front of the user’s eyes.

Figure I-2: Vuzix 3D headset[12].

- Simulation equipment: mainly used by armies and companies, it is a set of tools simulating a flight for example [13]. All the commands normally available in the plane or the helicopter are also available in the simulation equipment. The user has a good impression of immersion in this case.

Figure I-3: ELITE Simulation solutions: the all-in-one simulator [14].

- 3D cave: this is a big cube-shaped box. To play a game, the user must go into the box where images are projected on every face of the cube. In addition, the user can wear 3D glasses, in order to see the images as if it was real 3D. This kind of tools gives a very impressing experience of immersion, because the user really feels like he is inside another world. He can also move a bit inside the box, and this freedom in his movements adds to the immersion. Of course, this is not a product available for mass market mainly because of its cost and of the space needed to use it.

Of course, there are other ways to enhance the immersion feeling of a game. It would be too long to describe them all. What is important to note is that this kind of tools are becoming more and more

used, and cheaper and cheaper. The success of the Wii console is a proof itself. People want the games to match reality, to simulate it. This need for immersion also begins to appear for some software as well.

2

Motion tracking

2.1 What is motion tracking ?[15]

Motion tracking consists of set of technologies developed to detect and track objects movements throughout a defined period.

This technique is used in wide range of products and more particularly in:

- Surveillance purposes such as human aggressive motion detection [16], human gestures detection in a specified environment, robots motion checking in factories for example.
- Virtual reality systems to observe and adapt the virtual environment “on the fly”.

2.2 How to track motion ?

The type of motion tracking system has to be chosen depending on the application developed, the types of motion to handle, the tracking quality, the processing power and the period between each motion detection. We will present in the following part the two main parts of a motion detection system:

Hardware: It usually consists in one or more sensors such as cameras, a gyroscope or pressure detector that monitor periodically a specific characteristic of an environment such as light, noise, pressure and that send the data retrieved to a dedicated software to analyze the motion and its evolution in the time. Depending on the needs, the system may implement multiple sensors to enhance the detection accuracy or the numbers of motion types tracked.

Software: Using the information retrieved by the hardware sensors such as a picture raw data, 3D acceleration signals, the software first improves the data by removing the noise, analyzes them and describes the motion (type, speed, shape…). The most advanced systems are able to recognize a specific part of an object and to follow its motion in the time (used in head or fingers detection projects for example). Amongst the software based motion detection system, **computational vision** has a huge potential. Although it requires huge power processing and that visual cues are complex to analyze [17], it has been in constant development since the 70’s because it enables to remove or reduce the equipments installed on the object to track. Most of the system developed use comparison between a signal and a model to extract the features of a picture such as contours, known shapes stored in a database or a human face. After having detected the objects to track, the computational

vision system has to follow their motions on the next frames. Numerous ways of processing can be chosen.

We will present in the following part some of the most common ones [17] :

- **Brightness detection**: In case the background and the object to track have different brightness, the new position of the object can be easily detected using pixel comparison. This method is inaccurate in complex background frames.
- **Background detection:** In case the camera is fixed during the motion tracking, the position of the object can be detected with pixel subtraction processing the difference of a background picture stored in the system (with no object to track) and of the frames to analyze.
- **Optical flow detection**: The best way to check the position of objects in a picture is to process the optical flow between each frame. If a threshold value of pixels having a flow value is exceeded then the system has to search the region of theses pixels and can determine the region of the moving object. In order to track and identify two objects overlapping on a frame, the system has to determine the depth of each pixel using stereo vision (a pair of cameras). Most advanced systems use a combination of the different methods presented and predict the next motions to reduce the regions to analyze.

We will use in this project a combination of a webcam and a wiimote to follow the fingers and head motion of the user. We know that the system developed will be usually deployed in an indoor room such as a living room with controlled light and that there will be only one user. For this reason, we will be have the opportunity to use any of the previously presented algorithms to follow the head and fingers motions of the user. To limit the background brightness overlapping with the user we will develop a solution based on artificial infrared user lighting and detection for the head and fingers tracking.

The main ideas have been described in this part, and the Analysis part will define in details the technologies used and the main choices of the project.

Part II Analysis

In this part, all the technical fields related to our project are analyzed. The two motion tracking techniques used in the project, head tracking and fingers tracking, are presented. The main choices concerning the installation layout and the software development are described, in order to define the summary of requirements.

1

Head Tracking

Amongst the application scope of human motion detection, we can distinguish head tracking. We will present this technology in the following section.

1.1 What is head tracking?

"Head tracking" includes all the methods used to detect the head position and its related motions. Depending on the needs, systems may track the user's head using only two dimensions and neglecting the depth value (if the user is sitting or standing at a constant depth from the detection system), or three dimensions (if the users is walking during the tracking).

1.2 What for?

Head tracking has numerous uses in virtual reality:

- Adjust the camera viewpoint depending on the position of the head to a screen (used in flight, driving simulations).
- Track the localization of a human being on a large surface.
- Reduce the computing power needed for other motion detection. (Knowing an exact head position reduces the amount of works for an eye detection system for example) [18].
- Head recognition / pattern matching

1.3 How to do it?

Before the widespread use of electronic systems, motion detection was made using mechanical tools and equipments such as wires. Using such complex equipments, only experiments in laboratory were possible and the wires restricted the user's movements.
By enabling the development of light, sober and smart motion detection, the globalization of electronic systems and their huge processing revolutionized head tracking technologies:
Depending on the needs, we can differentiate two main types of technologies [19]:

- **Passive head tracking system**: a **hardware part** that consists of one or more detectors retrieving periodically data describing an environment and a **processing part** analyzing these data to find the head position. It also follows the head of the person moving in the specified volume (Face recognition provided by OpenCV [20] software for example).

Infrared or visible spectrum cameras and more rarely ultrasonic sensors are used to retrieve the environment data because they are harmless for the user, simple and easy to link within a system (S-Video, FireWire, USB interface).
This kind of system requires a lot of processing power to retrieve accurate information about the head motions especially in big volumes. However, we can hide the system to the user tracked. For this reason, it will be more used for security matters (house safe alarm…), to track users that do not move a lot (face detection when the user is sitting at a fixed distance from a screen for example).

- **Active head tracking system:** similar to the passive system except that it includes an **emitter** system carried by the user (infrared diodes, specific shapes lights…) enabling the **detection system** to find almost instantly an accurate localization of the user in a bigger volume without using a lot of **processing power**. This system is very accurate but the user needs to be "aware" of the tracking and to wear a specific device. This type of tracking system is the best solution to restrict the use of "traditional" HID [1] and to offer new degrees of freedom to a user. For this reason, this type of tracking system has numerous applications in games [21], medicine [22], etc.

1.4 Infrared glasses

In order to test the infrared camera embedded in the wiimote, we suggest developing a head tracking software. This device is able to process up to four artificial infrared sources. For this reason, we decided to analyze the different kinds of infrared emitters, the most adapted to this receiver and the different ways to fix it to provide the best head tracking. Even if infrared sources tracking technology is widespread on the motion detection market and even if there are numerous documents describing the infrared receivers, we noticed that the documentation about the types of infrared sources used was inadequate. Infrared emitters manufacturers do not give any information about the frequency, power, and diodes they are using and the developers using wiimotes chose their diodes using empirical methods. For this reason, we will study in the following part the different infrared sources manufactured/adapted and the materials to get to build our own fully adapted solution for head tracking.

We noticed that all the solutions offered had a common design. Using a headset or a pair of frames, at least two infrared diodes are positioned at each side of the head. More accurate versions, with three infrared sources, add a diode on the top and center of the head allowing the receiver to triangulate the user localization. Most of the wiimote users chose to modify safety glasses built in with visible spectrum diodes at each side, removing the diodes and replacing them with infrared diodes.

[1] HID : Human Interface Design such as a mouse or a keyboard.

Figure II-1 : An example of infrared emitters glasses adapted from safety glasses [23].

In case the user is sitting near a camera with a built in infrared light, manufacturers produce a less complex version. Some infrared lights are fixed on the infrared camera and directed toward the user's head. Materials positioned at the glasses edges replace the infrared diodes by reflecting infrared light to the camera. The use of reflective materials reduces the weight of the device (no battery).

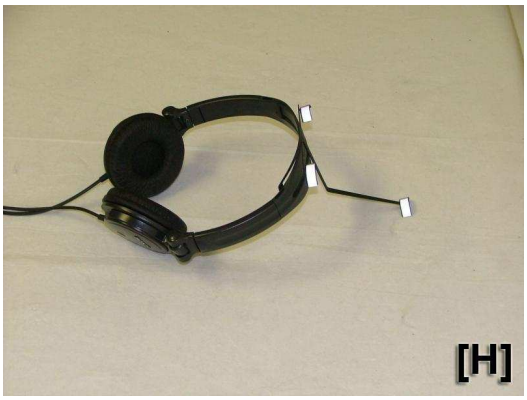

Figure II-2 : Glasses built with reflective materials made by TrackIr [24].

2

What is Fingers tracking?

"Fingers tracking" includes all the methods used to detect the fingers position and their related motions. As for the head tracking solutions, we may find some systems that give a three dimensions or two dimensions position of the fingers. We may find fingers tracking in many electronic devices such as the touch screens that use two dimensions tracking by neglecting the depth value.

2.1 What for?

Fingers' tracking is one of the oldest motion tracking solutions developed. As it was integrated in many electronic devices and has numerous purposes, we will present two important applications of fingers tracking:

- **Replacement of the usual interfaces devices**: The fingers tracking is in this case used to replace common "Human Interface Devices" (**HDI**) such as a mouse, a keyboard or a joystick in a user interface. Most of the time, the fingers tracking is only used as a complementary solution by improving the intuitiveness and ease of use of a user interface. For example, the touch screen installed in the **Tablet PC** offers the choice to use fingers tracking or a mouse for the same tasks.

- **Development of new interacting ways between a user and an interface**: Whereas the use of common HDIs limit the possible actions of the user (Mouse => 2D…), the fingers tracking allow developers to invent new "human" ways of user-computer communication. For example, we can integrate more than one cursor on the same screen using numerous fingers, work in 3D spaces or even track complex gestures of the user. The user is then able, by doing some specific gestures, to move a virtual object in a virtual world in which using HDIs such as a mouse, a keyboard would be almost impossible. For example, the tracking system of the augmented reality chess game [25] and of the Intelligent Touch Screen Tables implement only one interface device: the user's hands.

2.2 How to do it?

Tracking systems conceivers explored numerous different ways to detect and follow fingers motions. Depending on the complexity of the gestures to track, the system may track only the position of the fingers or describe the whole morphology of a hand.

- **2D fingers tracking:** This tracking follows only the width values and height values of the fingers positions on a leveled surface as the depth between the finger and the origin is constant. It is used in tactile screens where the user chooses the position to point by pressing the screens with one or more fingers.

- **3D fingers tracking:** The system retrieves a position of the fingers moving in a three dimensions space and may be able to reconstitute the hand morphology. To track the fingers position, the user usually wears a glove that implements a tracking hardware.

As the user must be able to look at the screen and walk at the same time, we have to consider tracking his fingers at any depth. For this reason, adapted techniques of fingers tracking are presented in the next subsections:

2.2.1 Inertial tracking [26]

This solution uses some inertial sensors such as accelerometers or gyroscopes to capture the variations of the fingers locations in the time. The sensors are usually embedded in the gloves and must be calibrated. Almost all the gestures can be captured and the solution is less independent to the environment than others are (no electromagnetic or light perturbations). However, it needs an interface to transmit the data (wire or wireless) to an acquisition system and its price is more expensive than a light based tracking solution.

Figure II-3 : Posture Acquisition Glove with inertial devices detecting linear and angular motion, rotation and acceleration of the hand [27]

2.2.2 Magnetic tracking [26]

This system is built using some coils on the emitter and the receiver to detect the magnetic flux between the fingers and process the position of the fingers. The tracking has a very good accuracy but is more expensive than the other solutions and is vulnerable to magnetic or electrical fields. The “cave” built by Aalborg university used this technology to track the fingers gestures.

2.2.3 Image tracking

This system uses a camera to film the fingers of the hand. The user has to wear a glove where specific images such as bar codes or shapes are fixed on each extremity of the fingers. The shot pictures are then processed. The tracking software first finds the position of the hand on the grabbed image, tries to match each figure image with its images library and processes the exact position of the fingers. This system has a good accuracy and is cheap. By adding some cameras on different angles, the software is able to retrieve “stereo” pictures of the scenery and gives a 3D position of fingers. However, this solutions works only in a controlled environment (with few infrared light emission) if the user is not too far from the camera.

Figure II-4 : Image based data glove [28]

2.2.4 Light tracking

This solution is quite similar to the image tracking system: a camera films the hand of the user and the user wears a glove where light sources such as diodes replace the images or shapes on the extremities of the fingers. The tracking software has to find the position of the lights on the grabbed pictures and process the exact position of the diodes with pairing methods. As visible spectrum light is too much dependant on the natural light of the room, infrared sources are used most of the time. This solution is able to track very complex gestures of the user as the “Inverse Kinematic Infrared Optical Finger Tracker” shown in Figure II-5. However, the hand of the user has to remain always in the field of view of the user. It is also quite cheap to develop.

Figure II-5 : “Inverse Kinematic Infrared Optical Finger Tracker” project [29]

In order to implement the fingers tracking in the project, we use the **light tracking** method. This solution is cheap and the excellent accuracy of the **magnetic tracking** system is not needed. As the objects displayed on the screen are quite big, the user can select them without problem with a pointer accuracy of ±3 pixels. Furthermore, the infrared light tracking methods is quite similar to the infrared head tracking explained in the previous part and we can share realization techniques and reduce the development length.

3

Installation layout

In order to show the immersion provided by the system, the user is allowed to move on a large surface (around 1.5 × 2 square meters). These surface parameters condition all the other settings of the installation:

- Minimal depth between the user and the screen.
- Minimal range of the infrared diodes.
- Size, resolution of the screen.
- Coordinates accuracy needed.

3.1 Minimal depth between the user and the screen

Knowing the field of view of the wiimotes (≈44 ° measured for the project wiimotes) and the range width (2m wanted, we use 2.5 meters for the calculation), we process the minimum depth between the user and the wiimote using the following formula :

$$\boxed{Depth_{min} = \frac{Width}{2 \times tan\left(\frac{FieldOfView}{2}\right)}}$$

$$Depth_{min} = \frac{2{,}5}{2 \times tan\left(\frac{44}{2}\right)}$$

$$Depth_{min} = \frac{2{,}5}{2 \times tan(22)}$$

$$\boldsymbol{Depth_{min} = 3{,}09\ \mathbf{m} \mp 5\%}$$

$$\boldsymbol{Depth_{\max} = 4{,}59\ \mathbf{m} \mp 5\%}$$

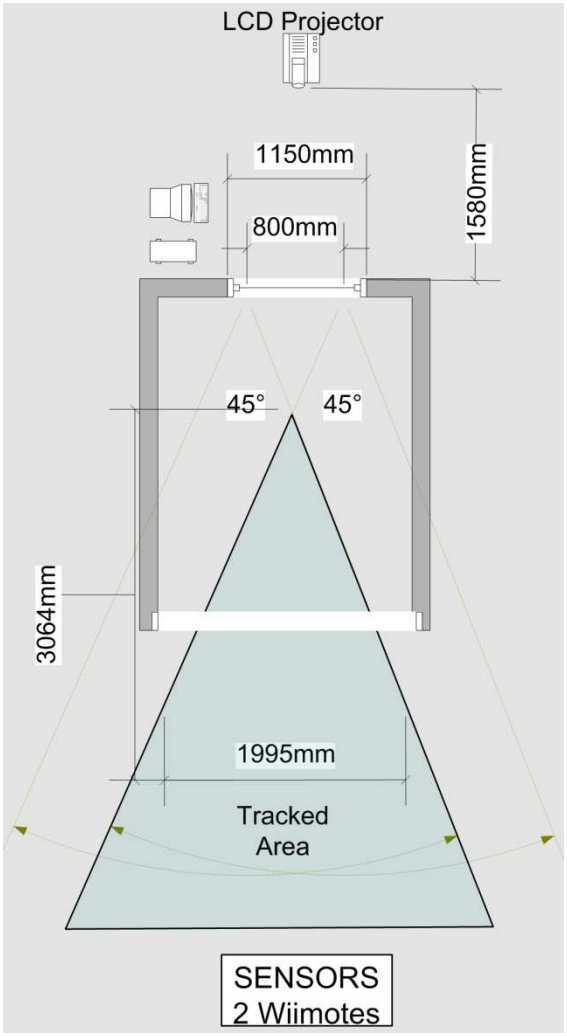

Figure II-6 : Wiimotes disposition layout (Head-tracking)

3.2 Minimal range of the infrared diodes

Knowing the maximum depth, it is decided that the user should carry infrared emitters that the wiimotes could detect and track at a 4.5 meters range at least.

3.3 Size, resolution of the screen

Some experiments made with a video projector using a 1024 × 768 resolution show that the diagonal of the screen should be at least 1.2 meters to allow the user to distinguish precisely the items on the screen.

3.4 Coordinates accuracy needed

As we have no existing solution to test the accuracy needed by the system, we fix it arbitrarily to ± 5% of the minimal depth (± 0,155 m). The tests made during the development of the project will indicate if more accuracy in the user localization is needed.

Once the requirements related to the installation layout are fixed, the design phase begins. A list of the requirements, resources available and possible answers is made:

3.5 Basic equipments needed

- A large screen diagonal: we have the choice between two different types of screen: a large TFT monitor fixed on a wall or a video projector to display the picture on a screen. The monitor is a very interesting solution (resolution larger than 1024*768, no space needed to project an image) but we finally choose to use a video projector. Although the resolution provided (1024 * 768) is inferior, it is a much more flexible solution because it allows to fix a specific screen size conveniently and can easily be moved in the optimal environment to avoid infrared perturbations due to the sun. Moreover, we have enough space in the laboratory to project an image.

- A large screen : to display the 3D environment (only if using a video projector)

- A computer equipped with a graphic card connected to the video projector that will execute the localization software and display the 3D environment on the screen. A Bluetooth interface to connect to the wiimotes to retrieve the user localizations and USB interfaces to connect to Webcams.

3.6 Project specific equipments needed

- Two wiimotes to retrieve the user's head position.
- One webcam to retrieve the user's fingers position.
- Frames with two infrared diodes: to track the head position using the wiimotes.
- One infrared source to light the user fingers.

4

Choice of the Application

The main idea of the project is to demonstrate the usefulness and the advantages of using motion tracking features, such as head-tracking or finger-tracking. But in order to make this demonstration, we needed a support to show those technologies in action. It was decided that we would develop an application which purpose would be not only to be compatible with the motion tracking features, but also to enhance their usefulness. We thus wanted the software program to force the user to make use of the features we implemented.

To determine which type of software would be developed, we had to think about the advantages of motion tracking, and determine the best way to enhance each of them. The particularities of the motion tracking features that we implemented are displayed below, and the impact in term of usefulness will be explained as well.

- Johnny Chung Lee uses head-tracking to give the impression that the user was looking through a window into "another world". When the user moves to the left, he will see what is on the right in the "world" behind the window. Coming near the screen will allow him to see more, and, at the contrary, going away from the screen will make him see less.
  - The idea of looking at a window could be used for driving or flying simulation. The screen could be seen as the view from the car or the plane.

Figure II-7: Johnny Chung Lee presenting his head-tracking system using wiimotes [30].

- Head-tracking enables the software to know the position of the user, or at least his head. That means it knows from where the user is looking at the screen. As a matter of fact, it can change the image displayed on the screen, in order to create the impression of 3D. When the user moves, the image displayed will still seem the same, but seen from a different angle, or a different distance. The user can move in the room, looking at the screen, and view the same object differently, depending on his position.

- This means that the program should be based on a 3D model. It could be a game or any kind of software that would use 3D modelling to serve its purpose. The application also has to make the user want to move to observe objects with a different angle. It means that observation should be an important part of the program. The user should want to look at 3D objects, and interact with them.

- Head-tracking also gives the system the approximate position of the user. When the user moves, the system knows it, and can use this information.

  - This could be used in a game. The user would have to avoid or catch some objects coming toward him, by moving in the real space (the room where he plays the game). This feature could be especially useful for a FPS game, where the user could try to avoid the enemy's fire. A game where objects come rushing toward the user could be a way to show this feature.

- TrackerIR [31] is a system using head tracking as a way to control a game. It is considered as a 6 Degrees of Freedom device. Depending on the game, a movement of the head will have a specific interaction. This is quite different from Johnny Chung Lee's program, since it does not give the impression of a real 3D world behind the screen. This is just another way to control part of a game, like a joystick, a mouse, a keyboard.

  - TrackerIR can be used in all game. However, for most games it is only used like a new way to control the cursor (instead of using the mouse). Some games were specifically designed for TrackerIR, and use all its potential. Those games are driving, flying, or FPS-like games. The movement of the head are mainly use to orient the view of the game.

Figure II-8: NaturalPoint's TrackIR3. [32]

- The finger tracking should allow the user to interact with the game by pointing one or several fingers to the screen. This allows him to quickly select an area or an object on the screen.

- This feature would become very user-friendly in applications where the user must select objects frequently. This could apply to many programs, and even to operating systems.

- Fingers tracking can also provide other kinds of interaction with the software. For example, by moving his finger and pointing to another area on the screen, the user could get an effect similar to a drag-and-drop with a mouse.

  - This kind of interaction could be useful for many types of games or professional software. But combined with the head-tracking, it gives the impression that the user can observe an object, and interact with it (possibly moving it, by example). This feature would be especially useful in a program which main purpose would be to design, modify, or analyse a 3D object. A game based on the same idea could also be interesting.

With all those arguments in mind, we made choices. Head tracking and fingers tracking can be used with many purposes, and in very different ways. That is why we first chose a way to use those technologies.

For us, the most impressive demonstration of head-tracking is Johnny Chung Lee's. The fact that the user can move in a large area makes the head tracking really useful for any game, where the user can really have a feeling that he is "inside" the game. On the contrary, the trackIR system appeared to us like just another input device, with no added value compared to a joystick or a mouse. That is why we chose to design a software program that would be a 3D environment, and which would adapt the view depending on the user's position.

On the other hand, the fingers tracking seemed a good way to select and move objects. We immediately thought about a program which purpose would be to design something. By pushing the idea further, we thought that fingers tracking could be very well adapted to architecture design.

Combining head tracking and fingers tracking lead us to the idea of an architecture or building program that would allow the user to choose objects and place them at a specific position.

5

Choice of the programming language

5.1 Analysis of the possibilities

To develop a 3D environment, there are different possibilities. First, a 3D programming API could be used. The most famous are OpenGL and DirectX. Those APIs are “near the hardware”, and interact directly with the drivers of the graphics card.

5.1.1 OpenGL [33]

Description of OpenGL:

Quote from Wikipedia: “*OpenGL (Open Graphics Library) is a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.”* OpenGL is used in many 3D applications, especially in fields like flight simulating, computer aided design or virtual reality. It is also used in some games.

OpenGL is called a specification, because its goal is to define some functions, and describe the actions they are associated with [34]. Hardware companies will then program the graphics cards, so that they can perform those actions. Since 2006, OpenGL is managed by Khronos Group. It was previously developed by Silicon Graphics Inc.

Purposes of OpenGL:

When using 3D graphics, one of the problems faced by the programmer is that a lot of things depend on the graphic card, or at least on the company which created the graphic card. One of the goals of OpenGL is thus to offer the programmer an interface that will be compatible with most graphics card. The developer does not need to worry about the type of graphic chipset used on the machine that runs his software. He writes his code using OpenGL library, and OpenGL communicates with the graphics chipset, as long as it is compatible.

On which systems can OpenGL programs be executed?

OpenGL is cross-platform, so it can be used on Windows, Linux, MacOS and any other operating system.

5.1.2 DirectX [35]

Description of DirectX

Quote from Wikipedia: "*Microsoft DirectX is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.*"

The 3D graphics API of DirectX is called Direct3D. It is used in many video games, but also for other kind of application, such as computer aided design applications.

The DirectX Software Development Kit (SDK) contains the libraries that the programmer will call in the code.

Components of DirectX

DirectX consists of many components, each of them having its own role and purpose [36][37].

- DirectX Graphics: DirectX graphics contains several APIs. DirectDraw is the API used for 2D drawings. It is now deprecated. Direct3D is the 3D drawings API. The last API is DXGI, which lists the graphic cards and the screens.
- DirectInput: to manage all the processing of the data coming from input devices, such as joystick, mouse, keyboards, etc. .
- DirectPlay: to use the computer network.
- DirectSound: to manage the sounds.
- DirectMusic: to play music.
- DirectX Media: to display 2D animation.
- DirectShow: to manage multimedia streams.
- DirectX Transform: to display web effects.

5.1.3 3D Engines [38][39][40]

3D Engines are a higher-level solution to develop 3D software. They are software built on a graphic API (like OpenGL or DirectX). They use OpenGL or DirectX functions to make their own functions, which are nearer to the goal they want to achieve. Each 3D Engine is thus specialized in a specific field of 3D programming. A lot of them are game engines, which means their purpose is to help the user to develop a game. They provide functions that are definitely oriented towards game programming. However, 3D engines are not only used for game designing, but also for 3D rendering, or any other kind of 3D programming.

It is much easier to program using a 3D engine, rather than OpenGL or DirectX, because most of the basic work is already implemented in the functions.

Some very famous examples of 3D engines:

- OGRE (Object-Oriented Graphics Rendering Engine): written in C, this is a scene-oriented engine [41].
- Crystal Space 3D: also written in C, this game engine is free and portable [42].
- Irrlicht: Usable for C and available in .NET languages, this engine is an open source project. It is specialized in real-time 3D. It also uses its own software renderer, in addition to OpenGL and DirectX [43].

- Panda3D: Specialized in 3D rendering and game development, this engine has a C library with Python bindings [44].

Those are only examples. They are a lot more 3D engines available.

5.2 Choice of the API

In this part, choices we made for the software part of our project are presented. The software is related to a 3D environment. We thus need to use a 3D API to create the 3D scene. As explained in the analysis part of this report, several possibilities are available to us. The first choice is to decide if we would use a 3D engine or not. We can indeed create the 3D environment using a basic API such as OpenGL or DirectX, but using an existing 3D engine also suits the project needs.

5.2.1 Comparison between OpenGL/DirectX and 3D engines

Advantages of an OpenGL or DirectX solution:

- Using OpenGL or DirectX, we would be able to begin from a very low level of programming. This would offer us numerous possibilities, and would allow us more freedom during the development of the software.

- OpenGL and DirectX are the APIs that are the “nearest to the hardware”. They are directly interacting with the graphics card drivers.

- Most graphics cards are compatible with both OpenGL and DirectX.

- OpenGL and DirectX are both very famous in the world of 3D programming. That means a lot of documentation is available on the Internet. It is also easier to get help when facing an issue with OpenGL or DirectX.

Disadvantages of an OpenGL or DirectX solution:

- Those APIs consist of sets of very basic functions. If the programmer wants to draw complex objects, he will have to create his own functions.

- Most optimisation algorithms are not implemented. They are to be developed by the computer programmer.

Advantages of a 3D Engine:

- 3D Engines make it easier to create complex objects, or optimized algorithms, since they are made for this purpose.

Disadvantages of 3D Engine:

- 3D Engine being specialized, they allow less freedom during the programming.

- A bad choice in the 3D Engine could make the application run slow, or even render it incompatible with some systems.

- Some 3D Engines are not compatible with every graphics cards.

- Most 3D Engines are not as generalized as OpenGL and DirectX, meaning that help is harder to find in some cases.

Weighting all those arguments, we choose an OpenGL or DirectX solution. 3D programming being new for both of us, we decided that it would be more interesting to begin studying this field of knowledge with a low-level view. Moreover, we saw many tutorials for OpenGL as well as DirectX on the Internet. Also, choosing a 3D Engine would have exposed us to the risk of making a wrong choice, thus leading us to dead-end issues.

5.2.2 Comparison between OpenGL and DirectX [45][46][47]

However this leaded us to another dilemma. We needed to choose between OpenGL and DirectX. Both are very well known, and both are used in many applications, and in many domains. They are much documentation for each of them, on the Internet as well as in books, and there are also many comparisons between the two of them. It came to our mind that, in order to make a correct choice, we needed to make our decision based on the type of application we wanted to develop. We need to choose the best suited API for a 3D architecture game.

Advantages common to both OpenGL and DirectX:

- OpenGL and DirectX are both well known APIs for 3D programming. They are reliable and compatible with nearly all graphics cards.

- Documentation and samples are available on the Internet. Numerous books are dealing with OpenGL or DirectX. Help can easily be found for both of them, when facing an issue.

Advantages of OpenGL:

- OpenGL is cross-platform. It means that, providing the language of the program using OpenGL is also cross-platform, the application can be executed on nearly any operating system.

- Bindings with C++, C#, Java and many other languages exist, which means that OpenGL functions can be called from numerous programming language.

- Many support libraries were designed to complete OpenGL. GLUT, SDL, GLU, GLee, GLEW are good examples of such libraries. Each of them added new functions that are completing OpenGL.

Disadvantages of OpenGL:

- OpenGL by itself is not sufficient for most 3D programs. Support libraries are thus necessary. This is a disadvantage because OpenGL and its support libraries do not make a unite block, which can create some issues in some cases.

- When using a binding with C# or another .NET language, the program created is not cross-platform anymore.

Advantages of DirectX:

- DirectX, or rather Direct3D (the part of DirectX used for 3D programming) can be considered as one entity. There is no need to install other libraries to complete it. Every functions needed are already available in the Direct3D API.

- Bindings with C#, C++, C, and other languages exist.

- DirectX is much more used for developing games than OpenGL. Help is easily found on game programming.

Disadvantages of DirectX:

- DirectX is not cross-platform.

5.2.3 Our choice

We decided that our program did not need to be cross-platform. Indeed, some of the hardware drivers we found worked only on windows, and that is why we decided that the whole project would be running on a windows operating system. As a matter of fact, we chose DirectX, because it seemed easier to find documentation (since we were making a game). Also, the fact that DirectX could work with C# was also an advantage for us, since we were familiar with this language. Finally, OpenGL works with associated libraries, and it seemed to us that it was a potential source of problem (problems of compatibility for example).

6

Specification of requirements

6.1 3D environment

The 3D environment must offer the user a basic game experience. The goals and the rules of the game must be simple, and the user interface must be user-friendly. The user must be able to move objects of the 3D environment using the fingers tracking gloves, in a convenient and user-friendly way. Furthermore, the application must allow the user to manipulate those objects in different ways. It must allow the use of different types of objects, with some available actions for each of them.

The 3D game must be simple but still offer some immersion to the user. The scene must be limited to a specific area, but offer the possibility to change the point of view, according to the position of the user's head.

6.2 Motion Tracking

The fingers and head tracking hardware components must be cheap, reliable and generalized. To allow an easy use without any knowledge in Computer Sciences, the tracking motion must offer an option to initialize the tracking with few calibrating parameters. The tracking hardware components must be compatible with an "IBM PC compatible" computer. The fingers and head tracking software must distinguish reliably the fingers and head infrared lights emitters avoiding fingers loosing.

- **Fingers Tracking:** The fingers tracking accuracy must be sufficient to select and move an object displayed on the screen without any loss. The fingers tracking must be fast enough to provide the feeling that the objects follows the user's fingers.

- **Head Tracking:** The head tracking must be fast enough to provide a perfect immersion and immediate 3D effect when the user moves his head.

Part III Design

This part describes the different modules to develop for this project. The first part presents the hardware components developed for the fingers and head tracking. The tracking software is then described in a second part. Finally, the development of the three dimensions game and of the user interface using DirectX9 is detailed.

1

General Structure

This part presents the global structure and the flows of the project. To facilitate eventual evolutions or improvements of the system, the modules are fully independent between each other. New tracking interfaces can be added without any alteration to the game engine.

The underneath figure illustrates the interactions of each module in the project.

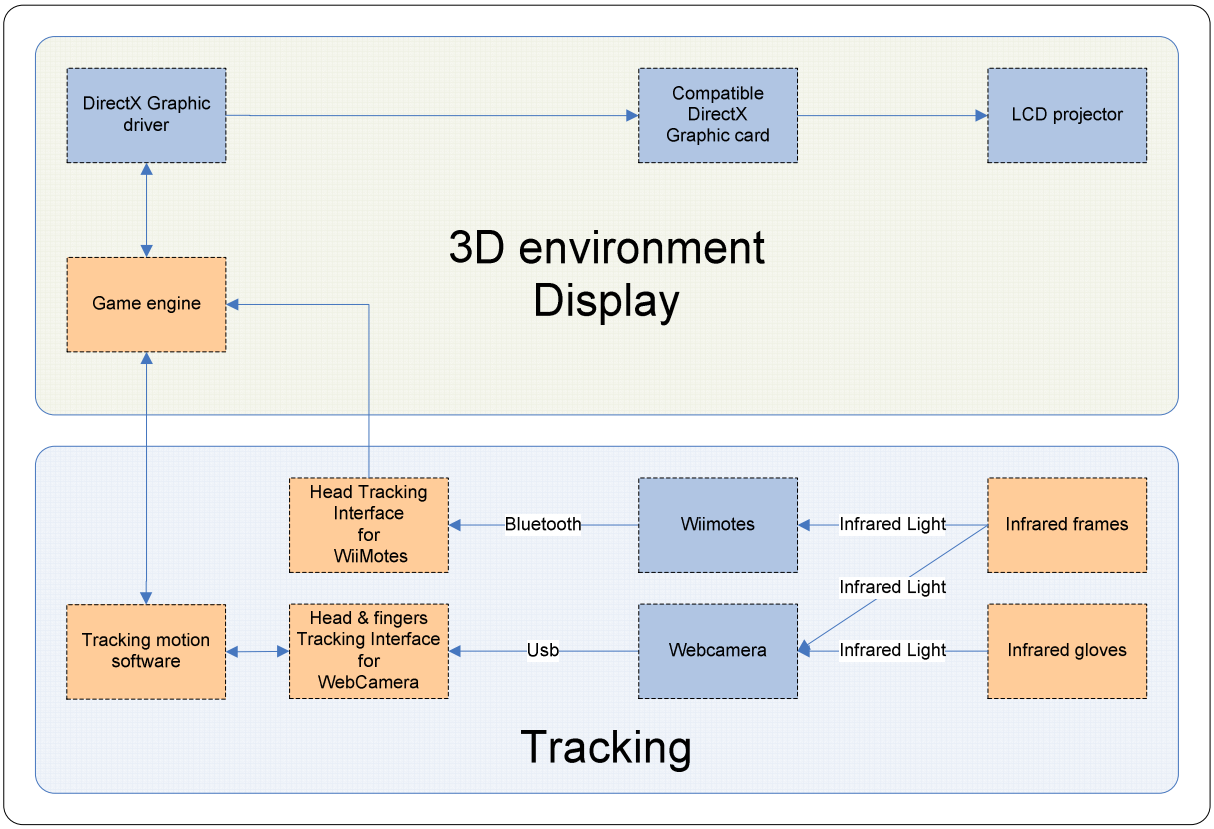

Figure III-1 : Project structure

Infrared diodes installed on the user's frames and gloves emit light detected by infrared cameras (wiimote & webcam) in an indoor environment not prone to sun infrared light emissions.

Two wiimotes process the picture shot and send the coordinates of the detected diodes using their Bluetooth connection. The 3D environment game engine directly connects using a Timer to the wiimotes using the head tracking interface WiimoteLib dll available on CodePlex website[48] and retrieve 2D position of the infrared diodes. It then calls methods from the tracking motion software to convert the 2D positions retrieved in 3D localizations.

The webcam directly sends the shot pictures to the motion-tracking interface. The tracking motion software then processes the wiimote and webcam information and sends the retrieved information using events. Using the fingers and head localizations, the game engine then updates the picture displayed to the user.

2

Tracking

This chapter describes the design of the tracking hardware equipments needed and is composed of two independent parts:

- Head tracking components
- Fingers tracking components

2.1 Head Tracking

This part presents the components designed to implement a head tracking technology that meets the quality requirements defined in the Installation layout part (Section Part II3): a head tracking solution able to localize accurately the infrared frames of the user at a range set between 3 and 4.5 meters with a pair of wiimotes.

2.1.1 Head Tracking Receivers

The head-tracking receivers used for the project are two wiimotes. Amongst the numerous controls implemented in these devices, we decide to focus on the infrared camera embedded. This camera offers a very good accuracy and is designed for infrared lights. Even if Nintendo Company did not publish any detailed datasheet about their product, an active community of users studied the components embedded and developed dedicated Application Programming interfaces (API) to use it with a computer. A Bluetooth connection and a C# compatible API Dynamic Link Library *(*dll*)* enable any .net software to receive the positions of diodes tracked by the wiimotes.

Name	Nintendo RVL-CNT-01
Communication	Bluetooth connection
Interface	Human Interface Device (HID)
Camera	128*96 (monochrome)
Accuracy	1024*768 (8x subpixel analysis)
Frequency	100 Hz
Diode optimal frequency	940 nm (with infrared filter)
Number of tracked diodes	1-4 (simultaneously)
Camera refresh rate	Unknown
Field of view	45° (horizontal & vertical)
C# managed API	WiimoteLibv1.7 - http://www.codeplex.com/WiimoteLib

Figure III-2 : WiiMotes technical details[49]

Figure III-3 : Wiimote device [50]

2.1.2 Head Tracking Emitters

The first tests made during the analysis of the capabilities of the wiimotes showed that the embedded camera is very sensitive to the infrared sources and can easily loose the tracked point if the user is at more than 2.5 meters. As the tracking software of the wiimote cannot be changed, the frames are designed and fitted with very strong infrared lights detectable by the wiimote at a distance of 4.5 meters at least. Only a few similar devices have been found on the studied sources, their technical descriptions are inexistent, and many prototypes have been designed before finding the best solution. Without any simulation software to test their efficacy, all the presented solutions have been built for testing purposes.

The different iterations of the infrared frames designed during the project are presented in the following parts. We will then conclude by a small comparison of all these prototypes.

2.1.3 Infrared frames without infrared diodes boosting

The first infrared frames prototypes design is very simple and mainly inspired by the pictures from the frames found on the internet. The glasses of the shades are removed and two infrared diodes are installed on the left and right edges of the frames in serial with a resistor that limit the maximum current.

Common infrared diodes have a forward voltage around 1,3V and a forward current of 100 mA. For this reason, a DC source of at least 2.6 V is needed. Non-rechargeable alkaline batteries are well adapted to this case. This type of battery is cheap, easy to buy for the user, safe and offers sufficient power for this specific need. Two AA batteries in serial (DC source : 1.5 volt each 2890 mA/h) provide the needed direct current. As the wiimote camera filter lets pass only light wavelength around 900 nm [49], diodes emitting in the [860-970 nm] range were implemented.

Three different prototypes based on this circuit were built with three different types of diodes:

Diode name	Forward voltage (V) T_c = 25 °C	Forward current (mA) T_c = 25 °C	Resistor (Ω)	Wavelength (nm)	Half angle (°)
LD 242-3 [51]	1.3	100	4	950 ± 20	± 40
LD 274-3 [52]	1.3	100	4	950 ± 20	± 20
SFH 487-2 [53]	1,5	100	1	880 ± 20	± 20

Figure III-4: Diode tested characteristics.

Figure III-5 : Diodes tested

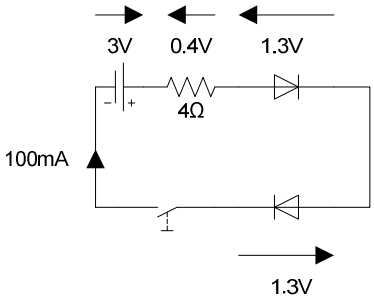

Figure III-6 : Infrared diodes circuit without boosting schematics

Using these prototypes, similar tests have been made with a wiimote to determine the type of diodes to use. Two specific factors have been studied:

- The depth range detection of the diode by the wiimote
- The maximum angle of light diffusion between the diode and the wiimote sensor

Analysis of the obtained results

- Depth range detection:
  LD 242-3: 2m
  LD 274-3: 2.5m
  SFH 487-2: 1.25m

- Maximum angle of light diffusion between the frames and the wiimote camera :
  LD 242-3: 40° (at 1.75m)
  LD 274-3: 15° (at 1.75m)
  SFH 487-2: 30° (at 1m)

These tests highlight the importance of the chosen wavelengths shown in the Figure III-4 for the efficiency of tracking. The detection is highly improved when wavelengths around 950 nm are used. The SFH 487-2 diode emitting at 880 nm is much less detectable than the two other types and cannot be implemented in such device. LD 274 is currently the diode with the best depth range tested but its low half angle value is a major issue. Indeed, to be constantly detected by the wiimotes, the LD 274

embedded frames have to remain almost perfectly parallel to the wiimote camera sensor. The LD 242-3 diode is much more easily detectable when the user's gaze direction is not perfectly perpendicular to the wiimotes sensors. However, the LD 242-3 depth range has to be enhanced.

Even if the LD 242-3 diode does not meet the depth range analysis defined requirement, it is by far the most homogeneous component studied and is selected to equip the infrared frames.

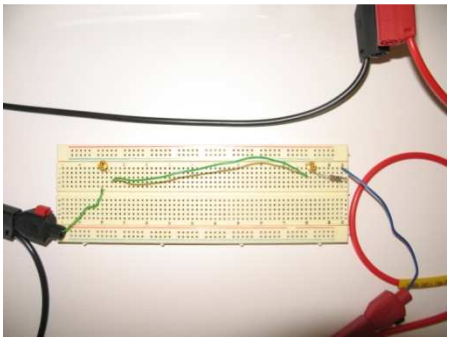

Figure III-7 : Infrared diodes circuit without boosting (with LD 242-3)

2.1.4 Infrared frames with improved light intensity

The diodes selection tests reveal that the most homogeneous diode for the head tracking is the LD 242 -3 type. It also highlights the need to improve the depth range value.

Two enhancement solutions are identified:

- Adding multiple diodes to obtain a larger blob.
- Increasing the infrared light intensity of the diode.

Coupling diodes solution

The precedent tests reveal that the wiimote has difficulties to detect the infrared diodes because the blob shot by the camera is too small. An enhanced prototype of the infrared frames has been developed using the LD 242 previous circuit. Four diodes are soldered in serial with a resistor of 8 Ω and a 6 V DC source (4 AA batteries).

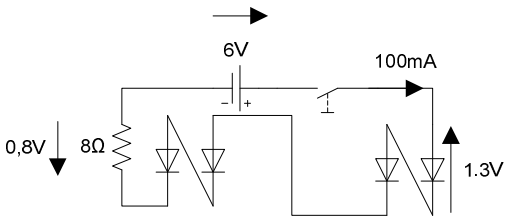

Figure III-8 : Coupling diodes solution

The tests realized with the wiimote are disappointing:

- The depth range detection improvement is negligible: 2 m with 1 diode ➔ 2.5 m with 2 diodes (+25%). The needed range is 4.5 meters.

- The weight of the battery holder becomes too important to be carried by the frames sticks : ≈ 160 grams
- The maximum angle of light diffusion between the frames and the wiimote sensor is increased to 45° (+12.5%).

As this solution is not optimal neither for the user comfort (weight of frames) nor for the efficiency of the head tracking (range limited), the following prototypes designs presented introduces the "led boosting" technique to increase substantially the infrared light intensity.

Switch based boost diodes solution

The LD242-3 diode is an excellent compromise between the light diffusion angle (80°) and the light power that are two main factors for wiimote light detection. For this reason, instead of testing new types of diodes, it is now preferred to improve the existing prototypes by increasing substantially the infrared light intensity.

Boosting a led is performed switching it on and off several times per seconds. During the "On" phase, a peak current is delivered to the diode with a high voltage value. During the "Off" phase, no current flows through the diode to cool it down. Besides cooling the diode, the "Off" phase is also necessary to limit the current consumption.

To prove the interest of diodes boosting, a first prototype has been designed using a Microchip 44-PIN demo board powered with two AA batteries (3V) to confirm the interest of diodes boosting [54] as a quick switch.

Embedded microprocessor	PIC16F887
Operating Voltage range (V)	2-5.5
Temperature range (°C)	40-125
Frequency (Hz)	32k-8M
Programming interface	PICkit™ 2 Programming Header

Tableau III-1 : Microchip 44-PIN demo board technical characteristics [55]

Capacitor (F)	33µ
Diode resistors (Ω)	1 (×2)
Transistor resistors (Ω)	100 k (×2)
Transistor (Mosfet N)	PMV60-EN (×2)
Power source (V)	3 (AA ×2)

Figure III-9 : Components used

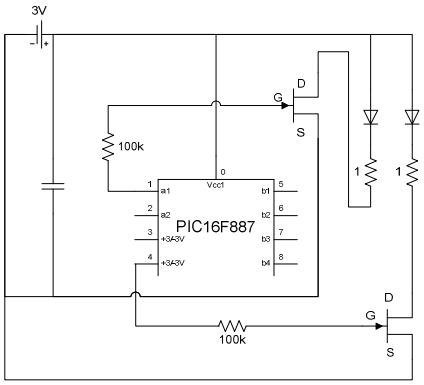

Figure III-10 : Switch based boost diode solution

The switch has been programmed using Assembly language to deliver some voltage input to the Mosfet gates. As the diode needs a high current to be boost, the "ON" current wanted is 1A. Using the graph Figure III-11 : Permissive pulse handling capability graph, the best "ON" status and "OFF" lengths have been selected: to obtain a 1A current, the maximum ratio between the "ON" and "OFF" period of 0.1 is possible without damaging the diode. An overall length cycle (**T**) of 10^{-4}s has been chosen.

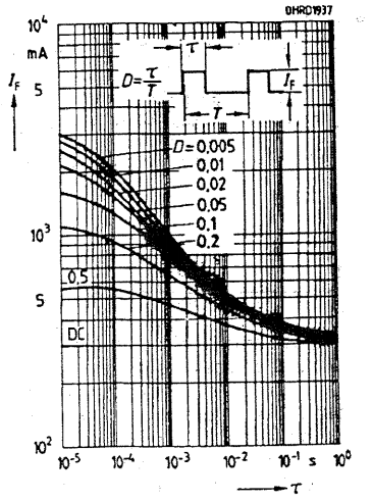

Figure III-11 : Permissive pulse handling capability [51]

Using this ratio, the diode "ON" status length has been calculated: $\boldsymbol{D = \frac{\tau}{T}}$

$$\Leftrightarrow$$

$$\boldsymbol{\tau = T * D = } \ 10^{-4} \times 0.1$$

$$\boxed{\boldsymbol{\tau = 10^{-5} s}}$$

On the Figure III-12 : Chip output voltage, one of the output voltage of the chip is displayed. The "ON" status is showed by the 3V and 10µs duration line whereas the "OFF" status is showed by the 0V and 90µs duration line. Two outputs of the chip have this type of signal with "ON" status occurring at different time to limit the batteries maximum current needs.

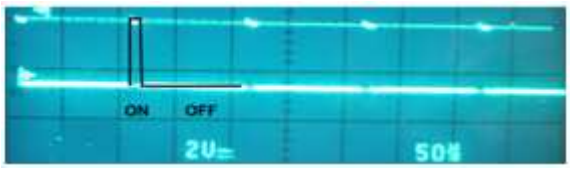

Figure III-12 : Chip output voltage

Each output is connected to the transistor gate. The transistor drain is connected to the diode and the source to the ground. The drain current is low when there the gate voltage is low but is amplified when the gate voltage is high (>1.5V). Using the chip signal, the transistor is used as a simple switch:

1. When the gate has a 3V voltage ("ON" status):
   The drain current is high. The voltage measured between the drain and the source is low. The voltage measured at the infrared diode is 1.7V and the voltage measured at the resistor is 1V. Using the Ohm's law (U=RI) with R = 1Ω, we deduce that the drain current reaches 1A when the diode is ON.

2. When the gate has a 0V voltage ("OFF" status):
   The drain current is insignificant (≈0A).The voltage measured between the drain and the source is high (≈2.9V). The voltage measured at the infrared diode and the resistor is 0V. We deduce that the drain current equals 0A.

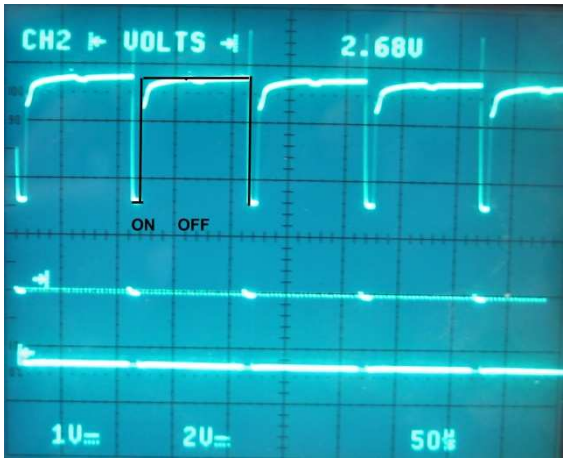

Figure III-13 : Transistor voltage between the drain and the source

The theoretical power consumption of one infrared diode and its resistor is:

$$P = UI = \tau \times (U_{DON} \times I_{RON} + U_{RON} \times I_{RON})$$

$$\boxed{P = 0.1 \times (2.7 * 1) = 0.27W}$$

The tests realized with this circuit and the wiimote show a good enhancement of the depth range. Increased by 40% compared to the coupling diodes solution, the detectable depth range reaches the 3.5 meters. As for the last circuits, the wiimotes often looses the tracked points if they move too fast or if the diode is pointing the wiimote with an angle greater than 30°.

The power consumption is also drastically reduced compared to the coupling diodes solution but could be reduced using a less complex diode with an adapted frequency processor (100 kHz instead of 8 MHz

for example). By using this chip, we would also need to develop a completely new miniaturized electronic board, which is not so easy due to the size of the chip. For all these reasons, after having confirmed the interest of diodes boosting, we have decided to seek for an equivalent miniaturized circuit.

PR4401 based coupling diodes solution

Looking for a miniaturized, low voltage chip with a reduced consumption to boost the diodes, we found that the led driver PR4401 chip meets the quality requirements [56]:

- Its size is reduced (2.92×10^{-3}m * 1.92×10^{-3}m).
- It powers the diode with some output peak currents (200-250 mA).
- The diode mean current delivered to the diode can be set using different coils.
- It has a good efficiency (80%).
- It requires a very low voltage input (>0.9V).

The PR4401[57] chip is a 500 kHz oscillating chip able to convert a low voltage source to a higher voltage up to 15V. During a "charge phase", the transistor output is switched "ON" and delivers a current to the coil storing energy. This "charge phase" depends on the power source current and voltage. Once the "charge phase" is finished, the transistor output is switched "OFF" and the coil releases its energy into the diodes with a peak output current. A comparator embedded in the PR4401 chip detects the end of the coil discharge and switches "ON" the transistor to begin a new "charge phase".

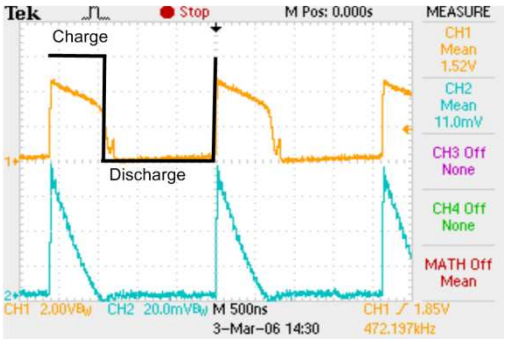

Figure III-14 : Diode voltage (yellow) and diode current (blue) using a 10µH coil [56].

To power the diode with the highest current, it has been decided to use a 10µH inductance that delivers a mean current of 22mA. As the chip requires low current and voltage, the power source is an AAA battery. The low weight of this kind of battery enables us to install the whole circuit on the frames. The typical battery lifetime with a 10µF coil is 8 hours according to the datasheet.

The tests realized with this circuit and the wiimote show that the maximum depth range using one diode equals 3.25 meters and can be increased to 3.5 meters coupling two diodes. The wiimotes still detects the diodes if they are pointing the wiimote with an angle of 25°. As we want to have the maximum depth range possible, we choose to use the coupling diode circuit in the infrared frames. Each side of the frames will be fit with the PR4401 based coupling diodes circuit and its own battery. Using this design, the symmetry aspect of the frames is respected, the weight is balanced and the wires length is reduced allowing more flexibility for the user.

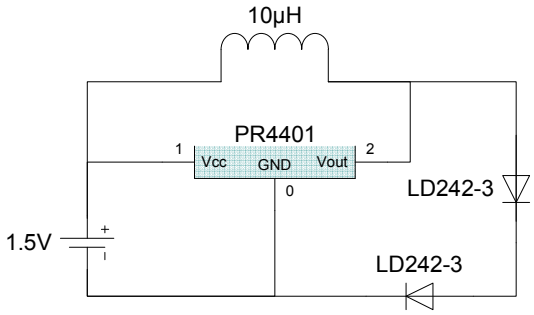

Figure III-15 : PR4401 based coupling diodes circuit design

Head tracking emitters comparative statement

Diode	Number of diodes	Boosting	Battery	Range (m)
LD 242-3	2	No	2 × AA	2
LD 242-3	4	No	4 × AA	2.5
LD 242-3	1	Yes (PIC16F887)	2 × AA	3.5
LD 242-3	1	Yes (PR4401)	2 × AAA	3.25
LD 242-3	2	Yes (PR4401)	2 × AAA	3.5

Figure III-2 : Comparative statement of the different designs tested

2.2 Fingers tracking components

This part presents the components designed to implement a fingers tracking technology meeting the quality requirements defined in Installation layout part (section Part II3): a fingers tracking solution able to localize accurately the infrared lighted fingers of the user at a range set between 3 and 4.5 meters with a webcam.

2.2.1 Fingers tracking receiver

The sensor used to track the fingers infrared diodes is a webcam. We take advantage of the possibility for the **CCD** sensor (Charge-Coupled Device) to capture infrared frequencies (745nm ⇔ 100 µm) by removing the infrared filter between the lens and the CCD sensor and by adding a visible spectrum light filter.

Webcam

Any model of webcam on the market may be used to track the infrared light if OpenCV (Open Computer Vision Library) recognizes it and if its infrared filter has been removed. A resolution higher than 320 × 240 pixels, an autofocus and a frame rate of 15 frames per second at least is recommended to use the application with a good accuracy. As part of the project, a **Sunnyline** camera is used.

Resolution (pixel)	640 × 480
Frames rate (picture/s)	15
Cmos sensor	300 000
Format	Avi
Focus	Manual

Interface	USB 1.1

Figure III-16 : Sunnyline webcam specifications

Visible spectrum filter

As webcam is used in a room with controlled light, we can minimize the infrared light emitted to the camera sensor. The visible spectrum filter is a square of black exposed camera negative placed on the webcam objective [58].

2.2.2 Fingers tracking emitters

To design the fingers tracking emitters, the studies made during the head tracking emitters' designs are reused. During the preliminary tests made with the visible spectrum filter on the objective, it appeared that the camera could easily distinguish a single infrared light from the background at a range of 4 meters (2.1.3 Infrared frames without infrared diodes boosting). These tests have been performed in a room with controlled light and low infrared light reverberation. Contrary to the wiimote sensor, the camera sensor allows a high angle between the infrared light direction and the lens (at least 50°).

Figure III-17 : Infrared diodes viewed by the camera at a 4 meters range

These tests highlight that we can use the same type of diode for the hand and fingers tracking as the webcam and the wiimote are both sensitive to the same frequency (950nm). Furthermore, they reveal that diode boosting technology is not required in the circuit design even at large depth.

A robust and simple circuit has been designed to power the infrared diode set on the index finger of the glove. Each glove has its own circuit to avoid long wires. To prevent any finger strain, a main issue in fingers tracking applications, the overall weight of the circuit has been reduced: a 3V coin cells (3g – 230mAh) replaces the two AA batteries and the wires connecting the diode to the battery are very supple. A press switch is installed to turn on the diode when pressed to power the diode only if necessary reducing the current consumption.

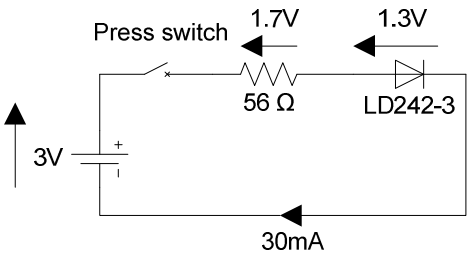

Figure III-18 : Fingers tracking infrared diode circuit design

2.3 MotionTracking solution

MotionTracking solution has been designed as a modular **Dynamic Link of Library** (dll) written in an object language. It includes numerous methods and algorithms processing the 2D positions of the infrared diodes, distinguishing the frames diodes from the fingers diodes and converting 2D coordinates in 3D coordinates.

2.3.1 HeadTracking class

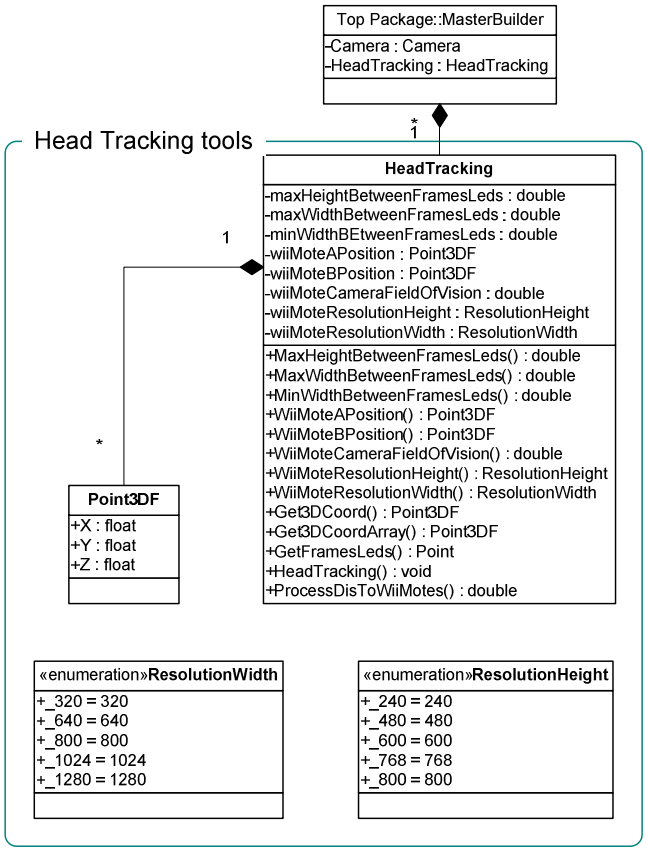

Figure III-19 : HeadTracking tools class diagram

The HeadTracking class design has been designed to handle the 2D infrared diodes positions retrieved by the two wiimotes. Before any position processing, it has to be calibrated with the needed parameters such as the position of the wiimotes, the wiimotes field of view or the maximum distance in pixels between the frames edges diodes.

It includes three main methods:

- **GetFramesLed**
- **ProcessDisToWiiMotes**
- **Get3DCoord**

GetFramesLed

GetFramesLed method recognizes amongst the infrared diodes detected by the wiimotes the diodes installed on the frames. Using the maximum distance and minimum distance values between the frames diodes (**MaxWidthBetweenFramesLeds** & **MinWidthBetweenFramesLeds** & **MaxHeightBetweenFramesLeds),** it processes the 2D points position retrieved by the wiimote, find the points validating the searching criteria and returned the frames edges positions in a table of Points.

Algorithm:

Sort the values of the infrared points by ordinates.
Sort the values of the infrared points by abscissa.

FOR EACH POINT A of the infrared points list.
    FOR EACH POINT B of the infrared points list.
        CHECK IF the distance between A and B is lower than MaxWidthBetweenFramesLeds (COND1).
        CHECK IF the distance between A and B is lower than MaxHeightBetweenFramesLeds (COND2).
        CHECK IF the distance between A and B is higher than MinWidthBetweenFramesLeds (COND3).

        IF COND 1 & COND2 & COND3 are true then ADD A and B to the frames list and exits FOR loops.
    END FOR
END FOR

Remarks: As the wiimote tracks a maximum of four infrared diodes and as two infrared diodes are already put on the fingers, only two infrared diodes can be installed on the frames.

ProcessDisToWiiMotes

ProcessDisToWiiMotes method processes the 2D coordinates of the frames infrared diodes to obtain the depth value of the head of the user. Using the wiimote field of view parameter (≈ 45°), it converts the pixels positions of the infrared points retrieved by the wiimote into angles position between the two wiimotes horizontal and vertical axes and the infrared points. Using triangulation and the distance (m) between the two wiimotes, it then processes the distance between the wiimote and the user's frames.

Mathematical explanation:

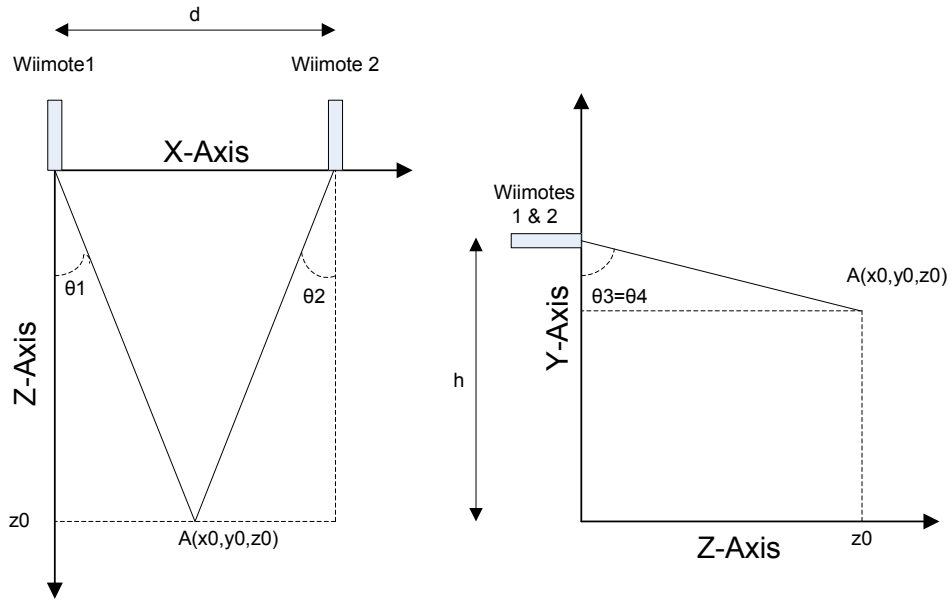

Figure III-20 : Wiimotes horizontal and vertical views

d: distance in m between the two wiimotes.
h: height between the wiimotes and the ground.
A: infrared diode tracked.
θ_1 and θ_2: lateral angles between the center of the wiimotes lenses and the infrared diode.
θ_3 and θ_4: vertival angles between the center of the wiimotes lenses and the infrared diode.

We assume that the two wiimotes are at the same height and depth. Using tangent definition in a rectangle triangle and the parametric equation, we find that that θ_1 , θ_2, θ_3 and θ_4 depends on the depth value.

$$\tan(\theta) = \frac{opposite\ side}{adjacent\ side} \Rightarrow$$

$$Wiimote1 \begin{cases} x_1(t) = \tan(\theta_1) \times t \\ y_1(t) = \tan(\theta_3) \times t + h \\ z_1(t) = t \end{cases}$$

$$Wiimote2 \begin{cases} x_2(t) = \tan(\theta_2) \times t + d \\ y_2(t) = \tan(\theta_4) \times t + h \\ z_2(t) = t \end{cases}$$

As θ_1 and θ_2 are known values, we can determine t value and finds the depth value (z_0).

$$\tan(\theta_1) \times z = \tan(\theta_2) \times z_0 + d$$

$$\Rightarrow z_0\ (\tan(\theta_1) - \tan(\theta_2)) = d$$

$$\boxed{\Rightarrow z_0 = \frac{d}{\tan(\theta_1) - \tan(\theta_2)} (in\ m)}$$

z_0 is the depth value measured between the user head and the wiimote.

Get3DCoord

Get3DCoord method converts 2D infrared diodes positions coordinates (in pixels) in 3D coordinates (in meters). Using **ProcessDisToWiiMotes** method to retrieve the depth between the user's head and the wiimote, it then processes the abscissa and ordinates values of the points coordinates ($\mathbf{x_0}$ and $\mathbf{y_0}$).

Mathematical explanation:

Using the formulas explained in 0 (ProcessDisToWiiMotes):

$$\begin{cases} x_1(t) = \tan(\theta_1) \times z_0 \\ y_1(t) = \tan(\theta_3) \times z_0 + h \end{cases}$$

$$\boxed{\Rightarrow x_0 = \frac{d \times \tan(\theta_1)}{\tan(\theta_1) - \tan(\theta_2)} \ (in\ m)}$$

$$\boxed{\Rightarrow y_0 = \frac{d \times \tan(\theta_3)}{\tan(\theta_1) - \tan(\theta_2)} + h \ (in\ m)}$$

2.3.2 Camera class

The **Camera** class has been designed to track the fingers infrared diodes. It retrieves pictures from a webcam with an USB interface and processes them to identify the points to track. It is also designed to track the user's frames because it has to consider the frames infrared diodes to distinguish the fingers diodes. It adopts an event-driven architecture improving the webcam/user interface interaction. Event-driven programming facilitates the integration of the **Camera** class methods in a application as the user interface programmer does not have to care for hardware specific characteristics such as the refresh rate of the webcam or the image processing time. As the image processing is time consuming, the image processing methods runs in its own thread enabling to process webcam data in parallel with the user interface display. The multithreading has been designed to be fully transparent for the user interface programmer who does not need to care about the thread management. OpenCV library (Computer Vision library developed by Intel company) methods are used for the real-time image processing (15 frames per second with a 640×480 pixels resolution).

Remarks: As the **OpenCV** library is a C++ library and that the **MotionTracking** dll will be implemented using a **.net language** [59], **EmguCV** [60] wrapper methods has been included in the design phase. **EmguCV** dlls enable any managed.net application to use directly C++ **OpenCV** methods.

Before any image processing, the **Camera** class needs to be initialized by defining the camera resolution and the resolution of the user interface screen in order to give fingers position adapted to the computer screen.

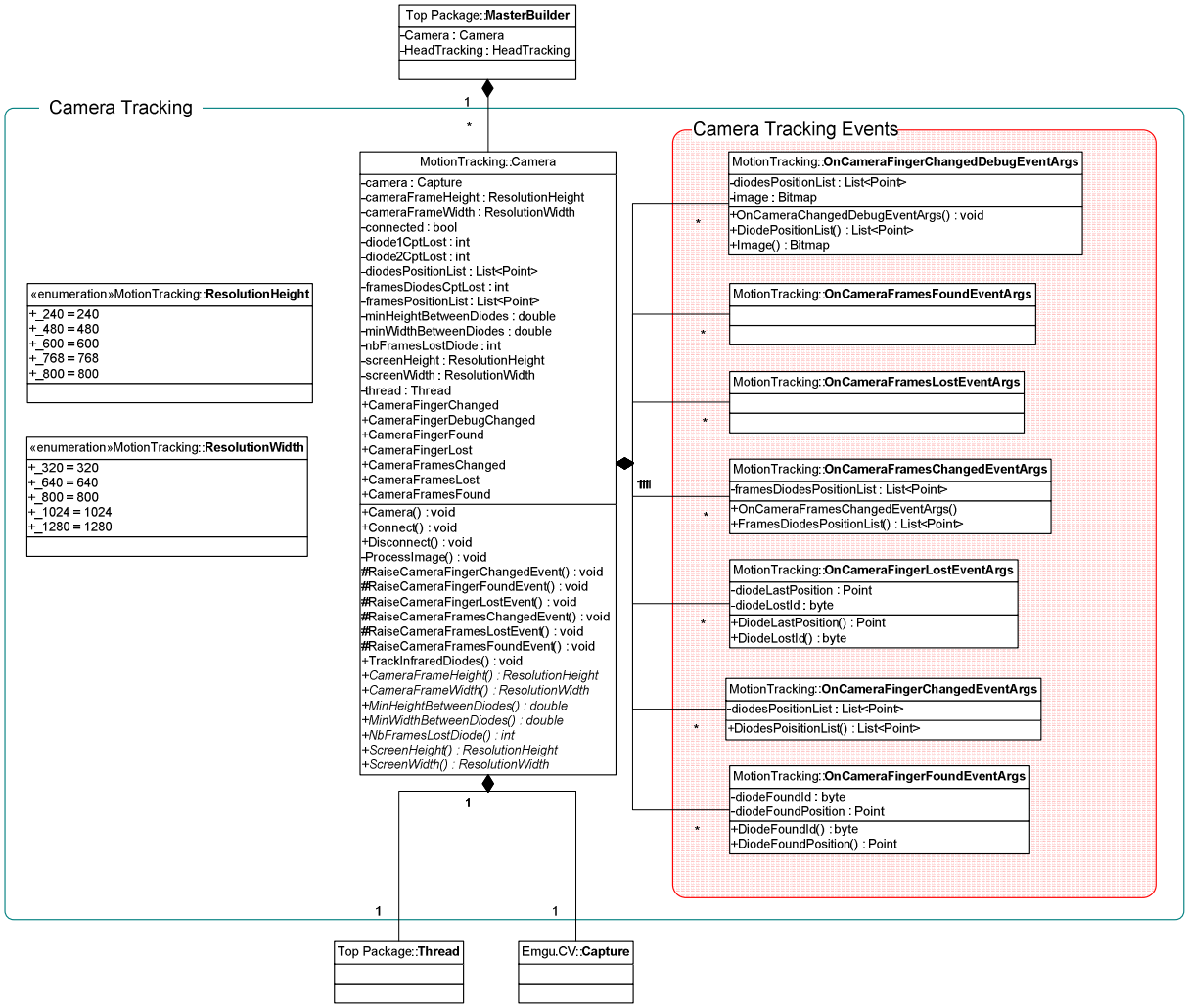

Figure III-21 : Camera class and its related classes

Camera Methods

Camera class includes nine main methods:

- **Connect:** This method connects to the webcam and sets the resolution of the image chosen.
- **TrackInfraredDiodes:** This method creates and launches a thread used to run ProcessImage method.
- **ProcessImage**: This method finds the position of the frames and fingers infrared diodes on the pictures captured by the camera. It also sends events to notify any modification of the frames or fingers positions.
- **RaiseCameraFingerFoundEvent**: This method launches an event notifying that a finger infrared diode has been found.
- **RaiseCameraFingerChangedEvent**: This method launches an event updating the position of one or two fingers infrared diodes.
- **RaiseCameraFingerLostEvent**: This method launches an event notifying that a finger infrared diode has been lost.

- **RaiseCameraFramesFoundEvent**: This method launches an event notifying that the frames infrared diodes have been found.

- **RaiseCameraFramesChangedEvent**: This method launches an event updating the frames infrared diodes position.

- **RaiseCameraFramesLostEvent**: This method launches an event notifying that the frames infrared diodes have been lost.

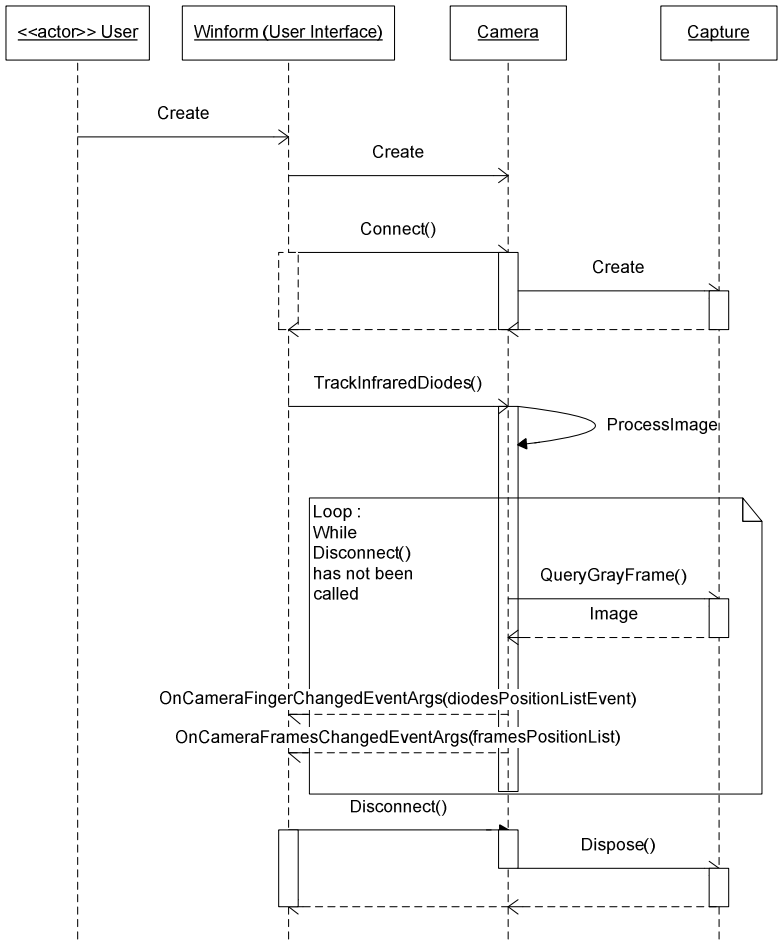

Figure III-22 : Motion Tracking Sequence diagram

The Figure III-22 : Motion Tracking Sequence diagram represents the asynchronous interactions between the **user** and the **Camera** class when the fingers and frames infrared diodes are tracked using a webcam.

Fingers and Frames detection algorithm

ProcessImage method is a loop receiving the pictures grabbed by the webcam and processing in real time each picture to find infrared diodes matching the user's frames or his fingers. To use this method, the maximum distance between the frames diodes has to be calibrated. The discovery of these diodes is achieved using the "Fingers and Frames detection algorithm".

Due to the sensitivity of the camera sensor, it could happen that a diode does not appear clearly on the picture provoking a LostEvent launching. For this reason, a variable enables the programmer to set the number of continuous pictures without the same diode to analyze before raising a finger or frames lost event.

Tracking more than four infrared diodes would introduce some issues with the head tracking performed by the wiimotes (maximum of four diodes tracked).For that reason, this algorithm is designed to handle four diodes, two for the frames and two for the fingers but can easily modified to track more than four diodes. Each finger diode has a permanent index value related to the order of apparition. The algorithm tracks the fingers motions enabling to keep permanent the finger index value even if the user crosses his hands (the left hand can be at the right of the right hand but is still recognized as the left hand).

The algorithm is decomposed in three main tasks:

- Discovery of the infrared lighted objects on the picture.
- Discovery of the frames amongst the lighted objects.
- Discovery of the fingers amongst the lighted objects.

Global algorithm structure

Initialize framesLostCpt counter to 0
Initialize finger1LostCpt counter to 0
Initialize finger2LostCpt counter to 0
Initialize a fingerPoints list
LostDiodeCpt is set by the developer and defines the maximum number of continuous pictures without a diode before launching a diode lost event

WHILE retrieving webcam picture

    Discovery of the infrared lighted objects on the picture.
    Discovery of the frames amongst the lighted objects.
    Discovery of the fingers amongst the lighted objects.

END WHILE

Discovery of the infrared lighted objects on the pictures

retrieve a gray picture (A) from the webcam
//Remove the noise lights and then improve the infrared diodes light area on the picture
erode picture (A)
dilate picture (A)
find the edges of the objects on the webcam picture (A)
find the contours of the objects using the previous edges detection

//Remove the flimsy contours
IF at least on contour is found
    remove the contours of big area (>200 square pixels)

```
        remove the contours of small area (<10 square pixels)
        //Remove the contours where the width and the height ratio is too high or too low
        IF contours width/contours height < 0.333 OR contours width / contours height > 3
                Remove the contour
        END IF
END IF
```

Remarks: The canny edging [61], consisting in plotting edges in an image, is processed by the optimized codes embedded in the **OpenCV** library through **EmguCV** .net wrapper.

Discovery of the frames amongst the lighted objects

```
FOR EACH contour A of the contours list
        FOR EACH contour B of the contours list.
                CHECK IF the distance between A and B is lower than MaxWidthBetweenFramesLeds
                (COND1).
                CHECK IF the distance between A and B is lower than MaxHeightBetweenFramesLeds
                (COND2).
                CHECK IF the distance between A and B is higher than MinWidthBetweenFramesLeds
                (COND3).

                IF COND 1 & COND2 & COND3 are true then ADD A and B to the frames contours list
                and exits FOR loops.
        END FOR
END FOR

//Launch the frames event

IF two frames contours have been found
        IF frames LostCpt > 0 and framesLostCpt < LostDiodeCpt
                framesLostCpt is set to 0
        ELSE IF frames LostCpt = LostDiodeCpt
                framesLostCpt is set to 0
                launch a CameraFramesFound event
        END IF
        launch a CameraFramesPositionChanged event
ELSE IF  frames LostCpt < LostDiodeCpt
        increment framesLostCpt
ELSE IF frames LostCpt = LostDiodeCpt
        launch a CameraFramesLost
END IF

remove the frames contours from the contours list
```

Discovery of the fingers amongst the lighted objects

```
//Attribute to each contours an index value

FOR EACH points (A) in the fingers point list
        find the nearest contours and assign to this contour the related point index
```

```
END FOR

assign to the others contours a new point index

//Launch the fingers events
//First finger
IF index 1 is assigned
        IF finger1 LostCpt > 0 and finger1LostCpt < LostDiodeCpt
                Finger1LostCpt is set to 0
        ELSE IF finger1 LostCpt = LostDiodeCpt
                Finger1LostCpt is set to 0
                launch a fingerFound event
        END IF
ELSE IF finger1 LostCpt < LostDiodeCpt
        increment finger1LostCpt
ELSE IF finger1 LostCpt = LostDiodeCpt
        launch a fingerLost event
END IF

//Second finger
IF index 2 is assigned
        IF finger2 LostCpt > 0 and finger2LostCpt < LostDiodeCpt
                Finger2LostCpt is set to 0
        ELSE IF finger2 LostCpt = LostDiodeCpt
                Finger2LostCpt is set to 0
                launch a fingerFound event
        END IF
ELSE IF finger2 LostCpt < LostDiodeCpt
        increment finger2LostCpt
ELSE IF finger2LostCpt = LostDiodeCpt
                launch a fingerLost event
END IF

//Update the fingers position
IF at least one index is assigned
        launch a CameraFingerPositionChanged event
END IF
```

After having presented the motion tracking design, the 3D environment which uses this tracking information will be described in the following chapter.

3

3D Game Design

In this part of the report, the 3D game design and the choices made in order to reach our goals are described.

3.1 Purpose and rules of the game

3.1.1 Main goal of the game

First, it is important to determine precisely the purpose of this game, and its role in our project. As it has been stated in the previous part of this report, the main idea of the project is to demonstrate that new Human Interaction Devices can be used efficiently in a 3D environment such as a game or some professional software. The development of the game has thus only one purpose in this project: it has to be a support for the two technologies we designed, fingers tracking and head tracking.

This means that the game itself must be designed in a way that will make the user use these HIDs as much as possible.

The main idea about the game, named “Master Builder”, has already been explained earlier. It is a game where the user has to take the role of an architect, and design some buildings. In order to design those buildings, the user has to put some objects in the scene, moving them and placing them at the right positions in order to achieve his goal. Keeping this idea in mind, there are other constraints that guided us in the design of the game.

3.1.2 Constraints for the game

Since the game is designed to test the HIDs, it is obvious that the game should enhance the usefulness of those new ways of interacting with the computer.

- In order to make the finger tracking useful, the game should ask the user to manipulate many objects. One of the main part of the game is thus to grab objects, move them, and place them on the scene. This kind of features really makes the user evaluate the usefulness of the finger tracking. To push further in this line of thoughts, the user should be able to make some modifications on the objects once they are on the scene. The action of grabbing an object and stretching it is the kind of handling for which people would find intuitive to use their fingers. The whole idea is to make the user feel like he is really grabbing objects and moving them in the scene with his real hands.

- As for head tracking, it is also very useful when manipulating objects. Depending on the position of the user's head, the position and angle of the camera change, allowing the user to look at the same object with different angles. It makes it easier for the user to visualize the scene, to understand precisely where each object is. Moving backwards, the user can go further from the scene, and thus see more objects in the view. Moving forward, it can examine an object with a closer view.

3.1.3 Rules of the game

Since the game itself is not the main part of the project, there was no use designing a very challenging game. The rules have to be simple all the while providing a minimum of realism as well. There is no "win or lose" part in the game. The user is quite free to build whatever he wants to. However, a small image of a model is displayed in the menu interface, and the user has to reproduce this model.

Figure III-23 : An example of model to imitate

Since it does not seem relevant to our project, no victory detection has been developed. The user can imitate the model, and stop only when he is satisfied by his imitation. It seemed pointless to try to detect the moment when the imitation is good enough.

Of course, since the user wants to build something, objects have to interact with each other. Thus, there is collision detection, avoiding that two objects go through one another. It also allows the user to put items on top of each other, and to put them exactly side by side.

In order to make easier the placement of objects, the game makes objects fall down to the ground or to an object beneath when they are not hold by the user. It is also more realistic this way, since objects staying in the air go against the basic gravity laws and seem strange.

3.2 Textures and Object Creation

3.2.1 Object drawing in DirectX

When creating a 3D scene, it is possible to design 3D objects using colors and shapes drawing methods in DirectX. However, this is a very slow and hard way to create those objects. DirectX allows the drawing of primitive shapes. Among those primitives, the most used are the triangles and the lines. It is thus theoretically possible to draw complex shapes with DirectX, using those primitive shapes. However, this is still very time-consuming, and hard. Textures may also be applied on those primitive shapes directly in DirectX code, but the same issue of time can be observed. Complex objects are usually imported from files located on the hard disk instead of being drawn using DirectX methods.

3.2.2 Meshes Import in DirectX

An object is called “mesh” in DirectX. It corresponds to the class “Mesh”, which is used to store all the data of a complex object in the same structure. This can become very useful when manipulating this object, or when drawing it.

It is possible to load a mesh from a “.x” file. With a simple method, the directX code can load and import the mesh structure contained in the file, and place it somewhere in the 3D scene. It is then quite easy and convenient to draw this object anywhere. In order to move the object, or draw it at a specific position, it is required to apply transformations to the view matrix of the directX scene, and then draw the mesh structure. Those are the transformations used in the program to draw a mesh:

- **Scaling:** The mesh dimensions in the file are too big, so they are reduced with a simple scaling. The scaling transformation is executed on the three axes with different scaling factors. It is thus possible to stretch a mesh (putting a higher scaling factor for one or two of the three axis).
- **Rotation:** The mesh is sometimes oriented in the wrong direction in the .x file. It is thus rotated so it has the right orientation in the game. Rotations may be made around an axis, or in a specific plan.
- **Translation:** When the right size and orientation of the mesh are set, the main problem is to draw it at the right place. In the .x file, the mesh often has coordinates near the origin of the axis. With a translation using the three axes, it can be drawn anywhere in the scene.

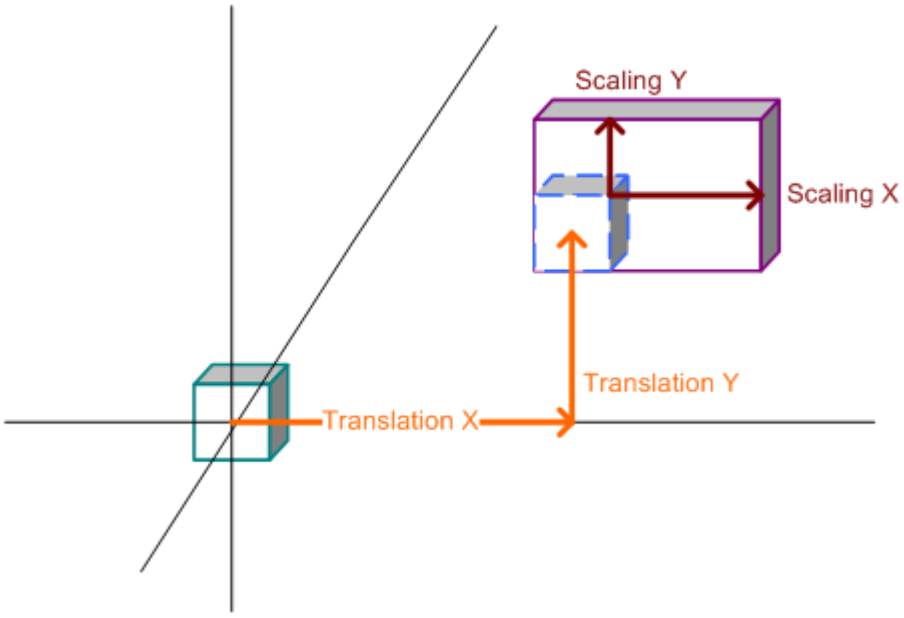

Figure III-24 : example of using transformations to place an object at the right position

Thanks to those three basic transformations, the mesh is easily drawn at the right place, with the right size and orientation.

3.2.3 Presentation of some of the meshes used in the game

All the meshes used in the game are elements of construction, which the user can create, move, and remove. Those meshes are not realist. They also share some textures, which give them a similar look,

enhancing the fact that they all belong to the same application. Also, the buildings created by placing those elements on top of each other do not look divided.

In this part of the report, some of those meshes are described. However, there is no point in making an exhaustive list of all the meshes used in the game. It is also important to note that new elements of construction may be added easily to the game.

- The Stone

The stone is the simplest elements available to the user. The mesh is a simple cube, to which a stone texture has been applied.

- The Pillar

The pillar consists of two cylinders, a cone part and a rectangular parallelepiped for the base.

- The Floor

The floor is a rectangular parallelepiped that has a very small height compared to its base.

3.3 Collision Detection

3.3.1 Purpose of collision detection

As it has been said, the objects handling is the main part of the game. It was thus important that objects interact with each other in a “human” way, which implies that they do not go through each other when moved by the user.

For this reason, the game engine has to process collisions detection, whenever an object is moved in the scene. This part of the report describes how the collision detection works in this application, and the design choices.

3.3.2 Circumstances of collision detection

Point collisions are detected only when an object is moved from a position to another. This happens only when an object is moved directly by the user, or by gravity law. The gravity law and the user interaction are explained later on.

3.3.3 Choice of the shapes for the collision detection

The main issue faced for collision detection is that objects are very different to each other, and their shapes are sometimes complex. This makes collision detection very hard to design, and very power-consuming for the processor. As the game must run smoothly, the collision detection design is optimized.

The simplest way, and the most used in game programming, is to check objects positions two by two, in order to find if there are any collisions.

With complex shapes, the program must check the intersection of every face, by trying all combinations, and this can be very power consuming for the processor. For that reason shapes are approximated for the collision detection. Many games use this strategy, in order to keep the collision detection simple and efficient.

By analyzing the shapes of the objects we designed for the game, we noticed that most of them can be approximated as rectangular parallelepipeds. This makes the collision detection much easier, since a rectangular parallelepiped is a quite simple and basic shape. It is also easier for the user to manipulate those objects. Moreover, it makes it easy to add new object type to the game if they can be approximated as rectangular parallelepipeds.

3.3.4 Detection of collisions

The collision detection is triggered by the movement of any object of the scene. This movement can be caused by an action of the user (grabbing the object and moving it, for example), or an action of the environment. The only action coming from the environment is the gravity, which makes objects fall to the ground. The gravity does not affect objects grabbed by the user, and the user cannot interact with objects currently falling. That means the two actions cannot act on the same object at the same time.

When a movement is requested by one of the two actions, it is not done immediately. The collision detection is “a priori”, meaning that it is performed before the object is moved. We preferred this method to the “a posteriori” detection, which implies a correction of the movement if collisions are found, after this movement has been performed. The “a posteriori” may introduce mistakes in the movement management. For this reason, we choose an “a priori” collision detection.

Each time a movement is requested, the collision detection algorithm is called. The purpose of this algorithm is:

1. To detect all the collisions caused by the movement requested. The algorithm provides a list of all the objects that disturb the movement requested.

2. To compute the “best” movement without collision. The “best” movement is the closest movement to the movement requested, with no collision. If no collision has been detected in the first step, then this “best” movement will be the movement requested itself.

In order to realize the step number 1, the algorithm checks for the presence of objects in a certain space. When the object changes, according to the movement requested, it then occupies a new space in the scene. The algorithm will thus concentrate on the space that was not occupied before the movement and will be after it.

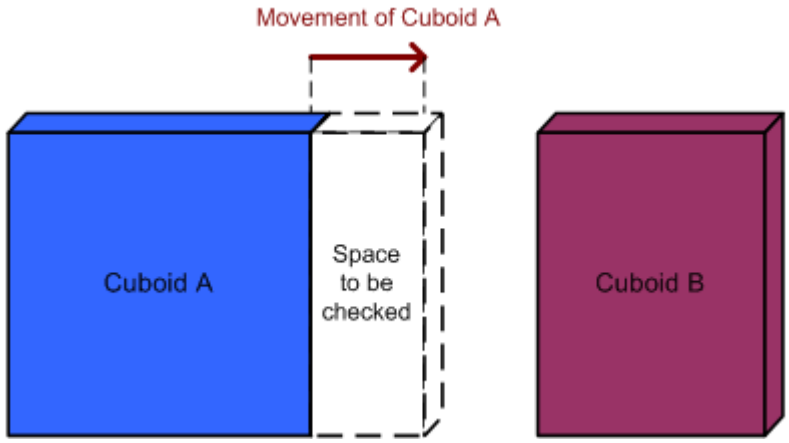

Figure III-25 : Cube A must be moved to the right. The program will check if the space corresponding to this movement is available.

Figure III-25 shows two cuboid shapes. The cuboid A requests a movement to the right, and eventual collisions caused by this movement must be detected. The algorithm will thus identify the space on the right of cuboid A. The space in white on the diagram is the space that becomes occupied by the cuboid A after the movement. This space must be checked for availability, which is true only if no objects are present in it.

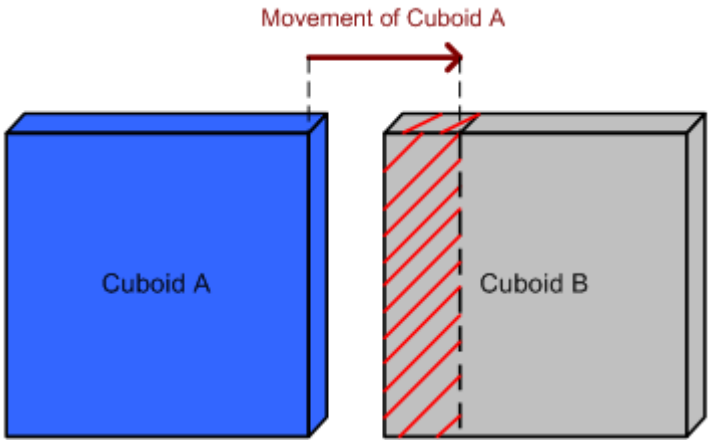

Figure III-26 : when checking the space for the movement, the program detects that there is a potential collision with Cube B

With Figure III-26, another case is represented. In this case, another cuboid is too close to the cuboid A for the execution of the movement without any collisions. The algorithm checks the space occupied by

the cuboid A after the movement, and notices the presence of part of cuboid B inside this space. Cuboid B is added to the list of collisions to be dealt with.

The algorithm is presented below with a pseudo-code.

Algorithm:

```
WHEN the user or the gravity law ask for a movement for object O
{
        IF the space S is available
                Changes the position of the Object O according to the movement
        ELSE
                Resolve the collisions found
}
```

The “WHEN” condition in the pseudo-code corresponds to event triggering in the program. When the user interacts with the program, it triggers events. This “event-based” system is explained in the User interface part of this report.

3.3.5 Collision issues resolving

The detecting collisions process has been described, however it is not sufficient for the game to react properly. The collisions detected must now be dealt with, which is the second part of the algorithm.

The most obvious solution to solve a collision is simply to cancel the movement. This is easy because the collision detection algorithm is an “a priori” detection.

However, this solution has disadvantages, especially in this type of game. The purpose of the game being to build a structure, it is very probable that the user would like to put objects side by side, or one on top of the other, without any space between them. With the solution of canceling the movement, it is nearly impossible to place an object right beside another, with no space between. It will now be explained why.

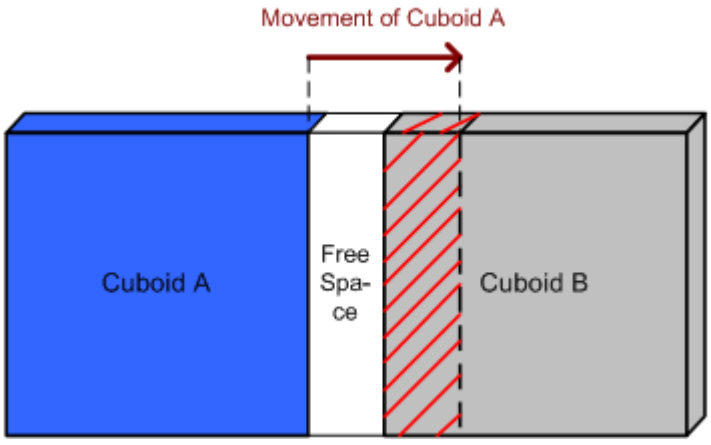

Figure III-27 : The user is trying to put cuboid A side by side with cuboid B. A collision is detected for this movement.

The Figure III-27 above shows that the movement of the user is too big, and triggers a collision. However, if the movement is canceled, some free space between cuboid A and cuboid B remains. The two objects will not be stuck together, as the user may wish them to be.

To solve this issue, we decided that in this case, the cuboid A is moved as much as possible in the direction of the required movement. In order to do so, the program selects one of the collisions, and determines the maximum movement that can be performed. Once this movement has been determined, the collision detection is used again, but this time on the new movement. The space occupied by this new movement is analyzed, and a new list of collision is made.

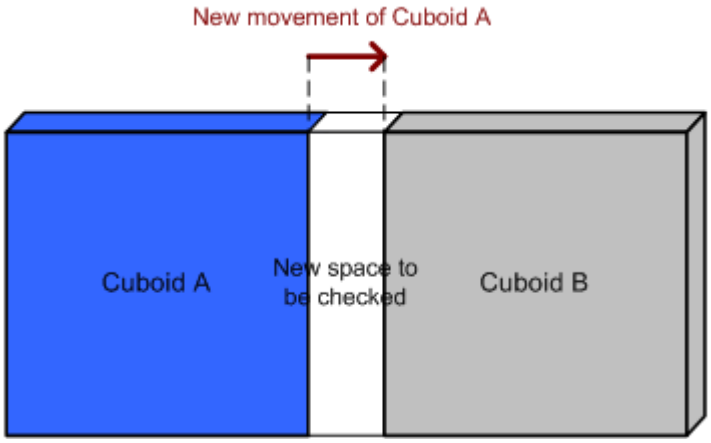

Figure III-28 : The movement has been reduced. A new space is to be checked for availability.

The algorithm goes back to the first step, with a new space to check. It continues until finding a space for which there is no collision. When this space is found, it means that the “best” movement has been determined. The movement can be performed.

3.4 Gravity Law

3.4.1 Purposes of the gravity law

There are two reasons for the presence of the gravity law in our game.

- Objects staying in the air, with nothing to hold them, are very disturbing for the user. In the previous part, we have stated several times that the realism of the game was not a priority for the project. However, it is required that a minimum of realism should be maintain, in order to keep the illusion that the user is building some architectural element.
- Gravity makes it more convenient for the user to build. The user does not need to place an object on the ground, or on top of another one : he can release his hold on this object when it is above the ground or another object, and the object previously hold will fall down by itself.

3.4.2 Gravity management

The purpose of this part of the game is that objects fall down when released. In order to make this happen, the program contains a timer, which triggers an event every 10 milliseconds. Since the gravity is not needed all the time in the program, this event does not trigger anything most of the time, in order to save processing power.

The gravity system is triggered only when the user grabs an object and releases it. The program then checks every object to determine if they have to fall down. This happens when two objects are on top

of each other and the user moves or removes the object below: the object on top is then floating in the air, and must fall down. That is why the gravity is triggered every time an object is moved and released by the user.

When this happens, a list of all the objects present in the scene is made. This list is given to the gravity system, to be checked out. Every 10 milliseconds, the program checks for each object if it can fall. If so, the object is moved down a little. When an object cannot be moved down anymore, it means its fall has stopped: it is then removed from the list of falling objects. When the list is empty, it is the end of the gravity check.

3.4.3 Example of gravity effect

An example of this effect is showed and explained. In the Figure III-29, three cubes are visible. Two of them are on top of each other, the third standing alone on the right. The user grabs and moves the turquoise cube.

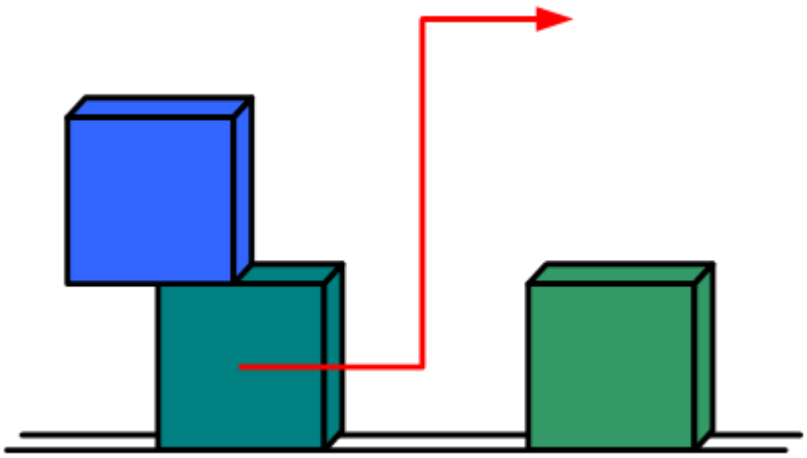

Figure III-29 : Example of gravity effect - One of the cubes is moved by the user.

As showed on the next figure, the turquoise cube is released in a position where it stands without support. Since an object has just been released, the gravity effect is triggered. A list of all three cubes is made, and checked. The blue cube and the turquoise cube can move down, whereas the green one cannot. The green cube is thus removed from the list at the beginning of the gravity effect.

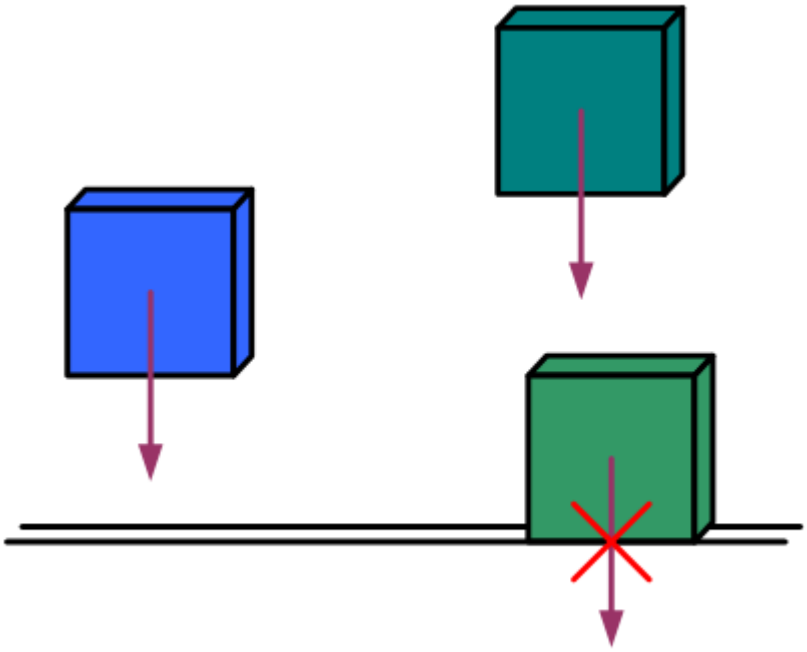

Figure III-30 : Example of gravity effect - Gravity effect is triggered on the 3 cubes.

The two other cubes are moved down every 10 milliseconds, until they cannot move down anymore. For the blue cube, this happens when it reaches the floor. It is then removed from the list. The turquoise cube becomes unable to move down when it collides with the green cube. It is then removed from the list as well.

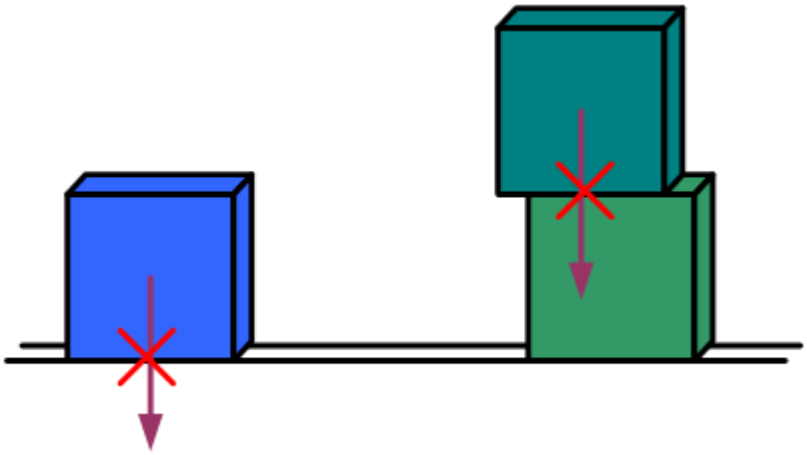

Figure III-31 : Example of gravity effect - End of the gravity effect.

The list being empty, the gravity effect is finished. All objects are in a “stable” position.

3.4.4 Limitations

This simple gravity is the only physical element that we designed for our game. Advanced physics could have been designed for the game as well, but it would have taken a long time. For this reason, the buildings designed through this application are not fully realistic, because obvious physics laws are not respected.

3.5 Shadows

3.5.1 Context

Nowadays, every game uses shadows to enhance the realism of the graphics. Shadows render objects more real, and give additional information to the user. The user is able to know the source light position and the exact localization of the objects.

Our game does not need to be realistic, but in a 3D environment, shadows help locate the position of an object. When an object is moved up, the user might get confused as to its real position in the 3D space. As a matter of fact, a simple shadow under the object helps the user see clearly where the object is.

On the other hand, shadows are hard to draw with DirectX. It requires a lot of processing, and complex coding. Most of the games prefer to use other software to compute shadows apart from the main display. This kind of software is called “Shaders”, and it is complex to use. It requires a long time to learn how to master such a tool, and we cannot not allow ourselves to spend this much time on a part of the project that was not necessary to achieve the objectives.

3.5.2 Decision

The main purpose of the shadows in our program is to show the position of the object on the ground plane. For this, a “real” shadow is not needed. We decided to draw only a dark square under the object when it is hold up in the air. The shadow is drawn directly under the object, and its size is related to the size of the object when approximated to a rectangular parallelepiped.

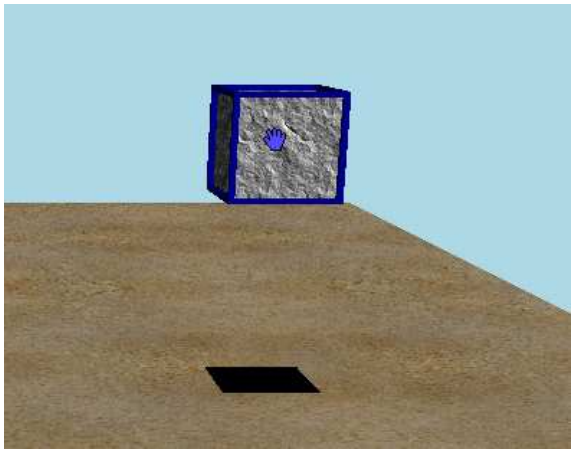

Figure III-32 : The shadow is drawn on the ground

If another object is under the object which shadow is being drawn, the shadow is drawn over this object rather than over the ground. The purpose of the shadow is not to add much realism to the scene, but rather to allow the user to see when an object is over another object, and when it is over the ground.

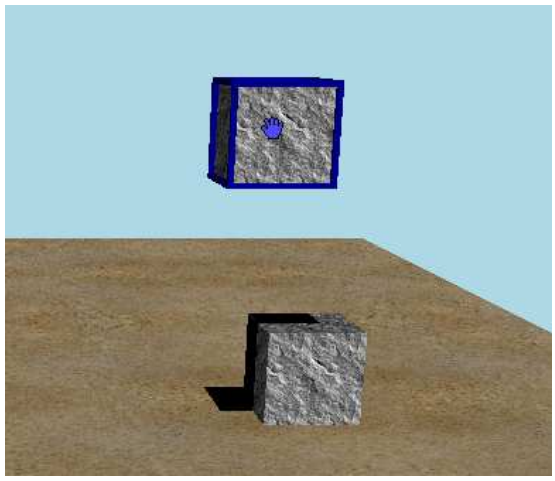

Figure III-33 : The shadow is drawn on the ground, on the side and the top of the other stone.

3.6 Program Structure

This report now focuses on the structure of the program itself. Details are not given here, but the structure, the classes, and the main algorithms are described and explained. As explained in the "analysis" part, the program has been written in C# using DirectX 9 libraries.

The application contains several classes that perform different duties in the game:

- **Program:** This is the class responsible for creating the WinForm and launching the application.

- **WinForm:** This is the form class, responsible for the display of the game window, as well as the display of the 3D environment. It contains the DirectX device, used to draw the 3D environment in the game window. It also contains the methods to initialize and manage the finger tracking and head tracking, using the motion tracking dll and the wiimote library.

- **SceneManager:** This class is in charge of storing all the information about the objects contained in the scene. It also contains the methods to draw the scene itself (not the menus and cursors). It also contains the identifier of the object currently selected, and the object currently held by the user.

- **TextureDataBase:** It is the class storing all the textures used in the game. The idea is to centralize all the textures objects inside one class object, which also contains all the methods to load those textures and manage them.

- **Menu, MenuTab, and MenuButton:** Those classes store all the information about the center menu. They have the position of the menu and its elements, their sizes, as well as the buttons

and the tabs contained in the menu. The class Menu contains the method used to draw the menu.

- **DesignBox and InfoBox:** Those two classes contains data and methods related to the left menu and right menu.

- **Cuboid:** This class contains the attributes and the methods used for all objects that can be approximated as cuboids (rectangular parallelepiped). Since in the scene all items are approximated as cuboids, the methods of this class are often used. Some examples of methods are the collision detection methods and the drawing methods.

- **Stone, Pillar, Floor and Stairs:** Those classes inherit of the Cuboid class. They add the specificities of each object to the attributes already defined in the Cuboid class.

- **Popup:** Class responsible for the display of pop-up message boxes.

- **ObjectID:** This class contains the attributes and the tools to identify an object in the list of objects stored in SceneManager.

The next part will focus on the user interface used for the 3D environment.

4

User Interface

In this part of the report, the ways for the user to interact with the 3D environment are described. First, the interface menu is showed and analyzed. The purpose of each button is explained. After this, the ways to interact with this menu and with the scene, via finger tracking, are described. The purpose of head tracking is finally explained at the end of this part.

4.1 Scene and plane of construction

The scene is the 3D environment where the user can create, place, move and remove objects. It is represented in the game by a simple horizontal square, located lightly below the user's point of view, and in front of it. The 3D camera is looking a point of this square.

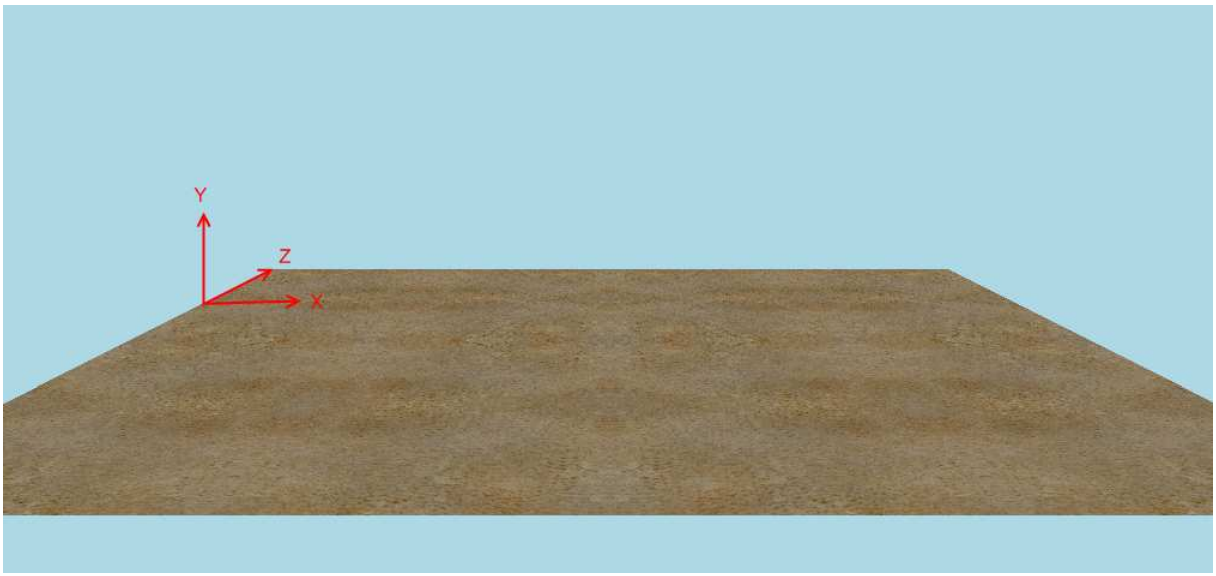

Figure III-34 : The scene is a square, where objects can be placed. The axes have been added to the image in red.

Since the camera of the 3D environment is above the scene square and looking at the scene square, it appears inclined on the screen. This gives the perspective impression. Objects are then placed in a 3D coordinates system. The axes have been drawn in red on the previous image.

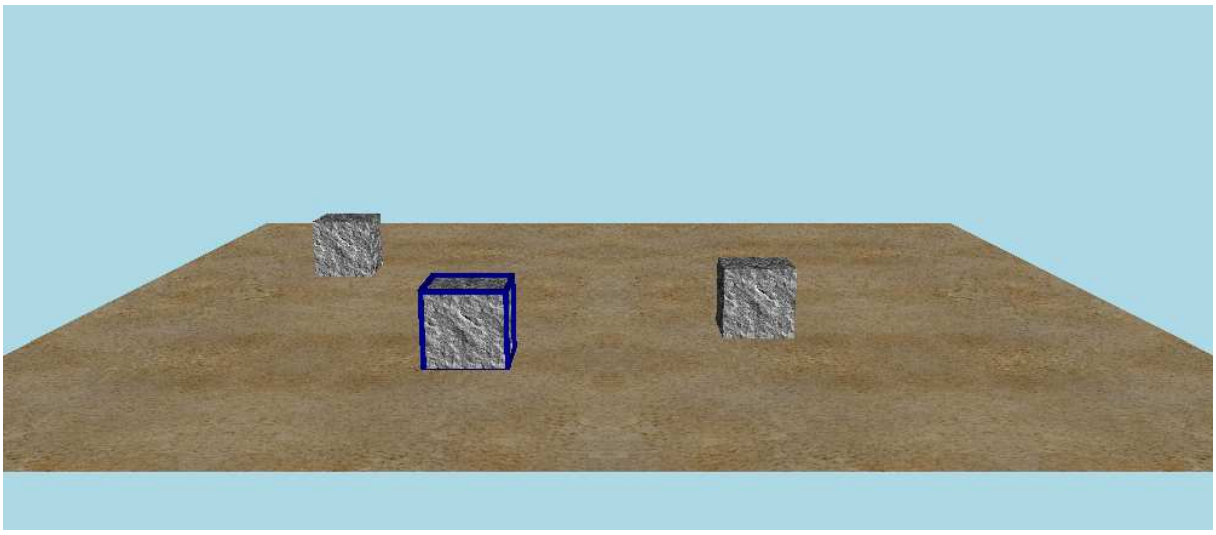

Figure III-35 : Objects have been added to the scene.

The Figure III-35 shows how the scene looks like when some objects have been added to it. The scene is a workshop where the user designs his building.

In order to add objects, and to modify them, the user has to use the interface menu, which centralizes all the tools available to him.

4.2 Interface menu

Like the rest of the application, the menu had to be intuitive and easy to use. That is why we chose a simple interface, with simple elements.

As in many games, the menu is divided in several parts, each part having its own purpose, giving information to the user or allowing him to take action on the scene.

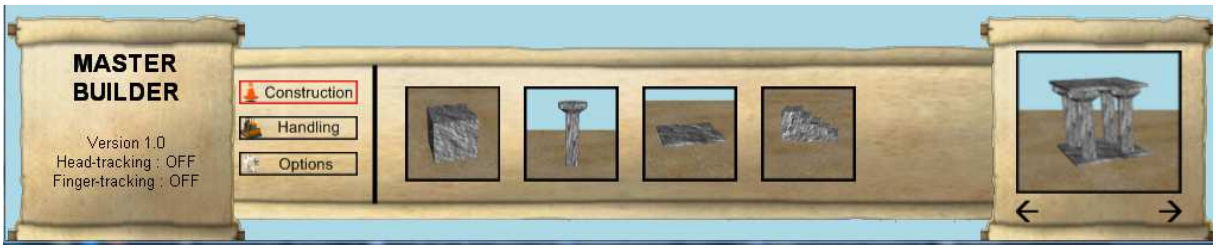

Figure III-36 : Menu of the game

4.2.1 Menu on the Left

Figure III-37 : Left menu

The left menu gives information to the user. There is no button on it, and it acts as a board that displays the main pieces of information that the user might want to know.

This information is the name of the program, as well as the number of the version. If head tracking or finger tracking is enabled, it is visible in this part of the menu. This is useful for the user to see quickly the most useful pieces of information.

4.2.2 Menu on the right

Figure III-38 : Right menu

The right part of the menu consists of one image. This image is an example of building that the user could try to imitate in the game. There are several “pattern” images which can be displayed in this box. The user can navigate from one to another by clicking on the arrows under the image.

This gives the user a specific goal, if he needs it. The user can then try to build a similar construction element as shown in the box. It can also give the user some other ideas.

4.2.3 Menu in the center

Figure III-39 : Center menu

The center menu itself is also divided in two parts. A small part on the left contains some rectangular buttons with labels. The biggest part, on the right, contains some squared buttons with icons.

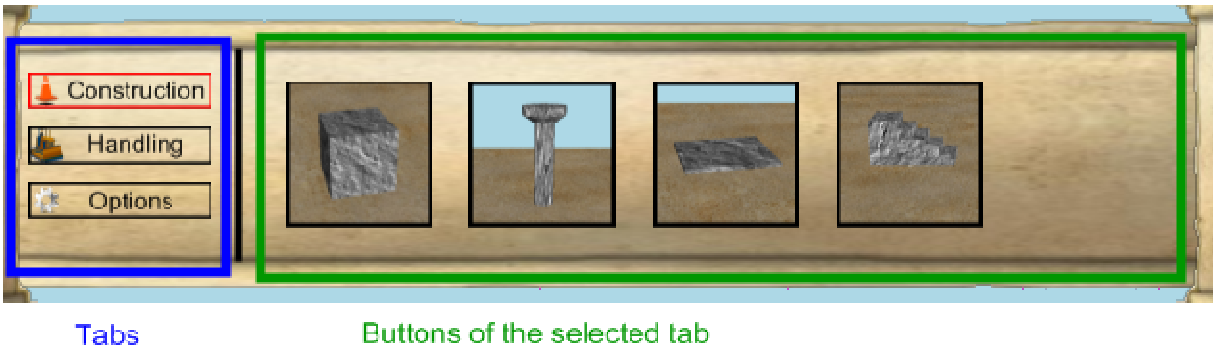

Figure III-40 : The two parts of the center menu

The left part is the tabs part. Each button of the left part is related to a tab. The buttons of the right part are the buttons contained by the selected tab. The selected tab has a red rectangle instead of a black one, which enables the user to know in which part of the menu he is.

In the previous image, the selected tab is the “Construction” tab. This tab contains all the tools that will allow the user to add new construction elements to the scene. In order to do so, the user has to select the button corresponding to the object he wants to add, and then drag and drop the object somewhere in the scene. The detailed description of this manipulation using the finger tracking is described in a next part.

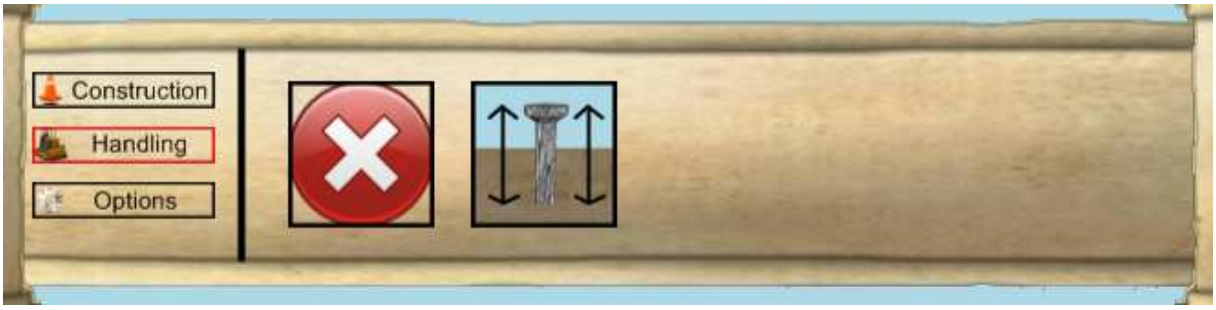

Figure III-41 : "Handling" tab

The “Handling” tab proposes a set of actions that can be performed on the currently selected object. This set of actions varies depending on the type of the selected object. A good example of possible actions on a “Pillar” object is the resizing. After selecting the resize button (the second one on the previous figure), the user can change the height of the “Pillar” object. An example of common action for all objects is the removal. After clicking on this button, the selected object is removed from the scene.

Figure III-42 : "Options" tab

The “Options” tab contains the settings of the application, as well as some actions which are not directly related to the scene. The first icon allows the user to set the “selection time”. The role of this parameter is explained in the Parmaters settings section. The second icon enables the user to activate or deactivate the head tracking. The last button is the “Exit” button.

Now that the scene and the menu have been described, we will explain how the user can select items or objects using his fingers.

4.3 Finger-tracking interaction in the game

In this part of the report, the interactions, using fingers tracking, between the user and the game are presented. The software has been designed to function with keyboard and mouse as well, but there are more options available when the user interacts with finger tracking. The whole project being focused on the finger tracking and head tracking, the case where the user does not use either is not discussed in this report.

4.3.1 Events generated by the finger-tracking

As described in a previous part, the user wears gloves equipped with an infrared diode and a switch. To interact with the game, the user must press the switch button, which turns on the diode. The program then gets the position of this diode.

The motion-tracking dll analyses the information received by the camera, and generates events accordingly. There are two events which are caught by the game. When at least one diode is detected, an “Update diode position” event is sent, to give the position of one or two diodes. When one of the diode is lost (is not detected anymore), another event is sent, “Lost diode”.

Based on those two events, the 3D game manages the fingers tracking interaction, by analyzing the position of the diodes. The choices made for this interaction are described in the following section.

4.3.2 Selection of a menu icon or an object

The main and first issue we had concerning the fingers tracking is the selection of an item, in the menu or in the scene. When using a mouse, it is easy to select an item: the user moves the cursor onto the object and presses the left button. However, when using the finger-tracking gloves, it is not possible to do exactly the same. The program knows the position of the diode only when the user presses the switch button. That means it is unable to determine the position of the cursor if the user is not pressing the switch button. An imitation of the mouse pointing system was thus impossible.

The cursor is hidden whenever the user is not pressing the switch button. This is due to the fact that the program cannot possibly guess where the user is pointing with his finger. As a matter of fact, the user presses the button to make the cursor appear somewhere on the screen, depending on where he is pointing.

However, the click, or selection of menu is still not defined. Several options were analyzed for this part.

- **The user selects an item by placing the cursor on the object.**
  Advantages: This is the simplest way to select an object with the glove.
  Drawbacks: It has a major drawback. When the user is moving the cursor on the screen, and accidently move over an object or a menu item, it will immediately be selected. This makes the selection very stressful for the user, since he must be very careful not to move the cursor over any kind of item that could be selected.

- **The user selects an item by placing the cursor on this item and waiting a certain time.**
  Advantages: It is a quite simple way to select an item. If the user accidently moves the cursor on the wrong item, he has time to move it away, so that the item is not selected. Moreover, it is sometimes impossible to move to an item without moving over other items around. In this case, the user can move fast enough to avoid the selection of the wrong items, and wait over the right one for the selection. Finally, the waiting time can be a parameter of the application that the user could change.
  Drawbacks: The only drawback is the time the user has to wait to select an object. It is impossible to select immediately an item. However, this time can be very short (500 milliseconds for example).

- **The user uses one glove to target an item, and presses the other glove switch button to validate the selection.**
  Advantages: With this method, there is strictly no possibility of mistakes. The user clearly sees which item is currently targeted, and can press immediately the switch button of the other glove.
  Drawbacks: The fact that the user must manage two gloves for a simple selection is the first drawback. It means the user must be able to press the switch button at the right times, just to select an item or an object. This makes it a bit too complicated for the user, who is used to the mouse as a pointing device. Also, it is not as intuitive as the other ways described above.

4.3.3 Description of the selection process

We chose the second option. The user selects an item by placing the cursor on this item and by waiting a set time. This time is a parameter that can be set in the “options” menu of the program, allowing the user to adapt the selection process to his needs and preferences.

In order to make this selection intuitive for the user, we designed several types of cursor images. Depending on the actions of the user, the cursor changes color or shape, in order to guide the user in the selection process.

1. The user presses the glove switch button and maintains it pressed. The cursor appears on the screen, its position depending on the position of the user’s finger. If the cursor is not over any selectable item, it appears in a dark red color. This color signifies that there is nothing to select at the cursor current position.
2. When the user moves his finger, the position of the cursor changes accordingly. If the user places the cursor over a selectable item, the cursor color changes to purple. Purple being flashier than dark red, the user directly sees that something is happening. The item is being selected. If the user wants to select the object, he has to keep the cursor over it, and the selection process continues to step 4. If the user does not want to select the item, he will move the cursor away from it, and the process goes to step 3.
3. The user moves the cursor away from the item. The cursor immediately changes back to dark red. The item is not selected.
4. The user keeps the cursor over the item. The cursor stays purple for the defined waiting time, and then changes color and shape. The new cursor image depends on the type of item selected. The item is selected.

The selection process has been analyzed, and this report will now focus on the actions that can be performed in the 3D game, using the finger tracking.

4.3.4 Creating an object

As described in the “Interface Menu” part of this report, the user must use the “construction” tab of the menu to add a new object on the scene.

1. To add a new object, the user must first select one of the object icons in the construction menu. Once selected, a red square circles the icon, showing the user that the icon is selected.

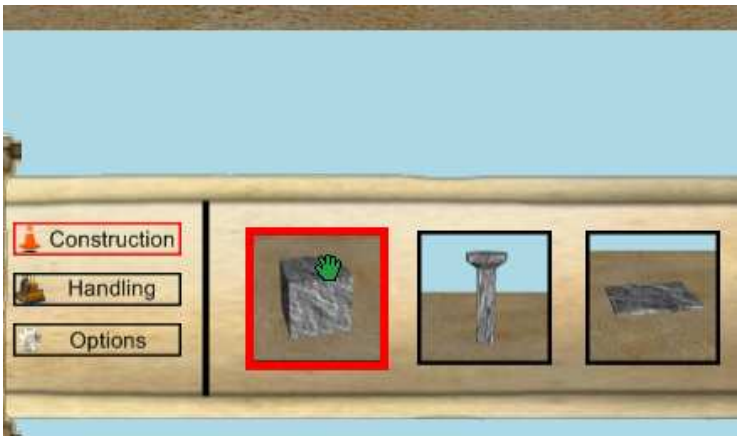

Figure III-43 : Step 1 of the creation of a new object

2. The user must then keep the switch button pressed, and move the cursor onto the scene. When the cursor exits the menu, it is replaced by a rectangular parallelepiped that symbolizes the place where the new object will appear.

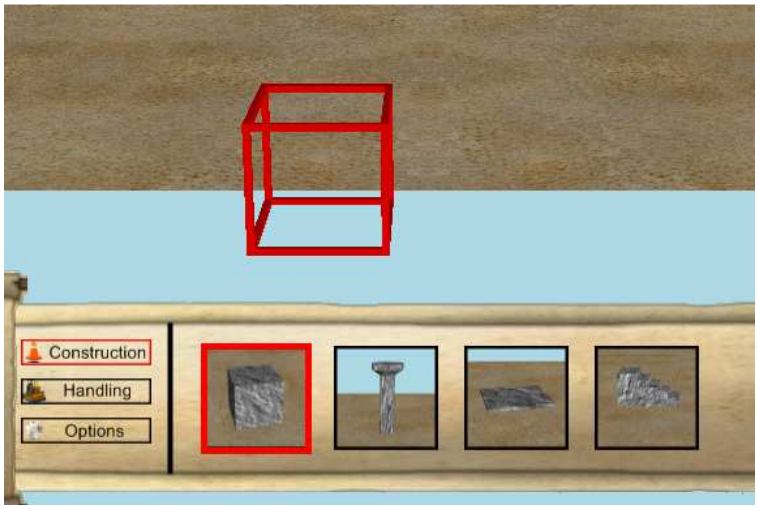

Figure III-44 : Red Parallelepiped because it is outside the borders of the scene.

3. The user can then move this parallelepiped in the scene. The color of the parallelepiped is an indication whether it is possible to create an object at this place. If it is green, it means the creation is possible, and if it is red, it is not possible. To create the object, the user must release the switch button. If the parallelepiped is green when the user releases the button, the process goes to step 4. If it is red, then it goes to step 5.

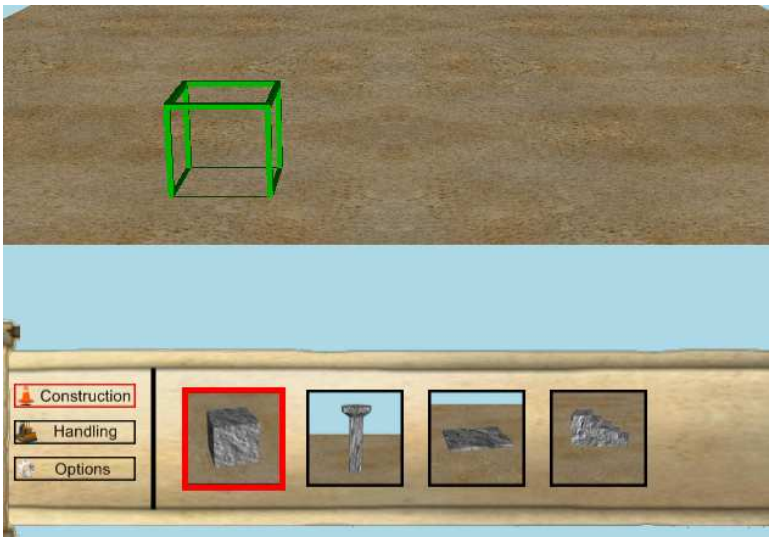

Figure III-45 : Green parallelepiped

4. The user releases the switch button when the parallelepiped is green. The parallelepiped is replaced by the new object. The creation process is finished.

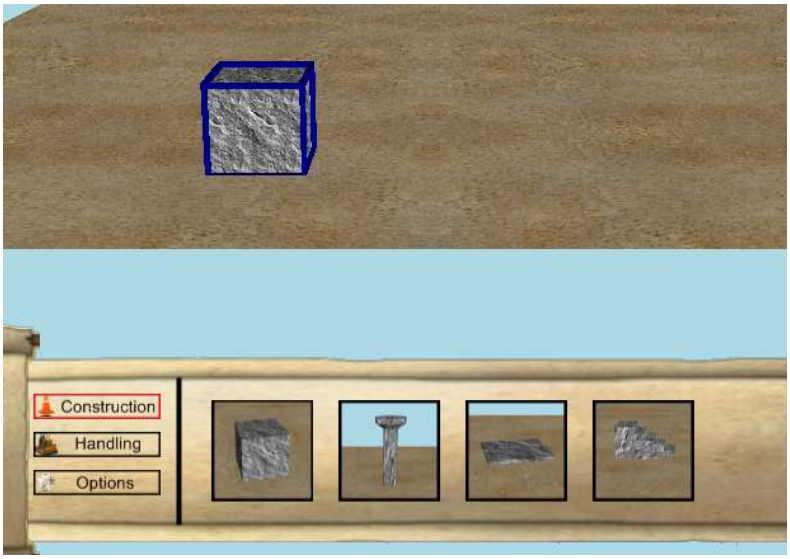

Figure III-46 : A new object has been added to the scene.

5. The user releases the switch button when the parallelepiped is red. The creation of the object being impossible, the parallelepiped disappear and the process is finished, without any new object added to the scene.

This process resembles a mouse “drag and drop” motion. That makes it quite intuitive for the user, and easy to understand. The colors clearly help the user in the choice of the place where the new object will be put.

4.3.5 Moving an object

As previously described, the user can choose the place of creation of a new object. However, it would be hard to build something if the objects could not be moved afterwards. That is why we designed a

way to move objects. It is executed through two main parts. The first part is to grab hold of the object. It is done by selecting the object, through the selection process. When the object is selected, the user must keep the switch button pressed and move his finger.

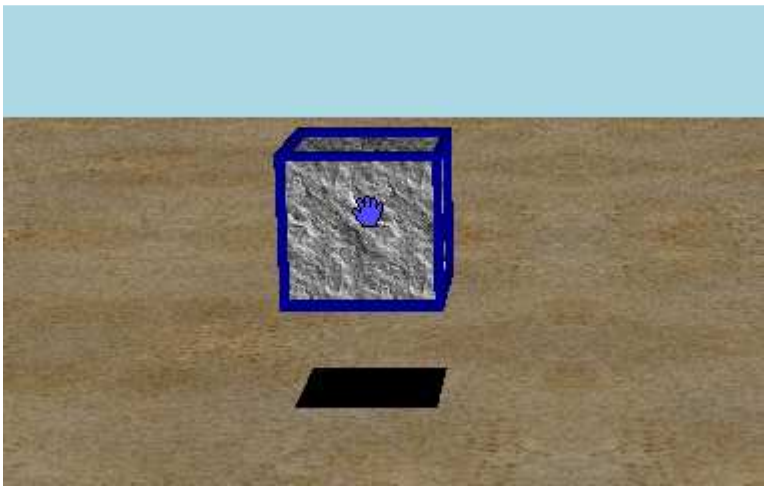

Figure III-47 : The cursor is blue because the object is hold by the user.

The scene is a 3D environment, which means that objects can be moved along three axes. However, the main issue we faced was that the finger tracking only works in 2D. The camera cannot "see" the depth, so the user can only move his finger up and down, as well as left and right. It gives him two degrees of freedom[2], when three degrees are needed.

To solve this problem, several options were analyzed. In order to simplify the following explanations, we will name the axes in the game: X-axis is the left-right axis from the user's point of view, Y-axis is the up-down axis, and Z-axis is the depth.

- The user can change the X and Y coordinates with the first glove, and the Z coordinate with the second glove. Moving left and right with the first glove causes the object to move along the X-axis. Moving up and down with the first glove causes the object to move along the Y-axis. Moving up and down with the second glove causes the object to move along the Z-axis.
  Advantages: It is not too hard to understand.
  Drawbacks: It is not a very intuitive way of moving the object, and it requires time to master the technique.

- The user could use the two gloves in a similar way, but changing X and Z coordinates with the first glove and Y coordinate with the second glove.
  Advantages: It is not too hard to understand.
  Drawbacks: It does not seem easy to master the object movement.

[2] Degree of Freedom: "*Any of the minimum number of coordinates required to specify completely the motion of a mechanical system*" [71] .

- The user can move the object along X and Y axes with the first glove, and use a specific motion to move along the Z-axis. To move along the Z-axis, the user can press both switch buttons and move his hands further apart or closer. Moving the hands further apart would cause the object to go further from the user, and an opposite motion would cause the object to move closer.
  Advantages: It seems to us an intuitive way of moving the object along the Z-axis.
  Drawbacks: It is hard to understand. When moving the two gloves, it is impossible to move the object along the X and Y axes, only along the Z axis.

We chose the third option for the implementation of the program, but we think that the best way to determine which option to take is to perform some user tests. This report will explain the tests and improvement we made in the Tests part.

4.3.6 Handling of an object

After creating and moving objects, the user might want to make some other actions on them. As described previously, the actions that can be performed on objects are gathered in the "Handling" tab of the menu. The object currently selected, on which the handling actions are performed, appears with a blue rectangular parallelepiped around it.

4.3.7 Deleting an object

One action that can be performed on all types of object is the removal. The icon is a red circle with a white cross in the middle. When the user selects this action, the object is removed, and disappears from the scene. The delete icon goes black and white, and all other handling actions are hidden, indicating that there is no object currently selected.

Figure III-48 : Delete icon

Figure III-49 : Delete icon

4.3.8 Resizing an object

Some objects, like pillars, allow changes in their size. In a case of a pillar, it is only possible to change the height of it, meaning its size along the Y-axis.

Figure III-50 : resizing icon

To do so, the user selects the pillar which size he wants to change, and then open the "handling" tab of the menu. In this "handling" tab, he will find the resize icon, which he must select. When the resize icon is selected, the user must place the other glove above the first glove, and then move the second glove up and down. When moving the second glove down, the height of the pillar is reduced (as the distance between the two gloves is reduced). When moving the second glove up, the size will be made bigger.

4.3.9 Parameters settings

The user can also change some parameters in the “options” tab. For example, it is possible to set the “selection time”, which is the time the user has to wait for the program to select an object. It is possible to change this setting by selecting the corresponding icon, and by using the two fingers and moving further apart or closer. This changes the value of the parameter, which is displayed on a pop-up in the middle of the screen.

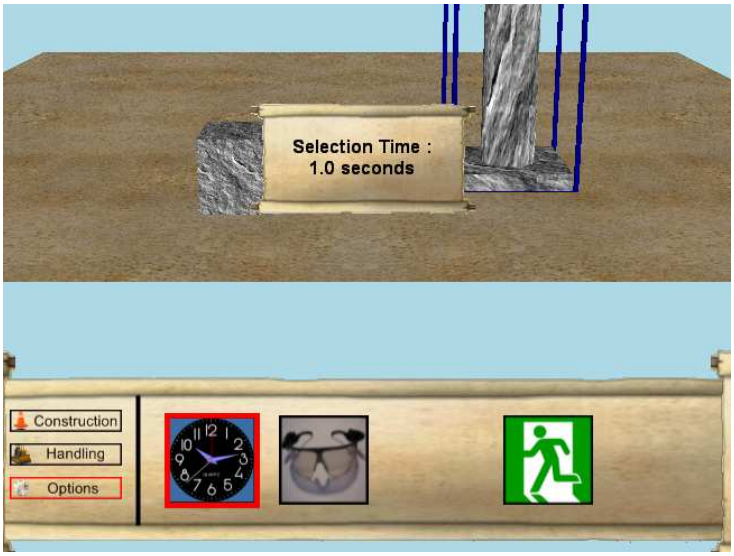

Figure III-51 : The user is changing the "selection time" value.

4.3.10 Exiting the application

When the user selects the “Options” tab, he can select the exit button. In order to avoid any mistakes, a pop-up asks the user if he really wants to exit the program.

Figure III-52 : Exit pop-up

This part has analyzed the possibilities of fingers tracking, and how the program is using them. The next part of this report will be focused on the second HID used with this program: the head tracking.

4.4 Head-tracking interaction in the game

In this part, the head-tracking interaction in the game will be described, from the information sent by the wiimote to the effect shown on the screen.

4.4.1 Events generated by the head-tracking

The wiimotes get the position of the frames and this information is stored in a “wiimote” object, accessible by the program. The program then analyzes the data of this event, which is the position of the frames. The purpose of the application is to determine the position of the user in the room, and to change parts of the game accordingly.

4.4.2 Purpose of head-tracking for the 3D game

The purpose of the head tracking is to make the 3D environment seem real, by moving the position of the point of view according to the position of the head of the user. In order to move the point of view, the DirectX camera has to be moved. The position of the DirectX camera and its angle determine how the 3D objects are seen on the screen. The movement of the camera is determined by the movement of the user’s head.

4.4.3 Interpretation of a movement to the left or right of the room

When the user moves to the left or to the right, the DirectX camera moves along the X-axis. However, its angle will change, since it must keep looking at the same point in the scene. This will cause the user to look at the same objects, but with a different angle, exactly as if the objects were real.

4.4.4 Interpretation of a movement up or down

For this kind of movement, the camera moves along the Y-axis. This causes the user to see the same objects with a lower angle or a bigger one, depending if he is crouching down or jumping.

4.4.5 Interpretation of a movement forward or backward

When the user moves toward the screen, the DirectX camera moves toward the scene, along the Z-axis. It causes the scene to appear bigger to the user. When the user moves backwards, the camera also moves along the Z-axis, going further from the scene. This is of course what happens in the real world.

Part IV Implementation

After the presentation of the system made in the Part III, the implementation part presents the development phase of the system. It describes the problems encountered during this phase and the improvements made to the previous design. The hardware components, the MotionTracking dll and the 3D environment realization are explained in the following parts.

1

Infrared Frames

As described in the Design part, the infrared frames were made removing the glasses from shades. A PR4401 based coupling diodes integrated circuit has then be installed on each side of the frames with its own battery. The two circuits have been fixed to the frames using two screws. The resulting product is strong and no malfunctions were experienced. The first tests revealed that the weight added to the frames (overall weight of the frames = 57g) is insignificant for the user's head and that the product is comfortable for the user as the screws do not touch his head.

To use these frames, the user simply has to turn on the two switches to light the diodes and to wear the frames.

Figure IV-1 : Infrared frames

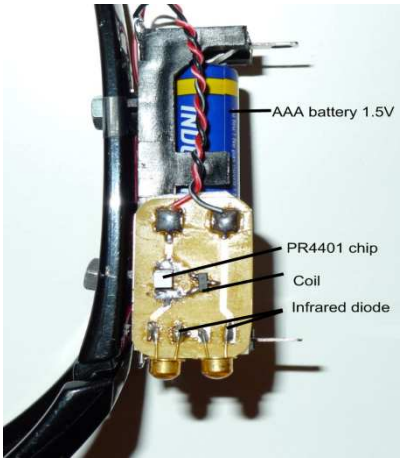

Figure IV-2 : Infrared frames left diodes circuit

2

Infrared Gloves

The infrared gloves were made using two gardening gloves. The resistor and the button switch were glued on the battery holder as showed in the Figure IV-3. To entrench the circuit, the infrared diodes and the powering wires have been sewed to the glove as presented in the Figure IV-5 . The overall weight of one glove is 30g, this low weight reduces the probability of the user's hand strain.

To use these gloves, the user has to put his fingers in the gloves and then put the battery holder under his palm. The switch button must be placed on the side of the glove (side down). To turn on the diode, the user simply presses on the switch by pressing with any finger (except the index which has to point the camera sensor) the switch.

During the realization of the glove, many problems were encountered with the soldering of the diode wires on the switch and battery terminal. A very small tension or torsion of the wire could break the welding and stop the diode electricity supply. To avoid this problem, the wires of each glove were rolled around the battery holder and fixed to the holder using retractable plastic. This improvement limits the strain on the welding when the user puts the gloves. Since the realization of this improvement, no others issues were encountered with the battery holder welding.

Due to the low energy of the battery used (CR 2032 [62]); it was also decided to limit the battery consumption. The button switch turns on the diode only when the users presses it and turns it off as soon he releases the button.

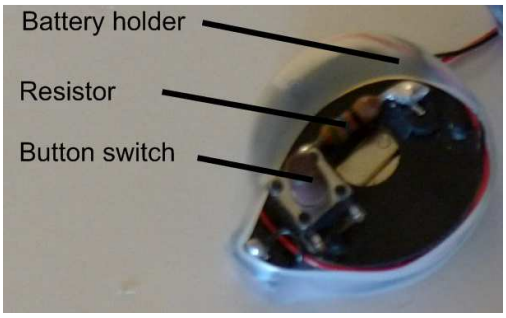

Figure IV-3 : Infrared glove diode circuit (side down)

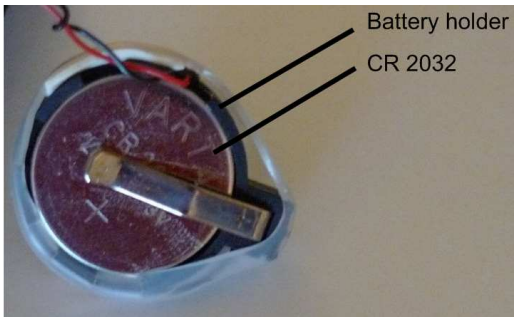

Figure IV-4 : Infrared glove right glove (side up)

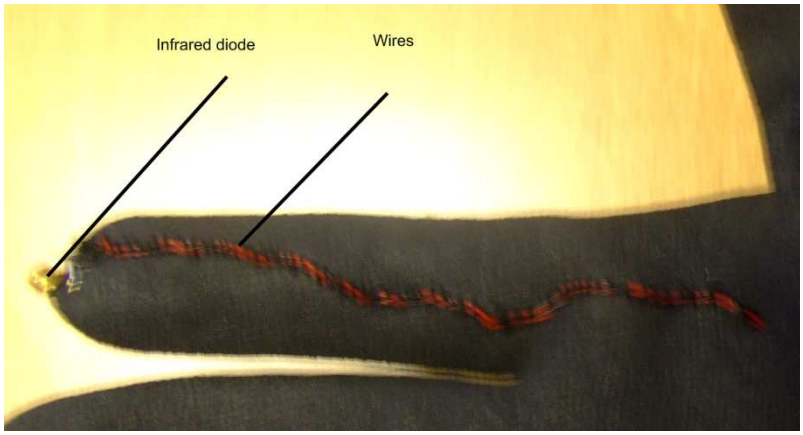

Figure IV-5 : Infrared diode and wires sewed to the glove

3

MotionTracking DLL

As planned in the Analysis and Design parts, the MotionTracking dll has been developed using C#.net and OpenCV methods with Microsoft Visual Studio .Net 2005 **Integrated Development Environment**. The .net language allowed to compile a library available for any .net managed application (C++/C#/VB) and fully compatible with the latest Microsoft Windows Operating Systems NT5.x and NT6.x.

Tested on a Intel dual core 2.4Ghz embedded computer with Microsoft Windows XP, it uses 20% of the processing time of one core and requires at least 35 megabytes of random access memory (mainly due to the image processing methods used such as canny edging). The dll implemented is entirely independent and can be improved without any modification of the 3D environment solution or reused in any other project. It is able to run any type of compatible OpenCv camera.

The MotionTracking solution is embedded with a complete installation handbook, HTML documentation and examples of the **Application Programming Interface** (API).

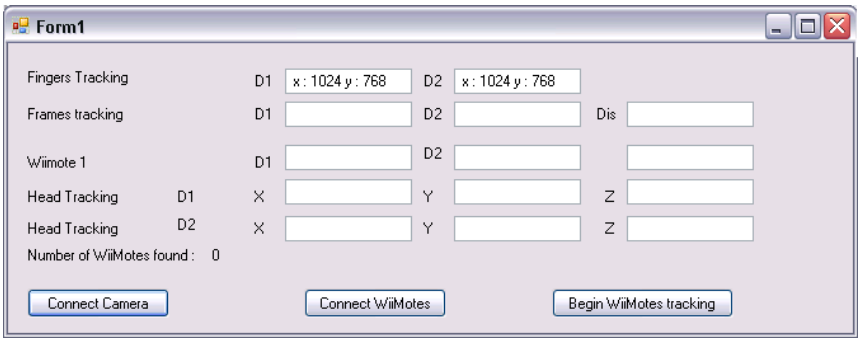

Figure IV-6 : MotionTracking dll integration example

The Figure IV-6 shows the intermediary application developed to test the head and the fingers tracking during the implementation of the 3D environment. All the methods of the MotionTracking dll were tested using this application before the integration in the 3D environment. This test program is able to connect to the wiimotes without any action from the user on the computer, using a C# library to handle automatically the Bluetooth devices [63].

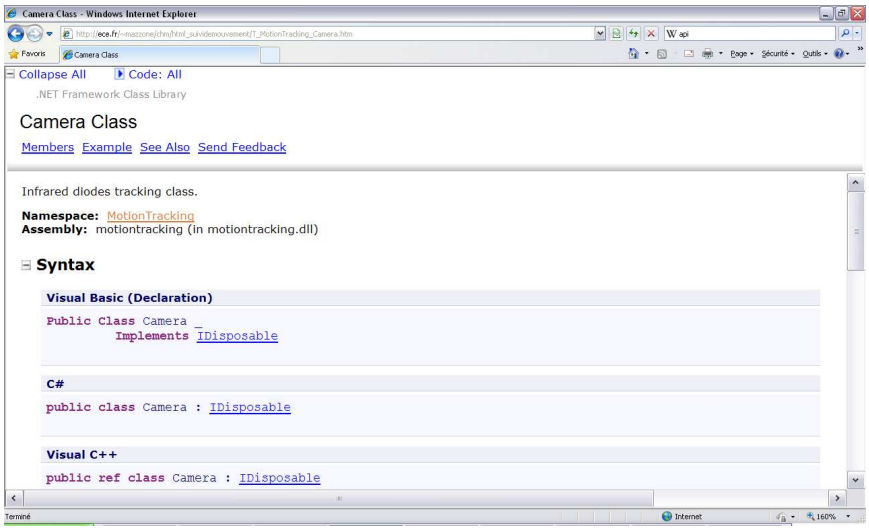

Figure IV-7 : Online MotionTracking API documentation

Remarks: The online MotionTracking API documentation is available at this address: http://ece.fr/~mazzone/chm/html_suividemouvement/N_MotionTracking.htm.

4

Implementation of the 3D Environment

This part will deal with the development of the 3D software, the tools used for this development, the difficulties encountered and the resources requirements of the final version.

4.1 Software development

As stated in the Analysi part, the 3D environment was written in C#, using DirectX 9 libraries. In order to write the code, Visual Studio Express 2005 C# was used, as it is a reliable software development kit. After creating the project in Visual Studio, we added the DirectX libraries and the motion tracking DLL as references to be used in the program.

The textures were made from image files got on the internet, on websites which propose this kind of images [64][65][66]. Some images were also designed with Paint.NET or Photoshop.

4.2 Mesh file creation

A mesh file (a ".x" file) has to be designed with a 3D modeling solution. The most famous of those modeling software is "Autodesk 3ds Max" (formerly "3D Studio Max"). However, many others have similar purposes and interfaces.

We want to design simple objects as we intend to spend more time on the engine of the 3D environment, rather than on the graphics. The handling of objects is more important that the realism of the objects.

4.2.1 Choice of the 3D modeling solution

The list below contains the names of some among the most famous ones.

- Autodesk 3ds Max
- SoftImageXSI
- Houdini
- Lightwave
- Blender
- Maya
- TrueSpace

For financial reasons, we wanted to use a freeware to make the object modeling. Most modeling packages are expensive. However, freeware exists also for modeling.

Among the freeware, **Blender** is the most famous. It is well documented on the internet and has a lot of resources and answers available on forums. Tutorials can easily be found, which makes it easier to learn the basics. For all these reasons, Blender has been chosen to design the three dimensions objects.

4.2.2 Mesh creation

There were two possibilities to create a ".x" file from Blender. The first solution was to download a model on the internet. There are several model file database on the internet, some of them free. The second solution was to design the model ourselves, using Blender. We decided to create the model with Blender, since it allowed us to have objects that fitted in our game properly. It was then possible to design a set of objects that had some common points (such as the textures), which enhances the fact that they belong to the same application. Downloading objects from a website could have created a bad feeling due to the fact that the objects were too different.

4.3 Difficulties encountered

4.3.1 Difficulties due to the Wiimote

In the beginning, it was designed that the wiimotes would track the head diodes and the webcam the fingers; the 3D environment (Master Builder) solution connected itself to the wiimotes and retrieved the localization of the diodes using the Managed Library for Nintendo's Wiimote available on Codeplex website [48] . Using the 2D coordinates retrieved by the wiimotes interface, the 3D environment then called methods from the MotionTracking dll to convert the 2D positions in 3D localisations. However, the first implementation tests revealed that the wiimotes frequently lost the infrared diodes tracked before finding them again. We tried the wiimote detection with some other frames developed by a professor of the university: the result was worst. Using the 3D display was very disturbing for the user who could even feel dizzy after five minutes of experimentation.

On the other hand, the fingers tracking made by the webcam revealed an excellent accuracy despite its lower resolution sensor. The diodes are infrequently lost even if they are not boost as for the head tracking. As the fingers tracking algorithm was already detecting the frames (as explained in the Design part), it was decided to retrieve the frames and the fingers positions using only the camera. The 3D environment game engine now only registers to the MotionTracking events to retrieve both head and fingers localizations. This improvement enabled us to present to the users a head tracking system where the 3D environment projected changes continuously and gradually depending on the user's head movement.

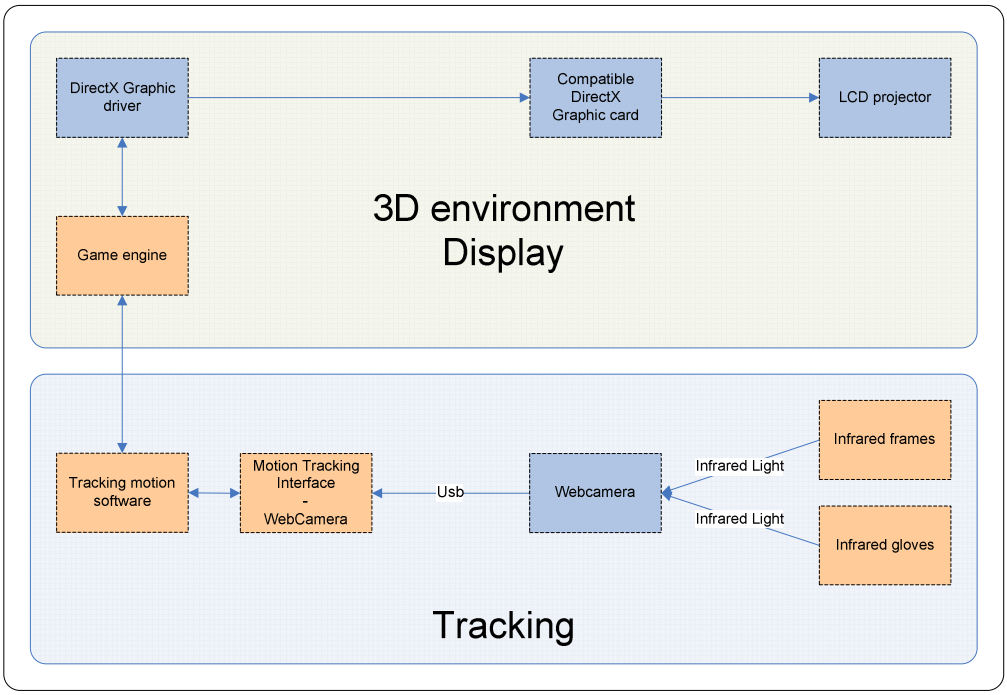

Figure IV-8 : Project structure updated

4.3.2 Confusions in the detection of the frames and fingers

Even with the camera as a single device for finger and head tracking, the program still encounters difficulties to distinguish the diodes of the frames and the diodes of the fingers. From the point of view of the camera, it is impossible to distinguish the lights coming from the frames and the lights coming from the fingers.

As explained in the Design part, some algorithms have been written to avoid confusion between the frames and the fingers. However, when the user place his finger or his fingers near his head, the distinction is hard to make, since all the diodes are in the same region, with similar distances between them.

This is the reason why sometimes the program reacts in an unexpected way, by making confusion between the frames and the fingers. This is one of the reasons why we decided to implement a mode with fingers tracking only, in order to test the program without confusion between the two types of motion tracking. When using fingers tracking only, the position of the cursor is always accurate as no frames disturb the detection.

4.4 Resource requirements

The 3D environment was tested on a Intel dual core 2.4Ghz embedded computer. The operating system used is Windows XP. When not using the fingers tracking or the head tracking, the program requires from 1% (when the user is inactive) to 10% (when the user moves some objects) of the processor. It also requires around 45 Mega bytes of memory. It can be noted that the number of objects created does not influence much the size of memory occupied by the program.

When using fingers tracking only, the program requires from 10% to 27% of the processor power. When the user is moving objects, the average processor power required is 20%. The program requires 60 Mega bytes of memory.

When using fingers tracking and head tracking, the program requires approximately the same processor power and memory space than when using only fingers tracking. This is due to the fact that they are not operating together at the same time.

After implementing the project solution, tests were run in order to measure the quality of the work done and to add relevant improvements.

Part V Tests

This part describes the testing process and its results. The analysis of those results led to some improvements that are also presented in this chapter.

1

Tests Preparation

This part of the report explains how the tests were performed, the tasks asked of the testers, and finally the list of questions asked during and after the test.

1.1 Three phases

First, we decided to divide the test sessions into three phases.

1. During the first phase, the user tests the fingers tracking only. Through a set of simple tasks, the user discovers the game without the head tracking. Some questions are asked during this phase.
2. After this first phase, the user has to put on the frames, and has to test the program with fingers tracking and head tracking simultaneously. The only task asked to the user at this point is to build a structure with the game.
3. For the last phase, the user puts off the gloves and frames, and he is asked some general questions about him and his thoughts on the program.

1.2 First phase tasks

Before the first task, the user is asked to put on the gloves. He is given a short explanation to understand how the gloves work. In particular, he is told that pressing the switch button will turn on the diode. He is also told that only one glove should be used for the first tasks. When the user is ready, he is asked to perform the tasks in this specific order:

1. The test-person has to add a new object to the scene. No new indications are given to him for this task. After 5 minutes, the manipulation is explained to the user if he did not find out by himself.
2. The test-person has to move the new object from one place to another, along the X and Y axes. The movement performed does not matter, and the test-person can place the object wherever he wants. No new indications are given to him during this task, and after 2 minutes the manipulation is explained to the user if he did not find out by himself.
3. The manipulation needed to move the object along the Z-axis is explained to the test-person. Since it is a more complex manipulation than the other ones, the user is not expected to find out by himself, and the finger movements are directly explained. The user has then to perform the manipulation himself, and move the object along the Z-axis. He has 2 minutes to do so.
4. The test-person is asked to place a pillar on the scene, and change its height. The manipulation needed is directly explained to him, and the user just needs to imitate it.

5. The test-person is asked to build something. He is free to build any kind of construction, and it does not have to be realistic. The main purpose of this task is to make the user perform all the actions he discovered during the first tasks. By performing them again, the test-person may have new comments. The test-person is also encouraged to try other actions which were not explained to him in the first tasks (like changing the settings). This task can last up to 20 minutes, but the test-person can stop whenever he wants to.

After each task has been performed, the test-person is asked some questions about it. He is asked if this action was easy to perform. He is also encouraged to give some comments on this specific task. We decided to ask those questions after each task, and not at the end of the test, because the test-person might forget the feeling they had when they first performed the task. It seemed more relevant to ask them just after they did the action, since they just experienced it.

1.3 Second Phase

For the second phase, the test-person is asked to put on the frames, and to create another building. The main purpose of this task is to test the head tracking. However, the task is also useful to test if the construction is as easy with the head tracking activated. The user is asked to move around the room, in order to observe the effect of head tracking on the game. This task lasts 10 minutes.

The user is then asked if he found that the head tracking added a new value to the game. He is also asked if the game was convenient to use with the head tracking. Finally, his comments are written down.

1.4 Questionnaire

The third phase of the test is the questionnaire. The test-person is asked a few questions. The questionnaire has been added at the end of this report.

2 Results

The tests were performed on 7 people, with different ages and background. This allowed us to get a quite accurate result, with different kinds of comments and points of view.

Since the questions were all open questions, it is impossible to write the answers in this report. However, it is possible to present the summary of those answers, and the main ideas which have been proposed during those tests.

2.1 First Phase

1. The first task was to add a new object to the scene. Most users found this first task very intuitive and easy to perform. None of the 7 test-persons found it difficult. They also all found out how to perform the task by themselves.
2. The second task was to move the object. As for the first task, all users found out how to do it. All users found the task very easy to perform.
3. Changing the depth of the object, using two gloves, was the third task. Only one test-person said the manipulation was quite convenient. Four people said it was really not convenient the first time, but they tried again and found out that they get used to it. Two test-persons found it not convenient, and did not get used to it after a few other trials. Only two people found this manipulation really intuitive. Most of the test-persons complained about the fact that, when changing the depth of an object, it is not possible to move it along X and Y axes simultaneously. Some test-persons said it could be stressful, since they had to plan carefully their move before changing the depth.
4. The fourth task was to change the height of a pillar. Four people said the manipulation was easy to perform. Two people had some issues with it, but got used to it after a few trials. One test-person found it hard and not convenient.
5. The fifth task was to continue to build something. Most users really enjoyed this task, and spent nearly 20 minutes trying all options. Five test-persons found the building quite easy, and said all features were interesting and intuitive. Two test-persons said that some features were hard to use, or not completely intuitive.

2.2 Second Phase

6. The sixth task was to test the head tracking. Six test-persons said the head tracking was working properly. One test-person said the camera lost the frames too often. Five test-persons thought the head tracking really added a good value to the game.

7. Exiting the application was the seventh task. All test-persons found this task easy to perform.

2.3 Third Phase

8. The age of the test-persons ranges from 11 to 55 years. One test-person was 11 years old, one 22, one 23, three 25 and one 55. This wide range of ages was useful to get feedback from different generations, used to different kind of technologies.
9. Five of the test-persons had already used a touch-screen (mostly public touch screens, but also Apple iPhone for some). Apart from touch-screens, none of the test-persons had ever used an application using fingers tracking.
10. One had already used head tracking in the “cave” of Aalborg University (electronic department). None of the others had ever used an application with head tracking.
11. All test-persons enjoyed playing the game. Some were very enthusiastic about it. They found it interesting, and appreciated the “creative” part of it. Most test-persons enjoyed the freedom in the game.
12. All users said that the fingers tracking and head tracking enhanced the immersion factor, but to different degrees. Four of them really thought this kind of technology would really make a difference for games, or architecture software. One test-person did not think it would be used in the future, and two test-persons thought it had to be improved a lot before it could really make a difference compared to the keyboard and mouse.
13. This last question was particularly useful, since most test-persons really tried to come up with some interesting improvements to the game. Some of them were unfortunately impossible to realize. For example, one test-person proposed to add some lines to show clearly the position of the object being selected. Another proposal was to make a difference of color between the selection state and the “holding” state (when the user holds the object). In our design, both states were represented by the blue color. Another idea was to use the two gloves as two cursors, instead of just one. In addition, several lights could be placed on the same glove, to increase the possibilities offered to the user.

3

Improvements

3.1 Analysis of the test results

After performing the tests, we gathered the results and analyzed them. The purpose of this was to identify which part of the project the test-persons liked, and which parts they thought needed improvements. We also analyzed the ideas and proposal made by the test-persons, and we determined which of them was possible and which of them was not. The test-persons being from different background, they gave some various and interesting ideas. Some ideas were very creative, and some were more technical.

3.2 Movement of the object along the three axes

The first analysis was the analysis of the tasks comments. The first two tasks, adding and moving an object along X and Y axes, were found easy to perform by nearly all test-persons. The test-persons that did not find it easy the first time got used to it afterwards. The same can be said about the fourth task: resizing the pillar. However, the third task was not found intuitive by the test-persons. They also found it strange that they could not change the position of the object on the three axes at the same time. As explained in the Design part, the position on the Z-axis can be modified only with two gloves, rendering it impossible for the user to change the position of the X and Y axes.

As a matter of fact, we decided to try a completely different way of moving objects. The first modification we made was to change the movement with one glove. During the test, the movement with one glove was along the X and Y axes. We changed it for a movement along X and Z. When the user moves his glove from left to right, it changes the position on the X-axis, and when the user moves his glove up and down, it changes the position on the Z-axis. This is actually as intuitive as the previous system.

If the user uses two gloves, the first glove will still have the same effect: it moves the object along the X and Z axes. This is very convenient for the user, who is not confused as some test-persons were in the tests. The second glove modifies the position of the object on the Y-axis. By moving the second finger up and down, the user moves the object up and down. It is thus possible to move the object along the three axes simultaneously.

This new method is more intuitive than the previous one, and it allows the user to feel less pressured, since it is possible to move the object on three axes at the same time: there is no need to plan the move in two sequences, as it was the case for the test version. It is important to note that this is a major change in the application, the movement of object being quite different. It is the main improvement made to the project after the tests.

3.3 Changes in the color code

Another interesting comments coming from the test-persons were that the difference between the “selected” state and the “held” state was not clear enough. A selected object is just an object on which the user can perform some special manipulations. An object that is held by the user is the object that the user is currently moving.

During the test, the program was showing those two states by showing a blue rectangular parallelepiped around the object being selected or held. There was no visual difference between those two states. In order to fix this, we change the blue color into purple for the “held” state.

3.4 Bugs correction

During the tests, we also discover some bugs, which were corrected after the testing sessions. The test-persons were trying manipulations in a way we never tested before, and that is why they uncovered some bugs.

Part VI Conclusion

First, this section summarizes the achievements of this project. Second, the future improvements of the tracking solution and of Master builder application are presented.

Project Achievement

During the project period, all the defined objectives have been studied and implemented in the final version.

Compare two cost-efficient vision-based motion-tracking devices: the wiimote and the webcam.

During the design and implementation period a complete comparison between a wiimote and a webcam has been made. It has been initially designed that the wiimotes would be used for the head tracking and the webcam for the fingers tracking. The wiimote has been manufactured to track infrared light whereas the webcam needed an adaptation to detect infrared light. A motion tracking solution Camera class has also been implemented to process the webcam pictures and find the fingers diodes positions. The two devices have then been used in the same conditions to track infrared diodes and unexpected results were obtained. With these few modifications, the webcam, infrequently loosing the user's fingers during tracking, turns out to offer high performance and potential to follow many more infrared dots than the maximum of four for the wiimote. The wiimote, despite its high resolution camera sensor and its infrared filter adapted to the diode frequency, obtained a so bad infrared frames detection range and lost so frequently the diodes lights during motions that ,after numerous improvement attempts, both the head and fingers tracking have finally been provided by the webcam tracking solution.

Several problems were also experienced during the connection of the wiimote to the computer via the Bluetooth connection and an automatic connection method has to be implemented. Another drawback of the wiimote is that few diodes detection parameters can be set to improve the tracking. However, the other main tracking component of the wiimote, the accelerometers-based motion tracking has not been tested and could prove very interesting to implement cheap tracking software in which the user holds the wiimote.

Implement a reliable and efficient motion tracking using wiimotes and a webcam.

The usability testing reveals that the fingers tracking is very reliable as the user has no problem to select and handle objects displayed by the 3D environment. The software is able to track two fingers simultaneously and to distinguish the left and the right hand allowing the utilization by both left- and right-handed users. The head tracking is a little less reliable and works in a smaller area than the fingers tracking. Nevertheless, it is fully functional and tracks efficiently the user's head when he is looking at the screen. These drawbacks can be explained with the short implementation time reserved to the head tracking using the webcam image processing after the abandonment of the wiimote tracking.

When using both fingers and head tracking the motion tracking reliability drops as the users put frequently their fingers near their head. The fingers and frames diodes localizations proximity then induce many frames detection errors.

The infrared gloves and frames implemented are reliable and only one glove had some technical issues during the testing. The energy consumption is reduced. The gloves are comfortable but revealed an issue during long tests: the user's hand becomes very hot after fifteen minutes and it can sweat. Using gloves with good ventilation is a good answer to this problem.

Experiment new means of object handling using fingers tracking to improve the use efficiency of an architecture modeling software.

From the results of the tests performed, this objective has been achieved. Most test-persons find the fingers tracking gloves very convenient to use, and very intuitive. The 3D environment developed for this project clearly shows that, for this kind of application at least, the fingers tracking is a very interesting and efficient Human Interface Device. The user can interact only by moving his fingers, and can appreciate the display on a projected image.

The manipulations that can be performed on the objects proved to be intuitive and efficient too. Some of them need to be tried several times before being mastered completely, but it seems normal since everybody needs some time to get used to a new device, and especially a new HID. The first design for the movement of objects was proved inconvenient during the tests, which allowed us to change it according to the feedbacks we got from those tests. The new manipulation is much more intuitive and convenient, and exploits the full capacity of fingers tracking with two gloves.

Finally, the project proves that the finger tracking really adds a value to the user's experience. The handling of objects with the hands offers a new dimension, which is nonexistent when the user is using the keyboard and the mouse. We believe that more and more applications will use fingers tracking or similar HID in the future.

Improve the immersion of the user in a three dimensions architecture modeling software using head-tracking.

The example we took for the head tracking was Johnny Chung Lee's video. In this video, Johnny Chung Lee presents a head tracking based on the wiimote. The program presented in the video works really well, and it gives the impression that the objects are standing behind the screen. We were unable to create the same atmosphere in our 3D environment, because it is much more complex than Johnny Chung Lee's program and because he uses unreal objects to distort the user's perception. The wiimote was not so efficient for our project, as stated above. However, the head tracking works well, and adds to the immersion in our game, allowing the user to see from a different angle, which can be very useful in some situations.

Future Improvements

In the future, various alternatives could be studied to improve the motion tracking efficiency. To date using both fingers and head tracking simultaneously is complex because the user cannot put his fingers near the frames. Two solutions have been mentioned:

- Currently four infrared lights are followed for the overall motion tracking. This limitation is due to the initial design choice to use a wiimote to track the head position as it can handle a maximum of four points. To improve the light intensity, it has also been decided during the design that the frames will have two coupled diodes at each side (four lighted diodes viewed as two points by the wiimote and the camera). Finally, only the webcam is used to track the diodes. As it does not require as much light intensity as the wiimote, the coupled diodes could be separated without any modification of the powering circuit and positioned to generate a specific shape of four points, such as a straight line, detectable by the motion tracking software.

- A more complex solution would be to implement another type of tracking such as accelerometers or inertial based tracking on the frames to avoid any interference between the head and fingers tracking.

Part VII References

1. Inventors of the Modern Computer. *Inventors.* [Online] 2008. [Cited: 04 02, 2009.] http://inventors.about.com/library/weekly/aa081898.htm.

2. **spideyivans.** PlayStation 3 headtracking. *YouTube.* [Online] 02 27, 2008. [Cited: 04 02, 2009.] http://www.youtube.com/watch?v=FFN8cW1mSVY.

3. **Woyach, Steve.** Immersion Through Video Games. *Illumin.* [Online] 2008. [Cited: 03 25, 2009.] http://illumin.usc.edu/article.php?articleID=103&page=1.

4. **Brown, Emily and Cairns, Paul.** A grounded investigation of game immersion. *Portal - Danish National Library Authority.* [Online] 2004. [Cited: 03 25, 2009.] http://delivery.acm.org/10.1145/990000/986048/p1297-brown.pdf?key1=986048&key2=9226897321&coll=GUIDE&dl=GUIDE&CFID=28198767&CFTOKEN=42669451.

5. **Immersive Factor.** Immersion Factor Part.1 – Definition. *Immersive gaming.* [Online] 2008. [Cited: 03 10, 2009.] http://immersive-gaming.org/?p=8.

6. **Immersive gaming.** Immersion factor. *Immersive Gaming.* [Online] 2008. [Cited: 03 10, 2009.] http://immersive-gaming.org/?p=8.

7. **Wikipedia.** Computer-aided design. *Wikipedia.* [Online] 2009. [Cited: 03 10, 2009.] http://en.wikipedia.org/wiki/Computer-aided_design.

8. **Cisarano, Jason.** GDC Expo Tech: Increasing the Immersion Factor. *Gaming Target.* [Online] 2007. [Cited: 03 10, 2009.] http://www.gamingtarget.com/article.php?artid=6678.

9. **Games Alfresco.** Top 10 augmented reality demos that will revolutionize video games. *Games Alfresco.* [Online] 2008. [Cited: 03 10, 2009.] http://gamesalfresco.com/2008/03/03/top-10-augmented-reality-demos-that-will-revolutionize-video-games/.

10. Nintendo Wii Remotes + Classic Controller + WiiMote. *Nintendo Wii Remotes.* [Online] 2006. [Cited: 03 20, 2009.] http://nintendowiiremotes.com/.

11. **SoundImmersion.** Sound Immersion. *GPrime.* [Online] 2009. [Cited: 03 10, 2009.] http://gprime.net/flash.php/soundimmersion.

12. **Marksway Ltd.** Vuzix iWear AV920 3D headsets. *phonesreview.* [Online] 2009. [Cited: 03 20, 2009.] http://www.phonesreview.co.uk/2008/04/22/vuzix-iwear-av920-3d-headsets-coming-to-all-3uk-customers-19995/.

13. **ELITE.** The all-in-one simulator. *ELITE.* [Online] 2009. [Cited: 03 10, 2009.] http://www.flyelite.com/advanced-atd.php.

14. **Azure Computing, Inc.** THE ALL-IN-ONE SIMULATOR. *Elite Simulation Solutions.* [Online] 2009. [Cited: 03 22, 2009.] http://www.flyelite.com/advanced-atd.php.

15. **Wauthier, Fabian.** Motion Tracking. *Motion Tracking.* [Online] 07 24, 2007. [Cited: 03 24, 2009.] http://www.anc.ed.ac.uk/demos/tracker/.

16. **C. Jia, H. Lu and R. Zhang.** Aggressive motion detection based on normalised Radon transform and online AdaBoost. *IEEE.* [Online] 02 26, 2009. [Cited: 03 24, 2009.] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4796351&isnumber=4796338.

17. **Hager, Markus vincze & gregory D.** *Robust Vision for Vision-Based Control of Motion.* New York : IEE Press, 1996.

18. **Wassner, Hubert.** Eye tracking ... la V2. *Esiea.* [Online] 2008. [Cited: 03 02, 2009.] http://professeurs.esiea.fr/wassner/?2008/05/16/139-eye-tracking-la-v2.

19. **Electronics Manufacturers.** Motion detector. *electronics-manufacturers.* [Online] [Cited: March 17, 2009.] http://www.electronics-manufacturers.com/products/security-equipment/motion-detector/.

20. **OpenCVWiki.** Open Computer Vision Library . *SourceFORGE.net.* [Online] 01 21, 2009. [Cited: 02 25, 2009.] http://sourceforge.net/projects/opencvlibrary/.

21. **Natural Point.** TrackIr. *Natural Point.* [Online] 2008. [Cited: March 15, 2009.] http://www.naturalpoint.com/trackir/.

22. **Westwood, James D.** *Medecine meets Virtual Reality 2001.* IOS Press,US : s.n., 1998. p. 131.

23. **Collin.** IR LED Glasses. *Narbotic.* [Online] 01 18, 2008. [Cited: 02 18, 2009.] http://narbotic.net/?p=131.

24. **Wall, Jason.** TrackIR 4 Pro Point-of-View Headset. *[H] Consumer.* [Online] 07 11, 2007. [Cited: 02 24, 2009.] http://www.hardocp.com/article.html?art=MTM2MywyLCxoY29uc3VtZXI=.

25. **Dorfmüller-Ulhaas, Klaus et Schmalstieg, Dieter.** Finger tracking for interaction in augmented environments. *Interactive Media Systems Group,Vienna University of Technology.* [En ligne] [Citation : 20 04 2009.] http://www.ims.tuwien.ac.at/media/documents/publications/FingerTracker.pdf.

26. **L, Vlaming.** University of Groningen - Bibliotheek. *Human interfaces - Finger Tracking Applications.* [Online] 08 04, 2008. [Cited: 04 23, 2009.] http://scripties.fwn.eldoc.ub.rug.nl/FILES/scripties/Informatica/Bachelor/2008/Vlaming.L./thesistex.pdf.

27. **ESA.** Hand Posture Analyser (HPA). [Online] 2003. [Cited: 04 20, 2009.] http://www.spaceflight.esa.int/users/index.cfm?act=default.page&level=11&page=iss01-sme-05.

28. **Pamplona, Vitor F., et al.** Vitor Pamplona . *The Image-Based Data Glove.* [Online] 2008. [Cited: 04 16, 2009.] http://vitorpamplona.com/deps/papers/2008_SVR_IBDG.pdf.

29. **Hillebrand, Gerrit, et al.** Advanced Realtime Tracking GmbH. *Technische Universität München.* [Online] 08 16, 2006. [Cited: 04 22, 2009.] http://campar.in.tum.de/pub/hillebrand2005fingertracking/hillebrand2005fingertracking.pdf.

30. **XORSYST.** Wii-mote Head Tracking. *XORSYST.* [Online] 04 21, 2008. [Cited: 02 23, 2009.] http://xorsyst.com/wp-content/uploads/2008/04/desktop-vr-display-wii-1.jpg.

31. **Natural Point.** TrackIR. *TrackIR.* [Online] 2009. [Cited: 03 10, 2009.] http://www.naturalpoint.com/trackir/.

32. —. TrackIR3 image. *TrackIR.* [Online] 2007. [Cited: 03 12, 2009.] http://nic.mm2c.com/sander/trackir/TrackIR0.jpg.

33. **Wikipedia.** OpenGL. *Wikipedia.* [Online] 2009. [Cited: 03 20, 2009.] http://en.wikipedia.org/wiki/OpenGL.

34. **OpenGL.** OpenGL - official website. *OpenGL.* [Online] 2009. [Cited: 03 20, 2009.] http://www.opengl.org/.

35. **Wikipedia.** DirectX. *Wikipedia.* [Online] 2009. [Cited: 03 20, 2009.] http://en.wikipedia.org/wiki/DirectX.

36. **Microsoft.** DirectX: Advanced Graphics on Windows. *MSDN.* [Online] 2009. [Cited: 03 20, 2009.] http://msdn.microsoft.com/en-us/directx/default.aspx.

37. **Mungler.** Just what is DirectX? *Digital Silence.* [Online] 2004. [Cited: 03 20, 2009.] http://www.d-silence.com/feature.php?id=254.

38. **GameDev.** 3D Engines. *GameDev.org.* [Online] 2007. [Cited: 03 20, 2009.] http://www.gamedev.org/wiki/index.php/3D_Engines.

39. **3DEngines.** 3D Engines. *3DEngines.de.* [Online] 2009. [Cited: 03 20, 2009.] http://www.3dengines.de/.

40. **Wikipedia.** Game Engine. *Wikipedia.* [Online] 2009. [Cited: 03 20, 2009.] http://en.wikipedia.org/wiki/Game_engine.

41. **OGRE.** Ogre presentation. *OGRE.* [Online] 2009. [Cited: 03 20, 2009.] http://www.ogre3d.org/.

42. **Crystal Space.** Crystal Space presentation. *Crystal Space.* [Online] 2009. [Cited: 03 20, 2009.] http://www.crystalspace3d.org/main/Main_Page.

43. **Irrlicht.** Irrlicht presentation. *Irrlicht.* [Online] 2009. [Cited: 03 20, 2009.] http://irrlicht.sourceforge.net/.

44. **Panda3D.** Panda3D presentation. *Panda3D.* [Online] 2009. [Cited: 03 20, 2009.] http://www.panda3d.org/.

45. **Wikipedia.** Comparison of OpenGL and Direct3D. *Wikipedia.* [Online] 2009. [Cited: 03 20, 2009.] http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D.

46. —. OpenGL. *Wikipedia.* [Online] 2009. [Cited: 03 20, 2009.] http://en.wikipedia.org/wiki/OpenGL#OpenGL_support_libraries.

47. **OpenGL.** GLUT and OpenGL Utility Libraries. *OpenGL.* [Online] 2009. [Cited: 03 20, 2009.] http://www.opengl.org/resources/libraries/.

48. **CodePlex.** Managed Library for Nintendo's Wiimote . *CodePlex.* [Online] http://www.codeplex.com/WiimoteLib, 01 19, 2009. [Cited: 04 05, 2009.]

49. **WiiMote Wiki.** Wiimote Project Wiki. [Online] 2009. [Cited: 05 08, 2009.] http://wiki.wiimoteproject.com/Main_Page.

50. **AlaTest.** AlaTest. [Online] 2009. [Cited: 05 08, 2009.] http://alatest.fr/popup/pic_gallery/54954311/?thumb=4%2F7%2FNINTENDO-WIIMOTE-WII-TELECOMMANDE-BLANCHE-0.jpg.

51. **Siemens.** LD 242-3. *AAU.* [Online] Siemens, 2009. [Cited: 03 01, 2009.] http://komponenten.es.aau.dk/fileadmin/komponenten/Data_Sheet/Opto/LD242.pdf.

52. —. LD 274-3. *AAU.* [Online] 2009. [Cited: 03 23, 2009.] http://komponenten.es.aau.dk/fileadmin/komponenten/Data_Sheet/Opto/LD274.pdf.

53. —. SFH 487-2. *AAU.* [Online] 2009. [Cited: 03 23, 2009.] http://komponenten.es.aau.dk/fileadmin/komponenten/Data_Sheet/Opto/SFH487.pdf.

54. **Microchip.** PICkit 2 Development Programmer/Debugger . [Online] 2009. [Cited: 05 06, 2009.] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805.

55. —. PIC16F882/883/884/886/887 Data Sheet. *Microchip.* [Online] 04 14, 2009. [Cited: 05 01, 2009.] http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en026561.

56. **PREMA Semiconductor.** LED DRIVER PR4401/PR4402. *Analog and Mixed-Signal ASICs.* [Online] 2008. [Cited: 05 23, 2009.] http://www.prema.com/pdf/pr4401.pdf.

57. **Burkhard Kainta.** PR4401 LED Driver. *elektor electronics.* 2007, 351.

58. **instructables.** Take Infrared Pictures With Your Digital Camera. *instructables.* [Online] 2009. [Cited: 05 10, 2009.] http://www.instructables.com/id/Take-Infrared-Pictures-With-Your-Digital-Camera/.

59. **Microsoft.** .NET Framework Developer Center. *msdn.* [Online] Microsoft, 2009. [Cited: 05 14, 2009.] http://msdn.microsoft.com/en-us/netframework/default.aspx.

60. **EmguCV.** Main Page. *EmguCV.* [Online] 2009. [Cited: 05 10, 2009.] http://www.emgu.com/wiki/index.php/Main_Page.

61. **Sastry, S. Shankar.** Canny edge detector demos . *Professor S. Shankar Sastry .* [Online] 2006. [Cited: 05 20, 2009.] http://robotics.eecs.berkeley.edu/~sastry/ee20/cademo.html.

62. **Varta.** Varta - CR2032. *Farnell.* [Online] 2009. [Cited: 04 29, 2009.] http://www.varta-microbattery.com/en/MB_DATA/DOCUMENTS/DATA_SHEETS/DS6032.PDF.

63. **Mazzone, Florent.** Gestion périphériques bluetooth . *Site de Florent Mazzone.* [Online] 02 01, 2009. [Cited: 02 15, 2009.] http://ece.fr/~mazzone/gestionbluetooth.html.

64. **Icon Archive.** *Icon Archive.* [Online] 2009. http://www.iconarchive.com/.

65. **alphonse-kun.** Tutorial Photoshop. *AidoForum.* [Online] 2007. http://www.aidoforum.com/tutoriaux-509-texture-pierre.html.

66. **IconDB.** IconDB. *IconDB.* [Online] 2009. http://www.icodb.com/.

67. [Online] http://www.mpi-inf.mpg.de/~sanders/courses/algen04/willhalm.pdf.

68. [Online] http://www.esc.auckland.ac.nz/Organisations/ORSNZ/conf36/papers/Engineer.pdf.

69. **Microsoft.** Bluetooth Functions. *MSDN.* [Online] 04 02, 2009. [Cited: 02 10, 2009.] http://msdn.microsoft.com/en-us/library/aa362927(VS.85).aspx.

70. **Cairns, Paul, et al.** Quantifying the experience of immersion in games. *Cairns Immersion.* [Online] 2007. [Cited: 03 10, 2009.] http://www-users.cs.york.ac.uk/~pcairns/papers/Cairns_Immersion06.pdf.

Part VIII Appendixes

Questionnaire

Phase 1

Add a new object to the scene.

1. How easy this manipulation was?

Move the object to the left, to the right, up and down.

2. How easy this manipulation was?

Change the depth of the object.

3. How easy this manipulation was?

Change the height of the pillar.

4. How easy this manipulation was?

Build something.

5. What are you comments on the different manipulation you had to perform?

Phase 2

Build something and move around in the room.

6. Do you think the head-tracking add some value to the game?

Exit the application.

7. How easy this manipulation was?

Phase 3

8. How old are you?

9. Have you ever used a device with a touch screen or using finger-tracking before?

10. Have you ever used a device using head-tracking before?

11. Did you enjoy playing the game?

12. How much do you think finger-tracking and head-tracking enhance the immersion factor of this kind of program?

13. Do you have any comments or ideas of improvement for the project?