



**DANILO DUARTE DE
SOUZA**

**PLATAFORMA PARA A CONFIGURAÇÃO DE
AMBIENTES VIRTUAIS INTERATIVOS**

**PLATFORM FOR SETTING UP INTERACTIVE
VIRTUAL ENVIRONMENTS**



**DANILO DUARTE DE
SOUZA**

**PLATAFORMA PARA A CONFIGURAÇÃO DE
AMBIENTES VIRTUAIS INTERATIVOS**

**PLATFORM FOR SETTING UP INTERACTIVE
VIRTUAL ENVIRONMENTS**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação sob a orientação científica do Prof. Doutor Paulo Miguel Jesus Dias, Professor Auxiliar da Universidade de Aveiro e da Prof. Doutora Maria Beatriz Alves de Sousa Santos, Professora Associada com Agregação da Universidade de Aveiro, ambos do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro.

*Dedico este trabalho às minhas avós
María e Palmira.*

o júri

presidente

Prof. Dr. Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar da Universidade de Aveiro

Prof. Dr. Paulo Miguel de Jesus Dias
Professor Auxiliar da Universidade de Aveiro

Prof. Dr. António Fernando Vasconcelos Cunha Castro Coelho
Professor Auxiliar do da Faculdade de Engenharia da Universidade do Porto

agradecimentos

Primeiramente agradeço aos meus pais, que sempre acreditaram em mim e me proporcionaram uma incrível oportunidade. Sem eles nada disso seria possível. Então deixo aqui o meu profundo agradecimento ao Sr. Altino e a Sra. Fernanda.

Agradeço à todos os meus amigos que me apoiaram e forneceram ajuda sempre que precisei, especialmente a ajuda da minha namorada Dayane que não apenas me deu apoio emocional, mas também operou o photoshop com maestria. Agradeço também à Cecília pela ajuda com os programas de modelagem, que foram um dos pontos principais para a realização deste trabalho.

Agradeço aos parceiros da TEGOPI e do INSECPorto que, no âmbito do projeto Produtech, gentilmente disponibilizaram os modelos 3D que foram utilizados na seção de resultados.

Agradeço ao Professor Paulo Dias e à Professora Beatriz Sousa Santos, que me orientaram durante este projeto e acreditaram em mim do começo ao fim, ajudando-me das mais diversas formas.

palavras-chave

realidade virtual, interação, ambientes virtuais

resumo

Este trabalho apresenta a criação da Plataforma para Configuração de Ambientes Virtuais Interativos (com o acrónimo em Inglês pSIVE). Tendo em mente a dificuldade necessária para a criação de ambientes virtuais, a plataforma tem como objectivo possibilitar a não especialistas tirarem proveito de ambientes virtuais, em aplicações genéricas, como por exemplo visitas virtuais que sirvam como publicidade ou treino onde seja possível interagir com elementos do ambiente para extrair informação contextualizada. Para isto apresenta-se um levantamento de tecnologias e *frameworks* passíveis de serem envolvidos no processo de criação e justifica-se a escolha dos mais adequados para integrar a plataforma.

A plataforma permite que utilizadores, a partir de uma ferramenta de configuração, criem ambientes virtuais e seus aspectos, bem como modos de interação e indiquem o *hardware* a ser utilizado. Para a construção do mundo, é possível carregar modelos 3D associando-lhes informação multimédia (Vídeos, Textos ou Documentos PDF).

Paralelamente ao desenvolvimento da plataforma, foi realizado um estudo comparativo entre duas técnicas de seleção por *ray-tracing*, que diferem quanto à origem do feixe. A análise dos resultados sugere qual técnica que melhor se adequa aos ambientes criados. O estudo também demonstra a flexibilidade da plataforma, uma vez que esta foi adaptada para servir como ambiente de teste.

Apresenta-se ainda um caso de estudo, onde se mostra passo a passo a configuração de um ambiente virtual e a sua utilização no âmbito do projeto PRODUTECH-PTI.

Por fim, são apresentadas conclusões e possíveis caminhos a serem seguidos para a evolução futura do trabalho.

keywords

Virtual Reality, Virtual Environments, Interaction

abstract

This dissertation presents the creation of the Platform for Setting-up Interactive Virtual Environments (pSIVE). Bearing in mind the difficulty required to create virtual environments, the platform aims to allow non-specialists to benefit from virtual environments in applications such as virtual tours as marketing or training where one could interact with elements of the environment to extract contextual information. For this, several frameworks and technologies possible of been integrated into the platform are presented, as well as which ones are more suitable.

The platform allows users, from a configuration tool, to create virtual environments and set up their aspects, modes of interaction and what hardware to use. The construction of the world is done by loading 3D models and associating multimedia information (videos, texts or PDF documents) to them.

Alongside its development, a comparative study between two ray-tracing selection techniques was performed. Based on the results analysis, it is suggested which technique better fits the environments created with pSIVE. The study also demonstrates the flexibility of the platform, since it was adapted to serve as a test environment.

A case of study is introduced where a step by step configuration of a virtual environment is shown, as well as its use within the PRODUTECH-PTI project. Finally, the conclusions are drawn, and suggestions for future work are presented.

Table of Contents

1	Introduction	1
1.1	Motivation.....	1
1.2	Objectives	2
1.3	Structure.....	3
2	State of the art	4
2.1	Virtual Reality.....	4
2.1.1	Output Devices	5
2.1.2	Input Devices.....	10
2.1.3	Interaction.....	15
2.1.4	Virtual Environment Graphic Menus	17
2.2	Graphics frameworks	19
2.2.1	OpenSceneGraph.....	19
2.2.2	OpenSG	20
2.2.3	VTK.....	20
2.3	Virtual Environments Frameworks.....	21
2.3.1	VR Juggler.....	22
2.3.2	inVRs.....	25
2.3.3	Vizard	27
2.3.4	3DVIA Studio	28
2.3.5	Other Solutions.....	29
2.3.6	Conclusion.....	30
3	Easily configurable Virtual Environment	32
3.1	System Requeriments	33
3.2	Archtectural Decisions.....	36
3.2.1	Frameworks and Modeling Tools	36
3.2.2	Virtual Environment Interaction	37

3.2.3	Configuration Interface	40
3.3	System Development	40
3.3.1	Virtual Environment.....	42
3.3.2	Configuration Tool.....	51
4	User Evaluation	52
4.1.1	Hypothesis.....	53
4.1.2	Method	54
4.1.3	Results	56
4.1.4	Conclusions	59
5	Case of study	60
5.1.1	Creating layout	60
5.1.2	Running Application	64
5.1.3	Conclusions	68
6	Conclusion and Future Work	69
7	Bibliography.....	72
	Annex 01 – VR Juggler 3.x Basic Installation Guide	79
	Annex 02 – VRPN Guide	85
	Annex 03 – Devices Configuration on VR Juggler	90
	Annex 04 – pSIVE Installation Guide	94

List of images

IMAGE 1. THE SENSORAMA (GUTIÉRREZ ET AL., 2008).....	5
IMAGE 2 OCULUS RIFT (OCULUSVR, 2013).....	6
IMAGE 3 PARTICIPANTS IN A CAVE SURROUNDED BY SCREENS DISPLAYING THE VIRTUAL WORLD (CRAIG ET AL., 2009)	7
IMAGE 4 GEOMAGIC TOUCH X (GEOMAGIC, 2013)	9
IMAGE 5 USER TESTING THE SMELLING SCREEN TO SENSE THE AROMA OF COFFEE (MATSUKURA ET AL., 2013)	9
IMAGE 6 FOOD SIMULATOR (IWATA ET AL., 2004)	10
IMAGE 7 RAZER HYDRA (RAZER, 2013).....	11
IMAGE 8 BOOM HEAD MOUNTED DISPLAY (CRAIG ET AL., 2009).....	12
IMAGE 9 LOGITECH 3D MOUSE (SOUVR, 2013)	12
IMAGE 10 VICON BONITA AND MARKERS (VICON, 2013)	13
IMAGE 11 INTERSENSE INTERTIACUBE3 (INTERSENSE, 2013).....	14
IMAGE 12 FAKESPACE PINCH GLOVE (FAKESPACE, 2004)	14
IMAGE 13 SPACEMOUSE WIRELESS (3DCONNEXION, 2013).....	15
IMAGE 14 LOCAL VERSUS AT-A-DISTANCE SELECTION (MINE, 1995)	16
IMAGE 15 ADAPTED 2D MENU IN A THREE-DIMENSIONAL ENVIRONMENT FIXED ON THE SCREEN	18
IMAGE 16 PEN AND TABLET METAPHOR. REAL (LEFT) AND VIRTUAL (RIGHT) REPRESENTATION (BOWMAN & HODGES, 1999)	18
IMAGE 17 VR JUGGLER ARCHITECTURE (VR JUGGLER WEBSITE, 2012)	24
IMAGE 18 VR JUGGLER CONFIGURATION TOOL.....	24
IMAGE 19 INVRS APPLICATIONS (ANTHES & VOLKERT, 2006)	25
IMAGE 20 OVERVIEW OF INVRS (ANTHES & VOLKERT, 2006).....	26
IMAGE 21 VIZARD IDE (WORLDVIZ, 2012).....	28
IMAGE 22 TRACKERS SETUP ON 3DVIA STUDIO (DASSAULT SYSTÈMES, 2012)	29
IMAGE 23 SUGGESTED BUTTON CONFIGURATION ON WII REMOTE.....	39
IMAGE 24 PSIVE ELEMENTS AND THEIR ROLES.....	40
IMAGE 25 PLATFORM OVERVIEW	41
IMAGE 26 PSIVE SAMPLE CONFIGURATION FILE.....	43
IMAGE 27 MULTIMEDIA MODULE ARCHITECTURE	43
IMAGE 28 PSIVE SHOWING A PDF FILE.....	44
IMAGE 29 PSIVE PLAYING A VIDEO	45
IMAGE 30 INTERACTION MODULE	46
IMAGE 31 USER SELECTING A DINNER TABLE BY HEAD ORIENTATION INSIDE PSIVE	47
IMAGE 32 USER SELECTING A DINNER TABLE BY LASER POINTER INSIDE PSIVE.....	47
IMAGE 33 MANIPULATION OF A SKULL MODULE USING PSIVE MODULES.....	48

IMAGE 34 MENU MODULE	48
IMAGE 35 A THREE ITEM MENU ON AN ENVIRONMENT CREATED WITH PSIVE.....	49
IMAGE 36 UTILITIES MODULE	50
IMAGE 37 PSIVE CONFIGURATION TOOL	51
IMAGE 38 SELECTION TECHNIQUE EVALUATION ENVIRONMENT CREATED WITH PSIVE	53
IMAGE 39 SUM OF ERRORS BY DISTANCE	56
IMAGE 40 AVERAGE ELAPSED TIME BY DISTANCE	57
IMAGE 41 AVERAGE ELAPSED TIME AND ERRORS ACCORDING TO THE STARTER METHOD	58
IMAGE 42 TEGOPI FACTORY RECREATED ON GOOGLE SKETCHUP	61
IMAGE 43 EXPORTING A SINGLE MACHINE FROM THE VIRTUAL FACTORY	62
IMAGE 44 GENERATED CONFIG. FILE	64
IMAGE 45 STARTING VRPN SERVER.....	65
IMAGE 46 STARTING PSIVE VIRTUAL ENVIRONMENT	65
IMAGE 47 TEGOPI'S FACTORY OVERVIEW INSIDE PSIVE	66
IMAGE 48 FERRULE MACHINE HIGHLIGHT INSIDE PSIVE	66
IMAGE 49 MENU ASSOCIATED WITH THE FERRULE MACHINE	67
IMAGE 50 VIDEO ABOUT THE FERRULE MACHINE OPERATION	67
IMAGE 51 STEEL TOWERS CATALOG	68
IMAGE 52 VRPN SHOWING MOUSE STATUS.....	86
IMAGE 53 VRPN SERVER MESSAGES	87
IMAGE 54 VRPN DEMO CLIENT	88
IMAGE 55 VRPN DEMO CLIENT IMPROVED.....	89
IMAGE 56 PSIVE PROJECT ON VISUAL STUDIO	95
IMAGE 57 PSIVE FOLDER	95

List of Tables

TABLE 1 FREE VERSUS COMMERCIAL SOLUTIONS.....	30
TABLE 2 LIST OF EQUIPMENT TO SUPPORT.....	34
TABLE 3 QUESTIONNAIRE RESULTS (MEDIAN OF EACH INDEX)	58
TABLE 4 CONFIGURATION OF TEGOPI LAYOUT	62
TABLE 5 LIST OF LIBRARIES REQUIRED BY VR JUGGLER.....	79

1 Introduction

1.1 Motivation

Virtual Reality (VR) systems are known to let users ‘feel’ the environment (either by images only or combining other stimuli), allowing them to be virtually anywhere and to perform different tasks from day-to-day events simulations to training under extreme conditions that were very difficult to have someone physically present. Companies from different areas of expertise are investing on Virtual Reality to reduce costs of physical infrastructure, time, and travel. For instance the automotive industry, that apply VR on various stages of its production line, from prototyping to productivity improvements (Mousavi, Faieza, & Ismail, 2011). Yet, the advance of hardware to allow VR to feel real, and therefore deliver accurate results, is not followed by the software, or when the software is very advanced it still costs a large sum of money.

The complexity of building a Virtual Environment (VE) along with the domain specific knowledge required and the lack of reusability of a VE (Gutiérrez, Vexo, & Thalmann, 2008) still keep possible users away from VR, either because of the cost (time and financial) or the lack of specialized manpower. Another limitation is the existence of several frameworks and libraries for developing Virtual Environments focused on several areas, both specific (graphics rendering, simulation of oil wells, medical applications, among others) and general (Virtual Reality, Augmented Reality, games, among others), each having their own way of implementing its functions and architectures (Gutiérrez et al., 2008). Because of these aspects, VR lacks of ways to

become more accepted and used, by delivering a high complexity that prevents general users to benefit from it.

1.2 Objectives

Addressing the needs previously mentioned, this dissertation has as main objective to develop tools to aid non-specialists to easily setup an immersive and interactive Virtual Environment to visualize 3D models originated from different sources and with different formats. Another objective is to provide a way to interact with additional information inside the immersive environment using non-conventional hardware, such as trackers and head-mounted displays, as well as exploring natural ways for the user to interact with the environment.

Beside the ease of usage, the tools shall be flexible enough to handle several devices such as motion trackers and head-mounted displays in a transparent way, giving users the ability to personalize their experience with the environment according to the behavior delegated to a specific hardware and allowing them to be applied to different situations, such as the training of employees from a metallurgic factory by viewing contextualized information on 3D elements of its workplace, or even just providing a visit to a virtual museum.

To support the development, a state of the art review ought to be done, on topics such as existing technologies and solutions that could be used to solve the issues hereby presented with the objective of finding out which ways would provide better solutions to cover the objectives of this work.

Part of the work included in this thesis was developed within the PRODUTECH-PTI Project. The project aims to create new processes and technologies for the technological production lines and is integrated by a wide consortium of companies and scientific entities, therefore ensuring a significant set of scientific and technological competencies and the presence of the needed agents and mechanisms to an effective appreciation of its results.

In particular some of the work of this dissertation was developed within the task A,2.3 of the project, which has the objective to provide a platform and tools to ease the development and availability of Virtual Training for developers. The idea is to allow Portuguese industrials to provide simulations and 3D models of their products that can

be integrated easily in simulation tools (such as SIMIO) and, more relevant to this thesis, used to create 3D Virtual scenarios for training and marketing of the products.

1.3 Structure

This dissertation is divided into five major chapters. This first chapter presents the motivation and objectives of this work. The second presents concepts of Virtual Reality and Virtual Environments along with a review of existing frameworks that aid the creation of these environments, presenting some conclusions on if they are to be used or not on this work.

The third chapter presents pSIVE (platform for Setting up Interactive Virtual Environments) with details on its architecture and development, as well as its components.

Chapter 4 presents a user evaluation used to assess the effectiveness of the selection techniques and points out which technique would be better applied to different situations.

In Chapter 5 a case study using 3D models that were produced within the Produtech-PTI project for a real company: TEGOPI is presented. In this chapter we present how to load and configure 3D models (built with sketch up) to create a virtual visit of a plant where users can navigate and interact with the models in order to validate the project of a plant before its creation or/and get training and equipment information within a VR set-up.

The sixth and last chapter contains final considerations about this work: difficulties encountered along its development and other issues that could not be included, but were identified as important for further development.

2 State of the art

2.1 Virtual Reality

In short terms, Virtual Reality can be defined as a “high-end human-computer interface that involves real-time simulation and interactions through multiple sensorial channels” (Burdea & Coiffet, 2003) to provide the user an immersion feeling. As Gutiérrez (Gutiérrez et al., 2008) refers to, Virtual Reality is The Science of Illusion.

The immersion feeling refers to the user to feel immersed in the Virtual Environment, which means the feeling of being inside that environment and being part of it. Bowman and McMahan (Bowman & McMahan, 2007) as well as Gutiérrez also agree that immersion itself is just the physical condition and the feeling of “presence” is the psychological state that leads the user to have the sense of being part of the Virtual Environment, and is triggered by sensorial simulations such as images, sound, force-feedback etc., even though the user knows s/he is in a Virtual Environment.

A typical way to induce the immersion feeling is through the use of a Head Mounted Display (HMD) to provide the visual rendering (with or without stereoscopy) along with trackers that detect the user position and allow to deliver adequate visual, audio and/or haptic stimuli. For instance, according to Gutiérrez, the basic setup of a classic Virtual Environment is a HMD to provide the visual rendering, attached with a positional tracker to keep track of the head or body position to adapt the view according to it, and a hand tracking device to point, select and manipulate virtual objects, whether it has force-feedback or not.

From the birth of the term Virtual Reality (VR) in the beginning of the 90’s, much was expected As much as to create the possibility of building synthetic worlds that were

indistinguishable from what was real (Gutiérrez et al., 2008). Nowadays, in spite of VR advances and its recognition as a relevant tool to be applied in different areas, this expectation along with many other still remain as a dream, always closer, but still a dream. While VR cannot deliver such promise, users take advantage of specialized equipment to provide input to the VR system expecting to receive an output that can be visual, audio, haptic etc. that is at least acceptable to be a simulation of the real world – Multiple senses were stimulated since one of the first VR systems ever created, the Sensorama (Image 1).

In the following sections the most used input and output devices will be briefly presented.



Image 1.The sensorama (Gutiérrez et al., 2008)

2.1.1 Output Devices

When designing a Virtual Reality system the display is usually one of the most predominant aspects of the overall design (Craig, Sherman, & Will, 2009), since it is historically one of the first and most usual way of communication between the system

and the user. Also the immersion level of a VR application is generally directly defined by the kind of display provided to the user. However, besides their relevance, visual stimuli are not necessarily the only and most relevant output method for Virtual Environments.

Output devices can be divided into as many classes as the human senses. Therefore, there are vision devices, audition devices, touch devices and the less common but not less important, taste and smell devices.

Vision Devices

As mentioned before, vision devices were one of the first to be used to deliver feedback from a virtual application, with the first computer generated images back in 1950's (Gutiérrez et al., 2008). The display technology has evolved along with television, projector and LCD technologies. For Virtual Reality devices the most common are head-mounted displays (HMD) and Cave Automatic Virtual Environments (CAVE) (Burdea & Coiffet, 2003) but other devices as handheld displays, virtual tables and panoramic projectors are also used.

Head-mounted displays are the most common type of displays used on Virtual Reality applications (Craig et al., 2009). From the first heavy helmets with screens attached to nowadays lightweight sunglasses-like with small screens. Yet presenting an additional weight to the user to carry, connected with cables to transmit the video and tracking information (the tracking information is usually supplied by an extra hardware that is attached to the HMD). The Oculus Rift (Image 2) is one of the most recent HMD low cost solution, and contains an internal tracking system for orientation.



Image 2 Oculus Rift (OculusVR, 2013)

CAVE systems consist of a room, where high resolution images are projected on the floor, ceiling and walls allowing multiple users at the same time to receive the visual stimulus. The projection is performed by projectors that are located on the back of the screen along with special glasses to give depth perception. Image 3 shows multiple users in a CAVE environment, with only a leader wearing a tracking system but allowing multiple users to join the interaction.



Image 3 Participants in a CAVE surrounded by screens displaying the virtual world (Craig et al., 2009)

Audition Devices

Sounds are present on real-life experience for most people and they offer precious information about the environment. Not differently, in VR systems the sound can play many roles (Gutiérrez et al., 2008). Providing rich information about the stimulated environment – for instance echoes and reverberations give the brain hints about direction and distance of elements as well as the size of an environment. Sound can also become an alternative feedback, for instance indicating the reception of user commands or alarms.

However VR Sound Systems has some requirements to meet. First is the accurate 3D positioning of the sound source. The position where the sound is perceived must match the position of the corresponding element in the Virtual Environment. Second is the acoustics simulation, which is essential to have spatial perception on the Virtual Environment. The last requirement is about the efficient generation of the sound in

(almost) real time which requires a trade-off between the accuracy of the simulation and the speed (Burgess, 1992).

The setup of a VR sound system could be done with headphones, which have more precise control since the signal reaching each ear may be controlled independently. But when the user cannot (or wants to avoid to) wear any device – for instance a collaborative CAVE – the other possibility is to deliver the sound stimuli through multiple loudspeakers, however with this configuration the control of spatial information reaching the user is less precise. Loudspeakers systems are also usually cheaper than headphone-based systems (Burdea & Coiffet, 2003).

Touch Devices

Devices that stimulate the touch sense are called Haptic Devices, and can deliver the stimulation either through the skin (tactile) or through muscles and the skeleton (proprioceptive). Devices that send tactile feedback have a sort of ways to stimulate the skin tactile receptors for example air jets, electrical stimulators producing vibration or inducing heat (Burdea & Coiffet, 2003).

Moreover, Gutiérrez (Gutiérrez et al., 2008) classifies these devices through the nature of the stimuli: passive devices and active devices. Passive devices are those which send feedback according and against the user's movement and act on the force exerted during the user's interaction with virtual elements, here the user is the energy supply for the device to work. While active devices use elements capable of sending stimuli actively, with no need for the user to input any kind of movement to supply the device. Passive devices are considerably safer since the user will not receive accidental torques or forces as s/he is the only source of energy. Commonly, devices combine both classifications, for instance a proprioceptive device (a joystick) that applies force against the movement through a brake (passive) can also be moved by a motor (active). An example of widely used equipment in research laboratories nowadays, the Geomagic Touch™ X, formerly known as Phantom, (Image 4) has a stylus (or a slot for the user to insert one finger) attached to a mechanic arm to measure its position and exert a controlled force vector against it.



Image 4 Geomagic Touch X (Geomagic, 2013)

Taste and Smell Devices

The sense of smell can stimulate the memorization of one's concepts or experiences while taste can trigger anxiety or depression (Gutiérrez et al., 2008). Even though smell was addressed by one of the pioneer VR systems, the Sensorama by delivering different aromas through the air, taste stimuli still lack research (Craig et al., 2009; Gutiérrez et al., 2008) and both, taste and smell, are not much addressed by current research, which does not mean they are forgotten.

Olfactory (smell) systems usually contain different odorants, a system to deliver them through air and a control algorithm to determine the mix of odorants, its concentration and the time of the stimulus. Recent work is a system that can place odor on determined regions of a screen – the Smelling Screen (Matsukura, Yoneda, & Ishida, 2013) – by delivering odorants through a four fans system, to arbitrary positions of the screen, see on Image 5 an user testing the system.



Image 5 User testing the Smelling Screen to sense the aroma of coffee (Matsukura et al., 2013)

The sense of taste has been only marginally addressed in VR systems and few taste interfaces can be found in literature (Burdea & Coiffet, 2003; Craig et al., 2009; Gutiérrez et al., 2008). Iwata (Iwata, Yano, Uemura, & Moriya, 2004), creator of the Food Simulator (Image 6), says that “Taste is very difficult to display because it is multi-modal sensation composed of chemical substance, haptics and sound”; his work addressed the chewing simulation: releasing flavoring chemicals onto the user’s tongue, giving the resistance to the user mouth as s/he chews a rubber cover while a sound is played, all corresponding to the food that is been simulated.



Image 6 Food Simulator (Iwata et al., 2004)

2.1.2 Input Devices

It is difficult to find a single 3D input device that is universal and has good performance in all applications (Frohlich, Hochstrate, Kulik, & Huckauf, 2006), due to the variety of tasks, which requires diverse interaction devices and techniques. A major factor in the development of an input device is compatibility between the degrees of freedom available and the needs of the task. The expression “degrees of freedom” (DOF - Degrees of Freedom) is used to describe the number of system parameters that vary independently.

Trackers are the main sensors used in VR to measure the position and orientation of users and 3D objects along time. These trackers are classified according to their working principle and can be divided into the following(Craig et al., 2009):

Electromagnetic – This kind of tracking system consist of a set of transmitter and receivers of magnetic fields allowing determining the six DOF (position and orientation) of the sensor device. They do not require line of sight between the emitter

of the field and the sensors, have a good rate, usually have a smaller cost than others applied to Virtual Reality (Burdea & Coiffet, 2003). However, metals interfere with the functionality of such a system, as well as walls and floors of concrete. They also are sensitive to any electromagnetic waves and the precision decreases with increasing distance from the tracked object. Fortunately, usually it is possible to keep control of the amount of metal within the environment. Also, Alan Craig says that “Cases where particular care must be taken to improve tracking accuracy are head-worn gear made of metal or with internal electronics, and wheelchairs. In the case of HMDs or stereo glasses with electronics, the best solution is to locate the sensor as far away from the electronics as possible” (Craig et al., 2009). An example of electromagnetic tracker is the Razer Hydra (Image 7), which is composed by a base emitting a magnetic field and two controllers that use the magnetic field to detect their position and orientation.



Image 7 Razer Hydra (Razer, 2013)

Mechanical – Although they were pioneers, mechanical tracking systems are still used and are probably more efficient tracking technique today. They are fast, with accurate calculation of the position of a single point on the target and do not need calibration. However, the tracked object is connected mechanically to a fixed position and this connection limits the user movements. But in situations where the movement of the user is already limited, the use of a mechanical tracking system does not imply an additional restriction and can be easily used – for instance a pilot sitting in a cockpit. The BOOM display (Image 8) was a common example of mechanical tracker, it is a Head Mounted Display connected to an articulated mechanical arm. The position and orientation information are obtained through that arm.



Image 8 Boom Head Mounted Display (Craig et al., 2009)

Ultrasonic – these trackers require a set of transducers – transmitters (speakers) and receivers (microphones) – to determine position coordinates and orientation. They work by measuring the time that the ultrasonic signal takes to arrive at the receiver. The system detects the arrival phase signal and compares the relative values of signals, thereby calculating the distance between the receiver and the transmitter. By applying this with multiple transducer-pairs it is possible to determine position and orientation of the object. The problems related to acoustic systems are the need of line of sight between the transducer pairs, the dependence of the orientation of the receivers, and external noise interference. One workaround to solve some of these constraints is by mounting several transducers on the sensor device, providing redundancy thus allowing the sensor to go through different orientations and keeping contact with the transmitters. Ultrasonic systems are simple, efficient and have low price. In Virtual Environments they are usually used in combination with other tracking systems to ensure a good performance. Image 9 shows the Logitech 3D Mouse, with three microphones to detect ultrasonic signals and track its position and orientation.



Image 9 Logitech 3D Mouse (SouVR, 2013)

Optical – This kind of tracking works similar to ultrasonic systems, but using light instead of sound, usually by recognizing the location and orientation of markers. These markers can be either passive or active; the first reflect light (typically infra-red) sent by the camera, usually reflecting spheres or circles, while active markers use LEDs that emit beams of light directly into the camera. The passive markers do not contain electronic nor mobile parts, which make them lightweight, robust and inexpensive, and do not require wires. However, these systems are expensive and require line of sight for at least two cameras at the same time. A known solution of optical tracking system is the Vicon Bonita (Image 10), that triangulates the position of markers to provide their position and orientation.



Image 10 Vicon Bonita and Markers (Vicon, 2013)

Inertial – Work with gyroscopes and accelerometers, which calculate respectively the position and orientation of users. The major problem of these systems is due to the fact that they only report relative movements, not absolute positions resulting in drift errors, for instance after a time using the system it may report a few degrees bias to a certain direction. Another aggravating circumstance is that accelerometers are sensitive sudden variations, generating errors that are accumulated, thus resulting in low accuracy. However, inertia-based systems do not require line of sight, have no problems with interference, are not very expensive or require transmitters. However, they need to be recalibrated often. Image 11 shows a solution by InterSense, the InertiaCube3, an inertial 3DOF tracking system,



Image 11 InterSense InertiaCube3 (InterSense, 2013)

Trackers, however, are not enough to provide the needed input for a system, and other structures are required. Devices like the **Data Gloves**, can provide different, inputs according to gestures of the user's hand, therefore providing more natural interaction with the system (Burdea & Coiffet, 2003). However, given its complexity they are expensive and, usually, are not adjustable for the user's hand. The Fakespace Pinch Glove (Image 12) avoids the adjustability problem by using conducting wires instead of hard components.



Image 12 Fakespace Pinch Glove (Fakespace, 2004)

Other devices may aid the user to perform navigation and manipulations on the system, for instance the **trackballs** and **3D mice** (Burdea & Coiffet, 2003).

They consist of cylinders or spheres that provide input according to the force applied to it by the user. Even though its behavior is very similar to a usual mouse, they do not need surface to work on, are compact and do not require much space. The 3DConnexion SpaceMouse® Wireless (Image 13) allow the user, for instance, to manipulate 3D models simply by pulling, pushing or twisting its cap. It also contains 2 programmable buttons.



Image 13 SpaceMouse Wireless (3Dconnexion, 2013)

2.1.3 Interaction

Interaction in a Virtual Environment strongly depends on the input devices/interfaces available and the way one interacts with the environment and its elements to achieve certain objectives can be divided into smaller tasks.

For instance if a user wants to change the color of a car that is far away s/he first will need to locate it, then s/he must tell the system that the object s/he wants to interact with is that specific car, if there is a restriction about the distance to the object to be manipulated, the user may also need to navigate first towards the car before.

As proposed by Bowman et al. (Bowman, Kruijff, LaViola, & Poupyrev, 2004), the universal tasks are navigation, selection, manipulation and system control.

Navigation is the act of moving from place to place and in a virtual world, as it is in the real life. It can be divided into two sub-tasks: travel (the motor component) and wayfinding (cognitive component). To move, one must define a path through the environment by using spatial knowledge about it – this is the wayfinding, while the travel task is conceptually simple, it is the proper act of controlling the movement through the world. In a virtual world all these information must be sent through input devices. Usually the travel is triggered by tracking the head and the input of commands (buttons, or other interfaces such as treadmills) to move the viewport accordingly (Craig et al., 2009).

Selection is the act of indicating options that were given inside the VE, for instance which way to go or which element of the scene the user wants to interact with. To achieve that there are a variety of methods, some of them use the placement and position of the user's body, for instance pointing with a finger and gazing with the eyes; these are called selection at-a-distance (Mine, 1995), which means, the object is outside the user's reach. One of the most common techniques to perform selection at distance is the ray-tracing, where a beam is fired and intersects the objects in the virtual world. Alternatively, it is possible to perform the selection by tracking the position of the user's hand or by controlling a virtual cursor (also called virtual hand) until it is within the virtual object's reach – this method is called local selection. Both methods are presented on Image 14.

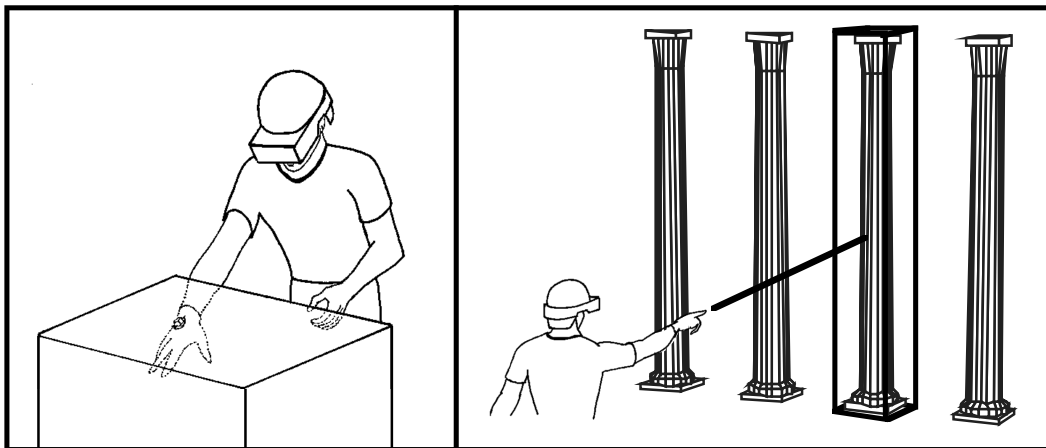


Image 14 Local versus at-a-distance selection (Mine, 1995)

Manipulation after the selection, the user might want to manipulate the selected element, either by applying forces, altering the 'physical' state of an object or by changing attributes of the object.

System Control is a command sent to change the system state or an interaction mode. Manipulation and System Control sometimes get under the same classification (Craig et al., 2009), and the issuing of a command to manipulate something (an element of the VE or the VE itself) can be achieved by different techniques for instance voice commands, gestures, graphical menus or by combining multiple techniques (Kim, 2005).

2.1.4 Virtual Environment Graphic Menus

Interaction between user and computers is achieved by the usage of a User Interface (UI), which includes both hardware and software (Kim, 2005). With the evolution of both, the methods and possibilities are rising even more, for instance the hardware could be from a simple keyboard to a high-end optical tracking system while software techniques go from 2D images to sound and image processing, creating a two-way communication – translating the user inputs to computer representations that can be understood by it and act upon and then translate back to the user (Hix & Hartson, 1993).

Considering the software, one of the most used graphical UI is a ‘menu’. Which can assume virtually any function: issue commands, change the mode of interaction, trigger events and so on. They are very common on 2D interfaces but, even though they present some problems related to the adaptation of 2D tasks, are still effective for 3D interactions (Bowman & Wingrave, 2001; Kim, 2005).

Just like the 2D counterpart, three-dimensional menus have many formats and styles of presentation. Some of them, as described by Kim (Kim, 2005), are the following.

Adapted 2D Menus are, as the name says, 2D menus that are mapped into 3D geometry – using text labels rendered on rectangles (Image 15) making them one of the easiest types of menus to implement into a VR application. Kim also classifies the placement of these menus, the surround-fixed windows that are displayed at a fixed position inside the world, display-fixed windows that are placed relatively to the head orientation and world-fixed windows that are located in the world or on objects and are evocable by the user. Bowman (Bowman & Coquillart, Sabine, Froehlich, Bernd, Hirose, 2008) calls attention to the drawback that pointing at items and menus is more difficult in free space than on the desktop and might lead to lack of precision and exhaustion (Kim, 2005).



Image 15 Adapted 2D menu in a three-dimensional environment fixed on the screen

The **Pen and Tablet** metaphor adds feedback to the user which allows a more precise pointing as, instead of being placed directly on the environment, the 2D menu is projected on a physical tablet surface and a stylus is used to activate the items, drag icons or press buttons (Bowman & Hodges, 1999). On this metaphor both stylus and tablet are tracked and duplicated into the virtual world (Image 16) so the menu is showed only when the tablet is within the user's viewport. Another variation of the pen and tablet metaphor is the use of a device that shows the menus on a real 2D display, for instance a PDA; however, the user must be able to view the device, removing some immersion of the application.

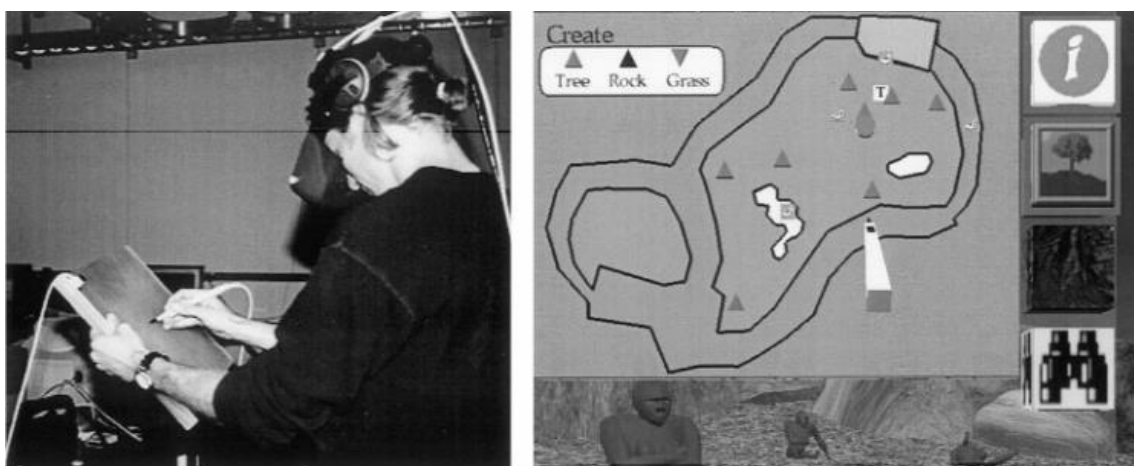


Image 16 Pen and tablet metaphor. Real (left) and Virtual (right) representation (Bowman & Hodges, 1999)

The use of graphical menus into VEs can be adapted to many needs, but even though studies attempt to better understand and apply them, there is no standard solution.

2.2 Graphics frameworks

Graphics frameworks and libraries have the purpose of presenting Virtual Environments visually, providing tools to interact with it through GUI (Graphic user interface) elements. Next, the frameworks covered by this work are presented, namely OpenSceneGraph, OpenSG and VTK (Visualization Toolkit).

Most of the frameworks mentioned here are scene graphs, which is a structure that arranges the logical representation of a graphic scene in a hierarchical way composed of a sort of nodes in a tree structure – the representation can also be a graph, but since the nodes often have only a single parent node the most common representation structure is a tree.

2.2.1 OpenSceneGraph

OpenSceneGraph (OSG) is an engine for graphics rendering which creates an abstraction layer to OpenGL, created to facilitate the development of applications that require advanced graphics features such as games, simulators, Virtual Reality and others. It contains several frameworks associated with it – called NodeKits, each responsible for one aspect of the definition of the graphic environment. The central framework (`osg`) manages the graphic scene as a whole and orchestrates the usage of the other frameworks, which are: `osgParticle` (particle system), `osgText` (fonts and texts), `osgShadow`, `osgTerrain`, `osgAnimation`, `osgVolume` (volume rendering) and `osgViewer` (GUI management) (R. Wang & Qian, 2010).

OSG manages the resources on a plugin-based approach, which means, the support to multiple data formats necessary to create the graphic environment (3D Models, Textures, among others) is customizable, giving the user a better control of the resources in its application. As for the formats supported, they can be as long as the list of OSG's plugins which can easily get up to 50. Most of the common and popular 3D formats are included, for instance Collada (widely supported by autodesk softwares), 3D Studio Max (.3ds and .max), Blender, Maya and 2D elements like gif, jpg, tiff, etc.

This framework was conceived under a strong open source paradigm and is free to use. Its functionalities (and users) have been growing since its creation in 1998, with

thousands of users participating in the official mailing list (R. Wang & Qian, 2010), which is very active. The current stable version is the 3.2 counting with 511 contributors, to the date.

2.2.2 OpenSG

The name OpenSG is very similar to OpenSceneGraph but they are (slightly) different frameworks. Even though they are designed to render a scene, OpenSG differentiates on the clustering and multi-thread safety capabilities (Voß, Behr, Reiners, & Roth, 2002). OpenSG focus on the clustering capability by simplifying for the user the whole process transparently for the application, supporting a wide variety of graphics clusters. The list of supported elements to get imported to a scene also covers the most popular formats, for instance Collada, Shape Files (cartography) and VRML. Moreover, OpenSG leaves a door opened to those who design scenes to OpenSceneGraph by supporting OSG's binary format (ive), so a whole scene could be exported from OSG directly into OpenSG.

OpenSG started around the same period as OSG, when SGI started to slow the Performer (another framework) development. It is also open-source and its current version is 2.0; however its development is currently almost stalled, with a not-so-active mailing list and few contributors.

2.2.3 VTK

The Visualization Toolkit (VTK) is also a framework that abstracts the usage of OpenGL, but differently of the other frameworks mentioned, it does not have a hierarchical structure. Instead it adopts entities called actors that contain the information regarding the geometry of the objects.

In addition to the Actors, VTK has other graphic entities: light sources, particle systems, shadows and 2D and 3D widgets. Regarding the GUI support, VTK already presents a sort of plugins that are well defined, for example QT and wxWindows for windows management (Schroeder, Martin, & Lorensen, 2006). VTK also supports various types of image files (png, jpeg, tiff, bmp and ppm) and 3D objects (VTK's own xml, 3D Studio Max, obj, among others). An important difference of VTK is the native capability to support Image Processing with a set of filters and effects.

VTK is open source and is supported by Kitware, a professional company that offers training and support. The core language of VTK (as well as the other frameworks mentioned) is C++ but it supports wrappers of the core to work with other languages, even interpreted ones as Python, Java and Tcl (Schroeder et al., 2006).

2.3 Virtual Environments Frameworks

When creating a Virtual Environment (VE), there are a few things the developer must take into consideration such as the handling of different input/output devices (Trackers, Projectors, Head Mounted Displays etc...), graphics rendering, the way to interact with them, etc. (Anthes & Volkert, 2006; Bastos, Raposo, & Gattas, 2005; Bierbaum et al., 2001; Gutiérrez et al., 2008). Given the high complexity of building a VE from scratch, in the past ten years the development of new and the improvement of existing frameworks to aid the creation of VEs has been very active (Anthes & Volkert, 2006; Bierbaum & Hartling, 2005; Dassault Systèmes, 2012; Kelso, Arsenault, Satterfield, & Kriz, 2002; Pavlik & Vance, 2012; Teixeira et al., 2012; F. Wang, 2010). Those frameworks give the freedom to build the VE with aid in important steps of the development, such as the handling of devices, projection systems, and interaction behavior. This saves the development team's time and effort although different frameworks offer different solutions, which means, the benefits offered by a specific framework could be enough to one case but not for another. For instance, a simple environment, with few functions would be more concerned about the device abstraction provided, so this aspect would be important to define the framework to choose but for a large environment with heavy and complex components would be mandatory to distribute the application among processing nodes, so a framework that supports distributed systems would be better.

Those frameworks also present themselves as a (partial) solution for the lack of flexibility on the VE systems since, as highlighted by (Gutiérrez et al., 2008), “the ‘reinventing the wheel’ and ‘not invented here’ syndromes limit the innovation and delay the use of VEs in wider areas for the general public” which means, the specialization applied to the VE systems tie them up to their own scope giving limited possibility to reuse for another purpose. Whether commercial or free solutions, such as VR Juggler (Just & Bierbaum, 1998), 3DVIA Studio (Dassault Systèmes, 2012), Vizard (WorldViz, 2012) and inVRs (Anthes & Volkert, 2006), the main objective of those is

either reduce the complexity of building the VE or provide the flexibility that VE systems lack.

On the following sub-chapters different solutions are going to be presented after an analysis based on available documentation and publications about different frameworks to build Virtual Environments as well as quick experience where possible, focusing on the following aspects:

- How to interact or use the framework;
- Which modules/interfaces it provides;
- Extensibility;
- Which Graphic Engine it depends on.

2.3.1 VR Juggler

Described as “a virtual platform for the creation and execution of immersive applications, that provides a Virtual Reality system-independent operating environment” (A. Bierbaum et al., 2001), VR Juggler (VRJ) was born on Iowa, United States on the Iowa Center for Emerging Manufacturing Technology from Iowa State University, in 1998 (Just & Bierbaum, 1998) and is still very active around the world (Aron Bierbaum & Hartling, 2005; Fowler, Carrillo, Huerta, & Fowler, n.d.; Melin & Allard, 2002; Pavlik & Vance, 2012) and even in Portugal (Costa, Pereira, & Dias, 2007).

It provides free and open source application framework along with a set of C++ classes to create VR Applications with support to various graphic APIs, such as OpenGL (Shreiner & Group, 2009), OpenSceneGraphs (R. Wang & Qian, 2010), OpenSG (Voß et al., 2002) and VTK (Schroeder et al., 2006), maintaining the easy and generalized management of displays and input and output devices (Just & Bierbaum, 1998) also supporting cluster applications.

VR Juggler introduces a modular architecture to VR Applications. Those modules, shown on Image 17, give VRJ the flexibility to be used on a wide variety of VR systems. Beginning with VR Juggler Portable Runtime (VPR), which is the Abstraction layer between VRJ and the operating system. Providing platform independent functionalities, such as threads, sockets and I/O primitives. It's the VPR that orchestrates the whole set of modules. In VRJ instead of programming the “main()”

function, the VPR defines a set of interfaces to be handled by its kernel, and all applications are built as objects, so called application objects, implementing the interfaces derived from base classes for specific graphic engines to be called by the kernel.

Along with the cross-platform possibility for the system to run, the Tweek module provides a collection of different technologies that allows Java user interface to communicate with a C++ application, maybe the only drawback of this module is the imposition of CORBA (Common Object Request Broker Architecture) as the communication middleware, given the fact that it went from a bleeding-edge technology to an almost forgotten technology (Henning, 2006) even though it is still been maintained and updated¹ by the Object Management Group.

The Juggler Configuration and Control Library (JCCL) handles the configuration of the components with a XML-based configuration system that allows runtime changes and it is a structured method to process XML configuration files that can also be used to control application specific settings defined by the user, for instance the position and color of an element on the virtual world. On a short attempt of using JCCL to parse user data to the application, it was very hard to define on which step of the application/rendering loop the configurations were parsed; moreover, the documentation regarding this tool is outdated and almost inexistent.

The Gadgeteer module provides a device management system that handles the control, acquisition and representation of VR devices data. It is very extensible since it is possible to write new drivers for devices that are not yet supported. In contrast with JCCL, the documentation on this matter is very detailed and defines all the steps and basic functions the driver interface must have (“Gadgeteer Device Driver Authoring Guide,” 2010). The Gadgeteer also integrates with the Virtual Reality Peripheral Network (VRPN) (II, Hudson, & Seeger, 2001) extending the supported devices list and allowing the distribution of peripherals through a network.

Sonix is an optional component that provides a layer to immersive audio capabilities.

¹ CORBA version 3.3 or CORBA/ZIOP was released on 2012 by the OMG

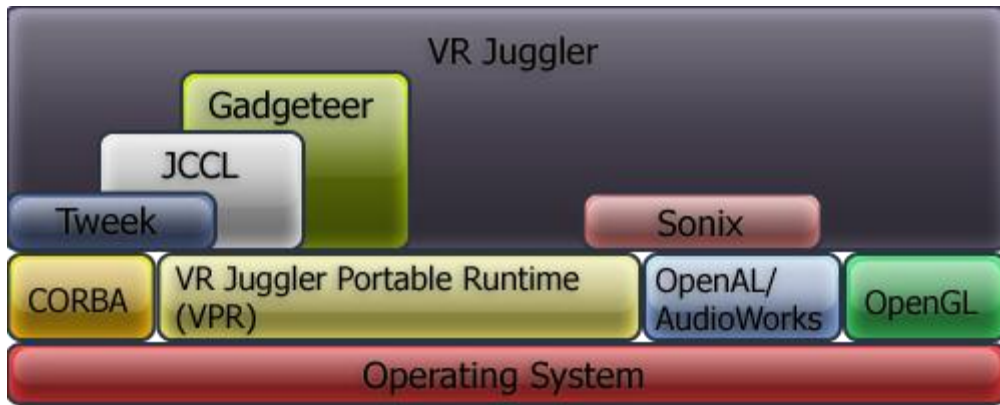


Image 17 VR Juggler Architecture (VR Juggler Website, 2012)

This architecture allows extension to a specific module without impacting the rest or existing applications, for instance adjusting the Gadgeteer to receive a new device is transparent to applications (Bierbaum et al., 2001) but this also makes the system become easily large since even if the application does not use a module it is still there inside VRJ, for instance the Tweek module and the Corba communication layer. Another drawback for VRJ is the complexity of the configuration files, since it requires many small adjustments that could be confusing, particularly for first time users and the tool (Image 18) provided to manage those files is simple and just provides an user interface to edit the XML fields which an experienced user would just edit directly on the file.

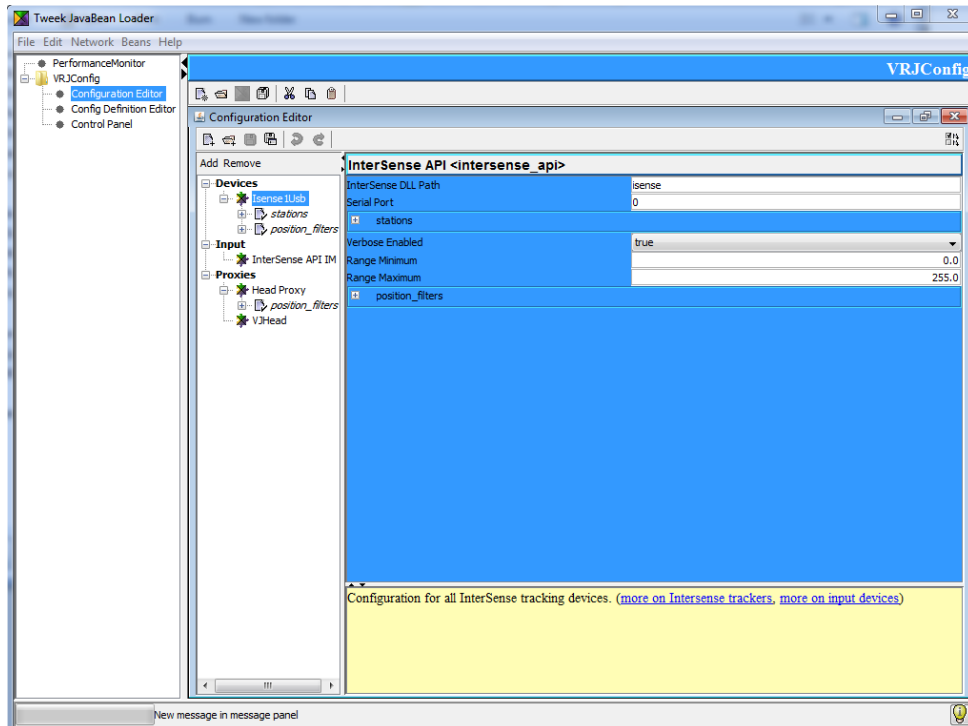


Image 18 VR Juggler Configuration Tool

2.3.2 inVRs

Born at Johannes Kepler University, Austria, the Interactive Networked Virtual Reality System (inVRs) presents a full VR Framework structured in a flexible and modular architecture (Anthes & Volkert, 2006). It provides “independent modules for interaction, navigation, and networking, an additional system core module, and two interfaces for the abstraction of input devices and output devices” (Anthes & Volkert, 2006). Most of the users of inVRs remain in academic environments from Austrian and German universities focusing on the network and collaborative capabilities of the framework, as seen on Image 19, that allows collaborative interaction on the same world by users geographically separated. Besides that, the framework presents itself, also, as a solution to build Virtual Environments (VE) on a not necessarily networked environment, given its modularity and modules independency that allow using not all modules at once.



Image 19 inVRs Applications (Anthes & Volkert, 2006)

The framework is written in standard C++ with additional libraries that are platform independent and Open Source. Currently inVRs is bound to OpenSG 1.8 as its scene graph API. Since late 2011 it is also possible to work with OpenSceneGraphs but the inVRs core is still dependent of OpenSG functionalities such as threading. As for audio, it supports only OpenAL, however Anthes and Volkert (2006) say that “other

libraries are planned as well". Currently its development is stalled without major updates since 2011.

The modular architecture of inVRs, seen on Image 20, as previously stated can be individually connected to the core but they also can be used as independent libraries. Either way, the modules have boundaries well defined, and they communicate with the interfaces to handle input devices, such as trackers and sensors, and output devices like displays and audio. The abstraction provided by the Input Interface allows the application to recognize only three components: buttons, axes and sensors. Each input device is represented by a combination of those three components.

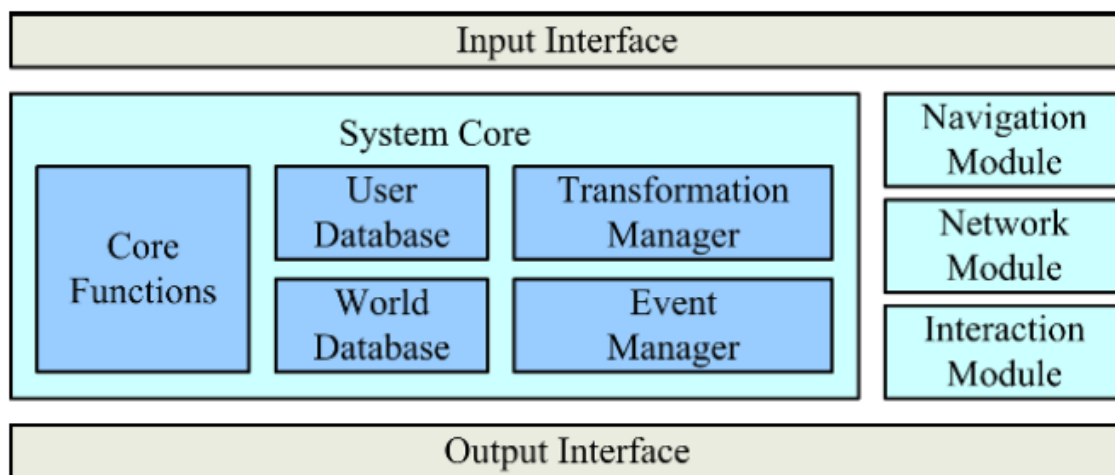


Image 20 Overview of inVRs (Anthes & Volkert, 2006)

Among all modules, the system core is the most important since it contains, beside the core functions, all information towards the VE created (user and world database). The Event Manager handles the communication between the databases and other modules but only the Transformation Manager manipulates objects in the VE. The framework splits navigation and interaction into two different modules, the Navigation Module provides pre-defined modes to travel along the VE, making use of the abstract controllers defined by the input interface. The Interaction Module also uses abstract controllers but it has two main tasks, object selection and object manipulation. The Network Module is the responsible to distribute the events among the networked VE.

inVRs seems to be a nice and reliable system but its setup consists of many XML files with very tiny details that demand close attention and show a high level of complexity. Those files must be written manually, since there is no tool to aid but the configuration guide, which makes the process of configuration slow, tiring and possibly

discourage the first-time user since the learning curve of the configuration step is long. Another drawback of inVRs is that even though its development is active, it is slow and has a small community of contributors so it is hard to expect new features or improvements to be released fast. But the possibility of reducing the complexity to connect separated people inside the same VE through network is a considerable advantage along with the pre-defined interaction and navigation techniques which save time and allow defining the technique to be used in a faster way regarding the world built.

2.3.3 Vizard

Vizard VR Software Toolkit stands as an integrated tool developed by WorldViz (Santa Barbara, US) designed for rapid prototyping. It is a commercial solution but a demo version is available at no cost. It is used among different institutions from universities, such as the Virtual Human Interaction Lab at Stanford University, to companies as Archidimex² on Netherlands.

Vizard at first sight appears to be an IDE (Integrated Development Environment) with a friendly GUI to help the user setup the environment but keeps the development process at a relatively low level, since the VE is built using Python Scripts wrapping a scene graph engine, specifically Open Scene Graph (OSG), and the core C++ functions developed by WorldViz. Even though it requires some programming knowledge, the core functions provide resources to make possible the creation and deployment of even a hard task rapidly. Also the scripting language is claimed to be something that even a user with no programming skills could start and take a dive into interactive 3D environment. Image 21 shows Vizard IDE with a snippet of the Python Scripting and the integrated preview feature.

² <http://archidimex.nl/>

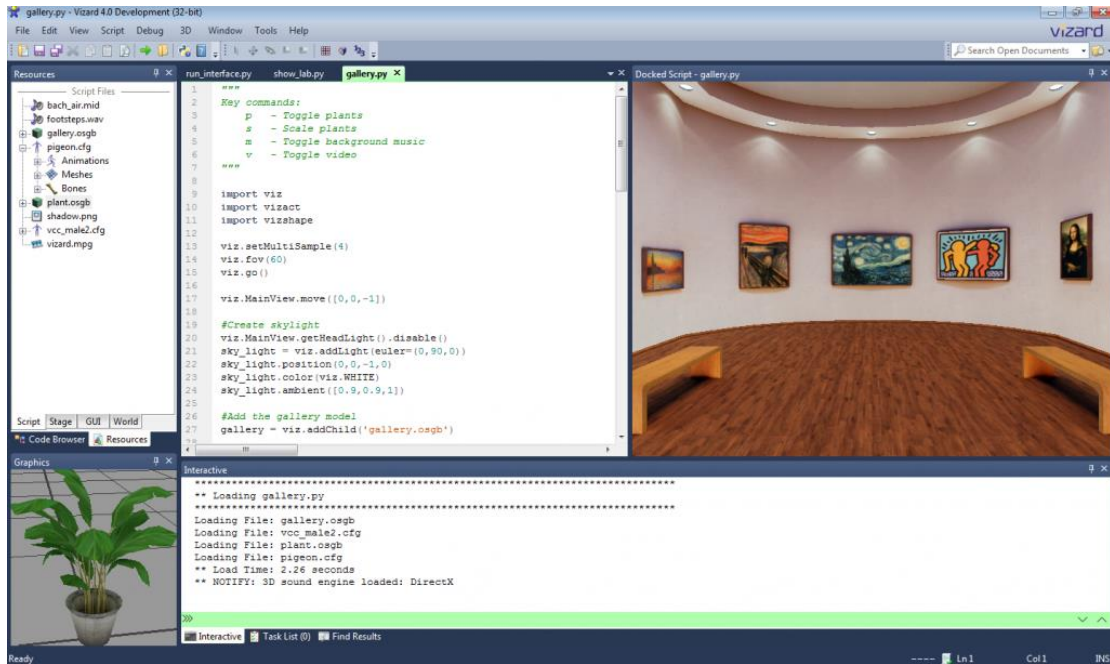


Image 21 Vizard IDE (WorldViz, 2012)

This solution also offers the possibility of using OSG directly through native C++ or using Python bindings. This possibility gives the user the ability to interact with the graphic engine and add tweaks or new modules to expand its functionalities or improve the existing ones.

The core functions of Vizard also include: a physics engine, a device abstraction module capable of communicating with a vast list of I/O devices (also integrated with VRPN), a clustering handler and a module to integrate augmented reality. The cheaper, but more limited, version starts at US\$ 75,00 (single license), however it only supports standard displays in full screen, limits the user on what is possible to customize and extend from core functionalities and has no distributed rendering. The most complete version starts at US\$ 6000 a single license for academic usage.

2.3.4 3DVIA Studio

Dassault Systèmes (2012) presents a toolkit highly used in industry to perform simulation or training, among other possibilities. Formerly called Virtools, the 3DVIA Studio is a solution to build, not only generic VEs covering a wide range of features from graphics rendering, creating user interfaces, physics simulation, artificial intelligence.

3DVIA Studio offers different editors to aid the creation of a VE with just some mouse clicks, for instance (Dassault Systèmes, 2012) shows the configuration of a new tracking device. Besides the graphic editors to aid the creation of VEs, it is possible to use the Virtools Scripting Language along with LUA. This characteristic makes it flexible enough to be used by non-specialists and to allow lower level coding to create personalized simulations or rendering but since its core engine is closed the personalization is theoretically limited.

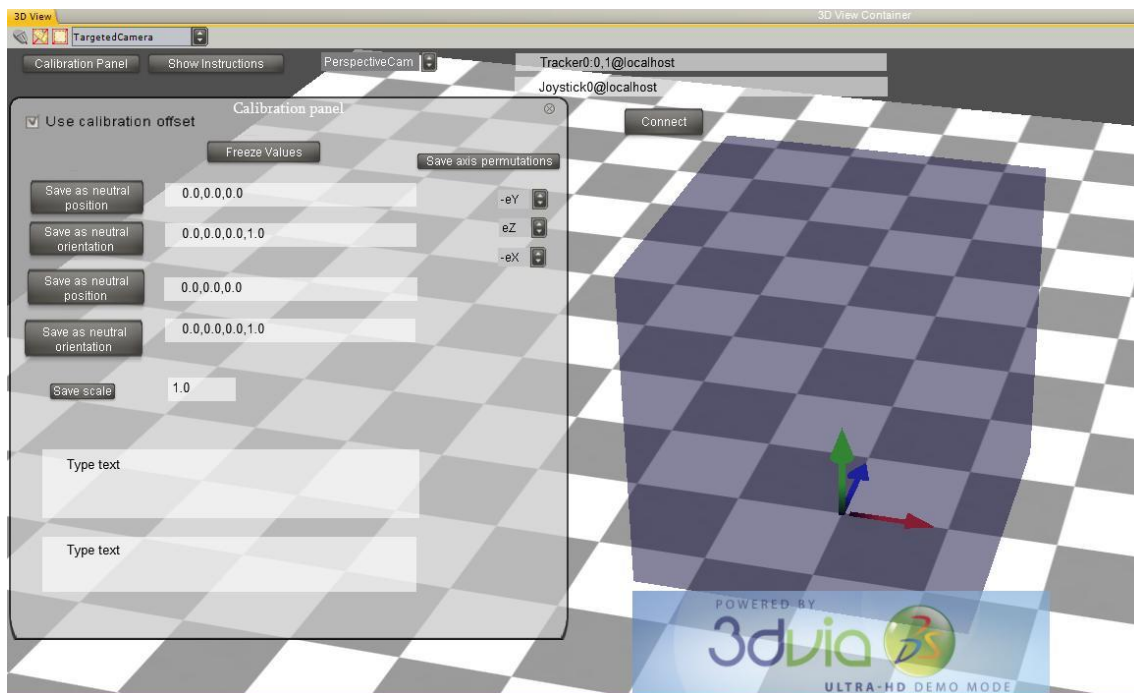


Image 22 Trackers setup on 3dvia Studio (Dassault Systèmes, 2012)

2.3.5 Other Solutions

Beside the tools hereby presented, the range of solutions is vast, when it comes to aid the user building a Virtual Environment, and solving problems related to it. Just like inVRs some of those will already provide pre-defined ready to use interaction techniques, for instance Avango developed by IMK/Fraunhofer Institute (Kuck, Wind, Riege, & Bogen, 2008) that stands as a framework to develop distributed Virtual Environments, focusing on high-end systems. The drawback of Avango is that it was built to work with SGI Performer Scene Graph, and since SGI stopped distributing the

Performer, they migrated to OpenSceneGraphs, but it still lacks of experience on this Scene Graph.

Not as simple as Avango but with more flexibility, the DIVERSE system was developed at Virginia Tech Institute (Kelso et al., 2002). It also supports distributed systems also depending on SGI Performer, but with the possibility of switch to VTK.

ViRAL (Virtual Reality Abstraction Layer) is a framework based on graphic components (Bastos, Silva, Raposo, & Gattass, 2004) that eases the creation of extensible applications with integrated WIMP (Windows, Icons, Menus and Pointers) interfaces. Its usage is through graphical interfaces, where the user creates, configures and connects to components. It was developed by the Group of Computer Graphics Technology at PUC, Rio de Janeiro.

Also from the Group of Computer Graphics Technology at PUC, Rio de Janeiro, the LVRL (Lightweight Virtual Reality Libraries) (Teixeira et al., 2012) was introduced aiming not only the creation of new VR applications, but also the conversion of existing desktop graphic applications to VR without altering its structure, providing a set of libraries with a minimalist programing interface to allow non VR developers to easily setup a Virtual Environment with manipulation and interaction techniques ready to use.

2.3.6 Conclusion

From the previous overview, it became clear that the first main difference to be taken in consideration on when choosing a solution is the pros and cons of commercial and free solutions. Therefore the first comparison is presented on Table 1, accounting the main differences observed.

Table 1 Free versus Commercial solutions

Free	Commercial
Low level tool	Interactive GUI
Possibility to work directly with the graphic engine	Core engine (mostly) not accessible
Requires programming skills	Layman friendly
Cross Platform	Majority for Windows Platform

Free solutions are, initially, harder to work with and stand as a challenge with a large learning curve but with higher flexibility, since they provide a set of low level tools and the possibility to interact directly with the core graphic engine and expand or add tweaks at will, while commercial solutions do not have the same flexibility due to theoretically³ limited level of customization, but their interactive user interface allows non-experts to work with it and develop VEs with little effort even though it would be restricted to proprietary plugins to extend the framework's functionalities.

Besides the differences they have similarities as the complex architecture, which hide many implementation details, which could generate a total chaos when a developer needs to change or adapt core functionalities. This complexity delivers also a problem when installing or configuring the tools, either the dependencies are a problem to put together, or small details could compromise the whole application and preclude its usage. For instance, inVRs depends on many sets of files that point to other large sets of files and if one of the paths is wrong, it will not be possible to use the framework – what is very easy to happen since all the configuration is hand-made with no configuration tool aid.

However, when it comes to define the best, it is impossible to fully answer without knowing what is the problem to solve. Each one of the solutions presented before will deliver acceptable solutions to different – or even equal – sets of problems. What must be taken into consideration is what the solution proposes to do, if that will be enough to solve the problem, and even if that will be too much. For instance, VR Juggler has so many different components that even small problems tend to have a complex solution provided, sometimes with unrequired components.

³ Theoretically since everything that does not require direct altering the core engine is possible. For instance Vizard encapsulates the graphic engine but allows extending its functionalities through C++ code.

3 Easily configurable Virtual Environment

Even though there are tools and options to aid on the construction of a Virtual Environment, it still represents a challenge to laymen and non-experts. To put together all the different elements (such as models, shaders and 2D information), program the behaviors and interactions from scratch would discourage one to explore the possibilities that Virtual Reality may open. Also, it would be a “reinvention of the wheel” every time a VE is needed, to put everything together and recreate all the behaviors and interactions when those could be reduced as universal tasks and reutilized, most likely to be similar among them.

Given those problems, hereby pSIVE (platform for Setting up Interactive Virtual Environments) is proposed, a platform to provide an easily configurable Virtual Environment to virtual immersive visits. Allowing the creation of customizable environments with domain specific information attached but without requiring the mastery of programming languages neither theory related to Virtual Reality and such.

The pSIVE was designed to be an abstraction layer between a set of frameworks and libraries, chosen from those identified on chapter 2.3 and the final user. The following sections present its structure with details on implementation and usage as well as more information on tools and components that were used to provide the flexibility and robustness of the system. It was conceived to support the fast and easy creation of Virtual Environments with the possibility to attach information to different positions or elements of the environment to later be viewed and interacted with on a three dimensional space.

Different combinations and setups of tools were available to build pSIVE over, but it was important to choose the one that proved to be the most flexible and easier to work with, to provide abstraction enough to hide to the user the implementation details by accepting new devices ideally without need of changing the source code, and to keep

any required alteration punctual and with a minor effort. Before choosing frameworks, it was important to list a possible set of functionalities that were expected to be delivered as well as the different types of accepted input/output devices to be hooked on the platform.

3.1 System Requeriments




It was expected that the platform would provide an easy setup, to allow the user get it up and running with few steps. To achieve that, pSIVE had to be at the same time simple, to avoid creating too many details that might confuse and discourage the user of using the system, and flexible enough to be able to handle robustly the different type of inputs whether models, information or ways of interaction. The user was supposed to just choose the hardware to work with - HMD or desktop as output, different types of trackers, gamepads or even mice and keyboards as input - the models to be loaded and their position in the Virtual Environment as well as the information to be attached either to a single model or to a defined point in space (multiple points on a single model).

It was required as well to support an art pipeline to give freedom to the user to create a Virtual Environment composed by 3D models arranged on CAD (computer-aided design) like software, which means, no need to calculate and apply transformations such as translation and orientation of the model on the world manually.

The hardware to be handled by the platform on a first stage was required to ensure the coverage of, at least, part of the devices owned by the Institute of Electronic Engineering and Telematics of Aveiro (IEETA), which are listed on Table 2.

Table 2 List of equipment to support

Device	Description	Picture
Intersense Intertrax2	USB 3DOF Inertial Tracker	
Intersense InertiaCube3	USB 3DOF Inertial Tracker	
Intersense InertiaCube BT	Bluetooth 3DOF Inertial Tracker	
Nintendo Wii Remote	Bluetooth Gamepad with built-in accelerometer	
Razer Hydra	Gamepad and 6DOF Magnetic Tracker	

Device	Description	Picture
Microsoft Kinect	USB Optic Motion Sensor	
Wintracker	USB 6DOF Magnetic Tracker	
Virtual Realities VR2000	Head Mounted Display with built-in 3DOF Tracker	

With this, three major objectives guided the conception of the platform:

- Simple Configuration,
- Flexible Virtual Environment,
- Well Defined Graphics Design Pipeline.

To achieve the simple and flexible configuration, pSIVE must have runtime changeable configurations that can be defined once, and then just be reused by any other environment and be prepared to handle extensions with ease.

For the graphics design pipeline, the first step is to define which kind of information is needed and then seek for a solution that can easily export this information and build a layout of the Virtual Environment just by drag-and-drop elements on the model or part of it.

3.2 Architectural Decisions

How to interact? Which libraries or frameworks to use? Create with no framework but OpenGL? Those questions were to be answered before the implementation of pSIVE started, and this section explains the main decisions taken during its conception.

3.2.1 Frameworks and Modeling Tools

At first, the main doubt was to use a framework or build pSIVE from scratch. After the study of graphic engines and frameworks, it was opted to use a framework to save resources and to fasten the development, since the implementation was conducted by a single programmer.

From all engines and frameworks studied, VR Juggler (VRJ) was chosen since it had all the qualities needed to meet the project requirements and its community is still active developing new features or aiding to solve problems encountered by the users since its creation back in the late 90's. The project activity was the main point that made VRJ the chosen, as while it had a very active community, inVRs had practically no activity at all. Even emails sent to its supporters were not answered nor was the mailing list working.

As for the graphics engine, the choice was linked to the choice of VRJ as the base framework. Even though it supports a number of graphics engines, some are more developed, accepted and therefore, easy to work with. The choice was made between OpenSceneGraph and OpenSG. Both were easy-to-work-with solutions and had the required characteristics for pSIVE. But again, the activity of the project weighted on the decision, OpenSG is very outdated and lacks updates and improvements. OpenSceneGraph was chosen even though it is known that VRJ makes use of an outdated version of OSG Viewer, which does not have features that were implemented for the newer versions, as semi-automated event handling. This might seem a minor problem, however it causes a chain reaction, for instance the 2D Widgets for OSG are fully based on the newest viewer architecture requiring adjustments and adaptations to work with the older viewer. But since it is necessary to simulate the behavior of the new

viewer class, they run with problems and weird behavior, for instance the resizing according to the content of a widget.

Since OpenSceneGraph supports a vast list of formats, the graphics design pipeline might be solved by using many known software, for example Autodesk 3D Studio Max, Google SketchUp, Maya, Blender and any other software that exports models into a format accepted by OpenSceneGraph. Most companies that integrate PRODUTECH, and would benefit from pSIVE, reported to have Google SketchUp as their main modeling tool and given to the facts that a plugin⁴ is available to export elements created with SketchUp directly to OSG's native format and it is possible to use it with SketchUp's free version. Based on this information, SketchUp was selected as the main modeling software, nonetheless pSIVE can handle any modeling software as long as the exported format is supported.

The alternative of using a well-known game engine such as Ogre and Unity was also considered but in general they are more specialized with focus on game applications. More generic graphics engines, as scene graphs, can cover more easily a wide variety of applications, and adapt to different application types more easily.

3.2.2 Virtual Environment Interaction

With the frameworks and the graphics pipeline defined, it was time to focus on how someone would interact with the Virtual Environments created by the platform. The interaction methods were defined according to the sub-division of the universal tasks as proposed by Bowman (Bowman & Hodges, 1999).

The **navigation** task was divided into three subtasks: direction/target selection, velocity/acceleration selection and input conditions. To select the direction (how the user would indicate the direction of the motion or the final point of the travel), pSIVE would work with a gaze based steering, meaning that wherever the user is looking (or orientating the head) is the direction to go. The gaze steering is just a first implementation, but the system would also allow more navigation styles, for instance controlling the direction with a joystick instead of the head.

⁴ Plugin available at <https://github.com/rpavlik/sketchupToOSG>

The velocity or acceleration would be variable according to the user's will, starting at a medium speed and allowing increasing or decreasing it by pressing a button or pushing a joystick. Navigation requires at least a 3DOF tracker attached to the head, to track the movement direction and a tracker or gamepad with at least 6 buttons or 2 axis and 2 buttons considering the possibility of traveling onwards, left/right or backwards the desired direction plus selecting the speed.

Selection's subtasks are indication of object, indication to select, and feedback. By decomposing the selection the decisions were mainly how to indicate which objects to select and how to trigger the act of selecting something using some visual feedback. Among the methods explained on section 2.1.3 for performing a selection, the ray-cast method seemed the most adequate for a fast development and combined with the equipment available would be possible to vary the origin of the ray – or how to indicate the object. Firstly, with the selection being indicated with the head orientation, and centering the object on the screen. Secondly, with the beam originating from a virtual laser pointer, controlled by the user's hand. These adaptations of ray-cast are called Gaze-Based Selection and Laser Pointer Selection (Mine, 1995). The indication to select would be done the same way as travelling, by pressing a button. The requirements for the selection vary according to the technique used for the first step – the Gaze-Based Selection would require a 3DOF tracker, which could be the same used for navigation, plus one extra button on the tracker/gamepad on the hand to trigger the selection. As for the Laser Pointer, a 3DOF tracker attached to the hand could be used, but it would not provide positional information, so the origin of the laser would be fixed on the space. Using 6DOF trackers would be better since it allows the user to properly control the laser pointer as if it was really on the hand.

While navigation and selection seemed pretty clear in terms of where and how they would be used, **manipulation** required first the consideration of where and what would require it. According to the needs of the project, one must be able to interact with objects and access information and multimedia items previously attached to them, so the first elements that would require manipulation were the items of an object. For instance a video could be fixed in the world for someone to view, but at the same time it could be placed according to the position of the user's hand. Some three-dimensional elements of the scene also would be manipulable, for instance part of a model that could be moved to enhance the vision of its details. For a first version of pSIVE, the best way of manipulating objects would be by attaching them to the user's hand as if s/he is really

holding something and had a mapping of the hand to control both position and orientation of the object. The manipulation could also be performed by using a 3DOF tracker, but it would not provide the positional information, so a 6DOF tracker would be better used.

One suggested interaction set up would be a Nintendo Wii Remote to input data by pressing its buttons as shown on Image 23, along with a tracker on the HMD to provide the system with the positional/orientation information of the user's head. The number of buttons on the Wii Remote fit for most of the basic interaction techniques suggested for pSIVE, but it would not be the only option.

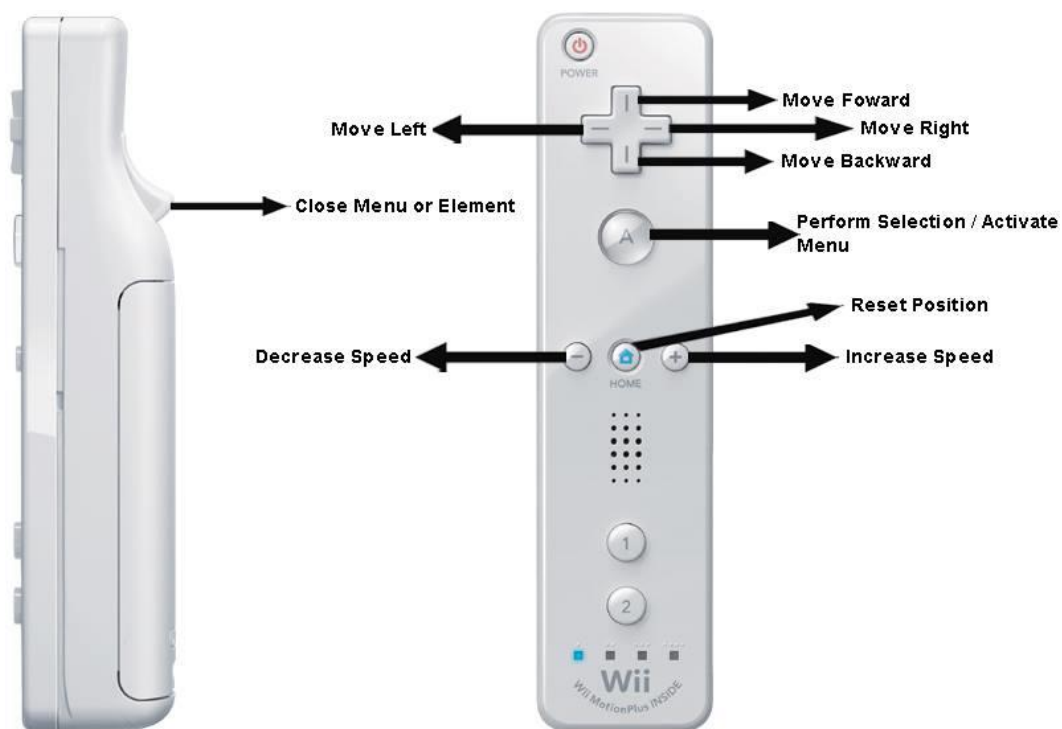


Image 23 Suggested button configuration on Wii Remote

The methods here presented are not the only option to interact with the system, as well as the types of devices recommended. The platform must be extensible to receive new styles without the need of much change – for instance a further version of pSIVE would allow the addition of a new navigation style without changing any of the other interaction tasks, and would be presented to the final user just as a new option on the configuration tool.

3.2.3 Configuration Interface

To put everything together pSIVE needed a simple configuration interface where even laymen could set up a Virtual Environment with the desired settings. This interface would allow the user to select which devices and interaction techniques are to be used, load 3D models previously built with SketchUp (or any other modeling tool) attaching information and multimedia files to them. Since some time would be necessary to build this configuration, it also should be able to export and import previously created designs.

3.3 System Development

The platform can be divided into three blocks, the modeling tool, the configuration tool, and the Virtual Environment itself. Image 24 shows the communication between the different blocks and its required functionalities.

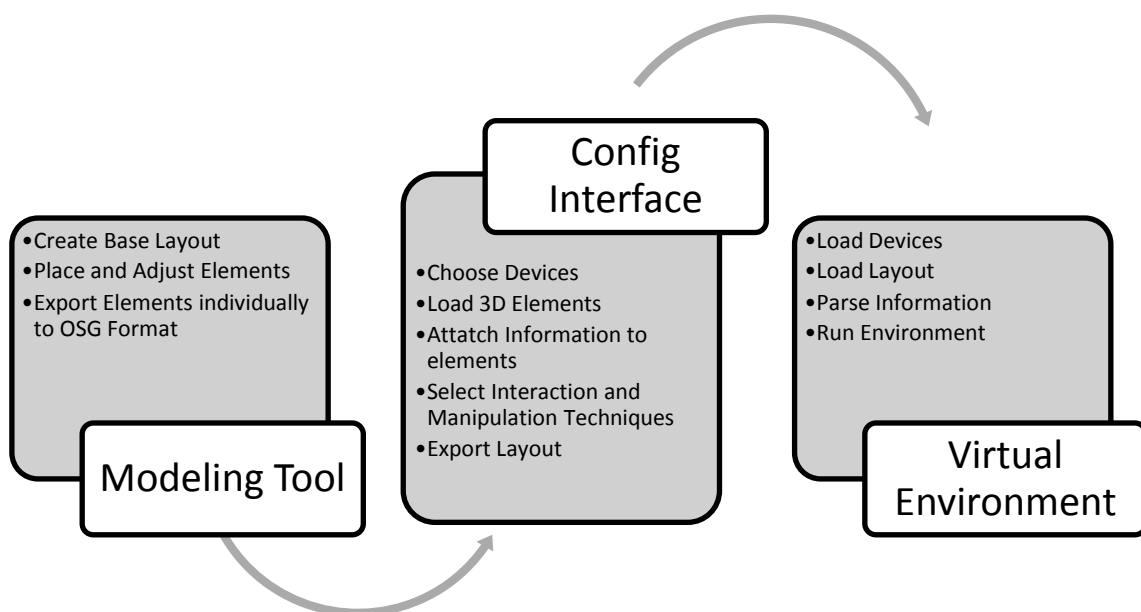


Image 24 pSIVE elements and their roles

Getting from the point where the modeling tool is already capable of exporting elements to a format that OpenSceneGraph can work with, this section will focus on the two other blocks.

Image 25 shows a short overview of pSIVE's structure. The Virtual Environment depends on a series of settings to be configured before running using the configuration tool. While the Virtual Environment is built on top of a group of frameworks, namely VR Juggler to handle input/output devices along with VRPN (that adds an extra set of supported devices to VR Juggler through network). VR Juggler also handles the window system creation and the system calls (to the operating system). OpenSceneGraph is the graphic rendering framework but most of its features are encapsulated by VR Juggler. The light blue elements are modules built using elements of both frameworks and manage the whole Virtual Environment.

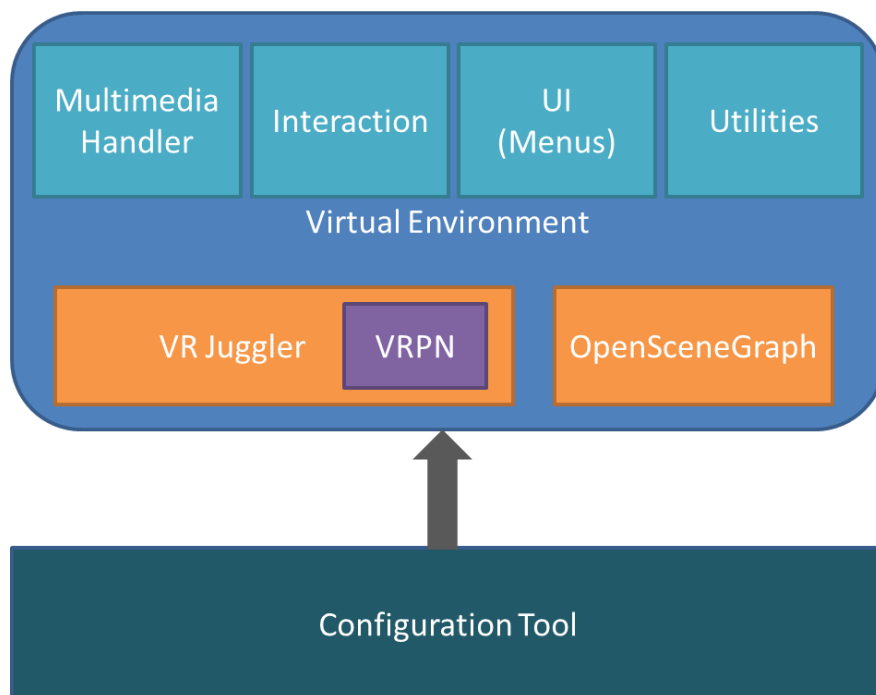


Image 25 Platform Overview

The Virtual Environment was developed using C++ since it is the native language of VR Juggler and OpenSceneGraph, and has also been widely accepted as a robust and efficient programming language. Even though the developer had short experience with this language, the wrapper of VR Juggler for JAVA is not well developed and still lacks functionalities, and thus the system was created with Microsoft™ Visual Studio 2010, however it is multiplatform, since all coding was designed to be OS independent.

To build the Configuration Tool, JAVA was the chosen language. Since it is the language on which the developer had most experience at, requiring less time to create it.

Externally to the developed software, pSIVE also required the preparation of the files that configure the hardware to be used. Please refer to the Annex 03 for more details on this topic.

3.3.1 Virtual Environment

The Virtual Environment for pSIVE can be compared to a blank canvas, with all tools available waiting to be painted. It had to be generic to the point where one could load a single model just to see and rotate it in a VE or load a whole factory complex with machinery on which the user could interact with a document describing the machines or watch a documentary on the maintenance of a part on the place where it is located. Not only the elements of the virtual world had to be generic, but also the device handling, the file formats to show as contextualized information as well as the interactions that trigger and control all of it. To do so, a group of modules was developed.

Their configuration as well as the whole environment characteristics required a layout design to be read by the program as well as for more features such as debugging or fine tunings. The simplest and quickest way to do so was by using the eXtensible Markup Language (XML). A pSIVE layout is based in two main elements: System and Data. While system properties are responsible for controlling environmental settings such as screen size and interaction styles, the data elements contain the location of files to be loaded and possible adjustments for the model positions and rotations. Image 26 shows a sample configuration file with just one 3D Model with information associated to it in the form of multimedia files and text.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<layout>
  <system>
    <config width="1024" height="768" selectionStyle="0"
      pdfAtHand="0" showWand="0" />
  </system>
  <data>
    <model filename="C:\dev\models\Casa\fridge.osg" label="Fridge"
      xRotate="0.0" yRotate="0.0" zRotate="0.0" xTranslate="0.0" yTranslate="0.0" zTranslate="0.0" context="true">
      <additionalData>
        <userData>
          <type>1</type>
          <label>Fridge Manual</label><!-- type 1 = pdf File -->
          <userContent>C:\dev\models\Casa\MetaData\FridgeManual.pdf</userContent>
        </userData>
        <userData>
          <type>0</type>
          <label>Fridge Content</label><!-- type 0 = plain text -->
          <userContent>Nothing, the fridge is empty</userContent>
        </userData>
        <userData>
          <type>2</type><!-- type 2 = video File -->
          <label>What is a Top-Mounted Fridge? Video</label>
          <userContent>C:\dev\models\Casa\MetaData\Videos\FridgeVideo.mp4.ffmpeg</userContent>
        </userData>
      </additionalData>
    </model>
  </data>
</layout>

```

Image 26 pSIVE Sample configuration file

3.3.1.1 Multimedia Module

To handle the input and exhibition of multiple formats of 3D models and its multimedia contents, the multimedia module takes advantage of the plugin architecture from OpenSceneGraph through the osgDB library, which loads dynamically the plugin needed for different kinds of formats. With this, many formats are possible of loading with generic code, requiring only the handling of its graphical output and behaviors. Besides the models, the currently supported formats for additional information are just 2D data: PDF Files, Videos and Plain Texts.

This module is composed by 5 major elements (Image 27): Extended Node, Video Player, PDF Reader, and Text Viewer.

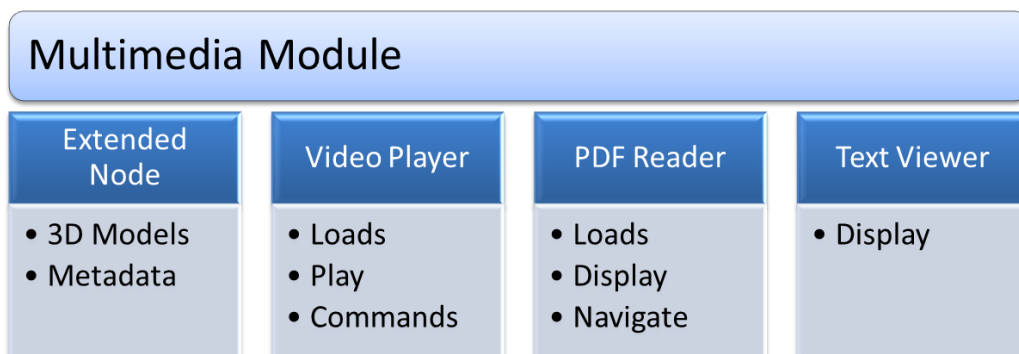


Image 27 Multimedia module architecture

The **Extended Node** was a method to keep the structure of OSG basic element (Node) to store the 3D model and to keep information on any multimedia element associated to it (Metadata). For the platform each model became an Extended Node and its loading was done by defining the location of the physical file on the hard disk and applying, when needed, the transformations to place it on the desired position or orientation. As for the Metadata, it contains the description of every external elements associated to the model, for instance the label and the location, provided with the configuration tool.

For the documents, the **PDF Reader** loads them from the metadata stored on the Extended Node but currently the implementation of a direct PDF loader is incomplete, and a pre-processing is required to transform each page into images. This step occurs hidden from the user: on the moment a layout is generated every PDF file is automatically converted to images and stored along with the layout. The display of a document is done by attaching each page as a texture of a plane on the space. The advantage of this method of loading the documents is the possibility of avoiding large files to be loaded during the running of the environment, impacting the performance – By converting the files from an external tool (configuration) it's possible to reduce the image quality/size to keep it compact and yet readable.

The user can control the reader by pressing buttons of the input device, the navigation controls are disabled and receive the function of changing pages (left/right), zoom in and out (forward/backward) and closing the reader. The reader stays partially below the line of sight (Seen on Image 28), requiring the user to look slightly down to read.



Image 28 pSIVE showing a PDF file

Similarly to the PDF reader, a **Video Player** was built within the multimedia module. The player displays the video on a surface that is placed right in front of the user, but still on the space, so the user can still look around and see the rest of the environment as shows Image 29. The loading of the video was done by using the `ffds` video library that is dynamically loaded by the `osgDB` library. Internally, it converts each video frame to a texture and places it on the plane in front of the user. Although, video plugin of OSG does not include audio, so the Video Player built had to manually get the audio from the video and play it synchronously.

Audio playback used the `SDL` (Simple DirectMedia Layer) library to provide access to audio devices without platform dependence. The current implementation of the Video Player does not provide spatial sound, so the sound is directly outputted to the speakers/headset as it is.

Just like the reader, the video player also takes input from the device chosen by the user to control the state of the video (play/pause) and to close the player and return to the menus or to traveling.



Image 29 pSIVE playing a video

Currently the display of plain text is done by simply mapping the desired text to the center of the screen fixed to the window, which means that no matter where the user looks, the text will stay in the same position until it is closed.

Yet the module currently addresses the basic requirements, it is expected to be extended to support more formats and methods to present them to the user.

3.3.1.2 Interaction

The module to handle interaction takes care of the whole set: navigation, selection and manipulation. Each element is responsible for handling the aspects of the interaction. The interaction module is divided according to the organization shown on Image 30.

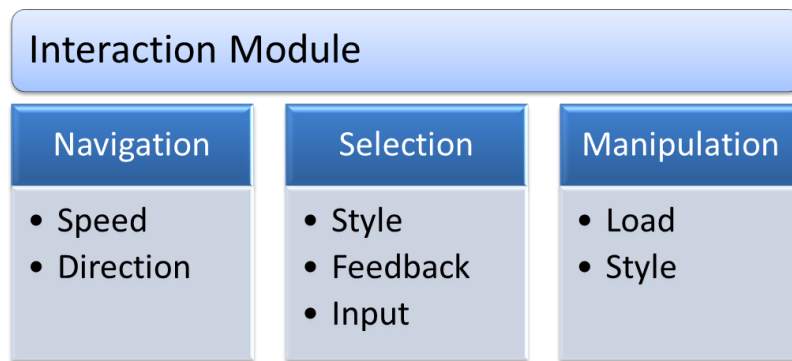


Image 30 Interaction Module

Navigation, as previously defined, is currently done according to the head direction. But the core of this element is built to allow further extension of its capabilities and addition of new navigation styles. The only required proprieties are the speed and the direction to translate the user to.

Currently there is no physics or limitations for navigation, so it is possible to fly around the environment without colliding to walls or be limited to a ground (no gravity).

Selection can be performed either by head orientation or laser pointer⁵. Depending on the style selected by the user, the interaction module provides adapted feedback to aid the user to know when an element contains information or when it is ready to be selected. In both methods, when the user selects an object with contents associated to it, a text box is showed on the center of the screen (Image 31) with the name or description of that object.

⁵ Using one or another selection style is up to the user to decide but the laser pointer selection requires a 6DOF tracker, if no tracker with positional data is found than pSIVE automatically switches to head orientation mode.

Also, to provide the user with additional feedback regarding if an object is correctly indicated for selection, when the selection mode is set to Laser Pointer the laser beam goes from red to green (Image 32) when the object is intercepted by it.

To perform the selection, the user has also to provide an input (currently a button press) to tell the environment that the indicated object is to be activated, this act triggers the Menu Module to start showing a menu with all the multimedia or text information attached to the model.



Image 31 User selecting a dinner table by head orientation inside pSIVE



Image 32 User selecting a dinner table by laser pointer inside pSIVE

Regarding **Manipulation**, the only interaction provided by the module is the control of documents positions and orientation by linking it to a tracker positioned on the user's hand. Yet for test purposes the manipulation of models was also attempted and is currently deactivated because of the lack of time to test and apply better

techniques. Image 33 shows a skull model⁶ that had its orientation controlled by a 3DOF tracker using pSIVE modules.



Image 33 Manipulation of a Skull Module using pSIVE modules

3.3.1.3 Menus

To provide a way for the user to access information inherent to virtual elements the Virtual Environment presents 2D adapted menus created accordingly to the elements that are available for a certain element. The structure of this module is as shown on Image 34.

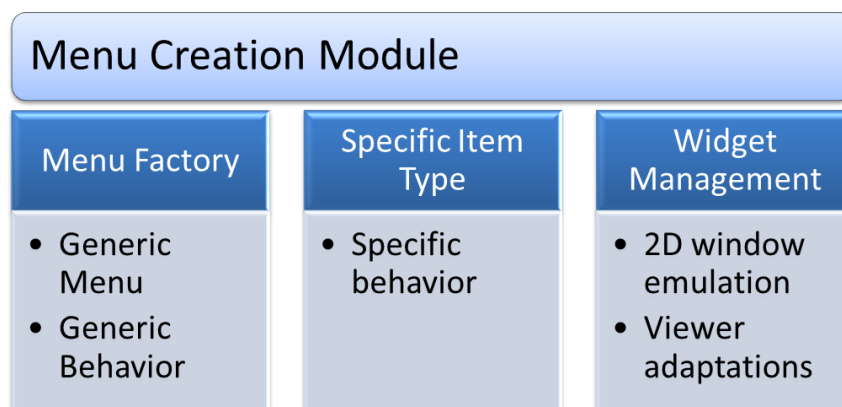


Image 34 Menu Module

⁶ Kindly provided by Dryas Arqueologia, Lda

Each Extended Node will be associated to a set of menu entries, according to its Metadata. The creation of the entries is done by the **Menu Factory** that will generate generic elements, to be extended according to its type. The behavior of each element defines how the multimedia module is triggered. Therefore each multimedia type requires a **Specific Item Type** defined into the Menu Module.

When an element that contains multimedia associated is selected, the menu module displays the previously generated items fixed to the screen. The **Widget Management** is responsible for the display and navigation on the items. The current version of the Menu Module uses a rectangular representation for each item, containing the label provided by the user during the creation of the layout file (Image 35). The controls for interacting with the items are simple: The controls for forward and backward change the highlighted (dark gray) item, a third button activate the item, triggering its behavior.



Image 35 A three item menu on an environment created with pSIVE

Currently the menu module is built over the osgWidget library, it adds support for 2D GUI windows and elements in the 3D world. Even though other options such as Qt gui elements were available, they were found to be complex to interact with from the Virtual Environment (computing intersections, modifying and visiting), while osgWidget is basically nodes and image derivatives, therefore well integrated with other scene objects.

However, the library depends on a virtual window created over the scene and events pre-defined to control the interaction with the widgets. Those elements were found to be not complete on the viewer required by VR Juggler, an early version of current OSG viewer class. Even though it was possible to emulate a 2D window, it lacks of the full functionalities required by `osgWidget`, restricting the menus of a single level (no nested menus or submenus) and with fixed content (not possible to resize the elements).

3.3.1.4 Utilities

The utilities module provides elements aid for the development of pSIVE and for the functioning. It contains 2 components (Image 36): Math and Text.

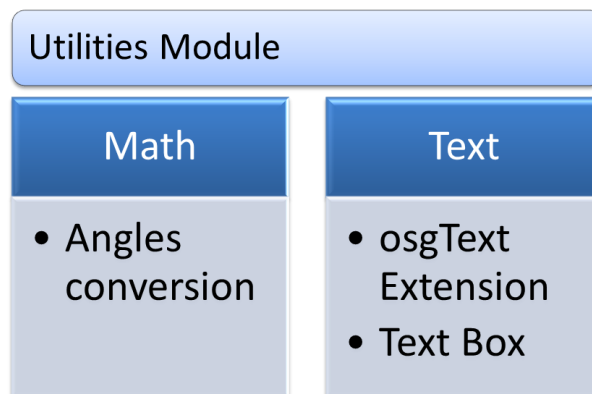


Image 36 Utilities Module

The **Math** component was required to provide functions that the default math library did not had. Specifically the conversion of rotation matrices to quaternion angles for the manipulation of objects – as mentioned before, this functionality is currently disabled but still present. The conversion was needed to simplify the communication between the VR Juggler math library (GMTL) and OSG math library. Since they are different and their conversion from one to another is manual, it was decided to avoid matrices where possible, so working with quaternions was a measure to keep a pattern for angles. This conversion is also not provided by both math libraries, requiring pSIVE to have custom math functions.

`osgText` is the library from OSG responsible for displaying text within the environment, but its functionalities were found rather complex to work with. **Text**

component provides methods that abstract the complexity of osgText, allowing the display of text simply by providing the position and the text to be displayed.

3.3.2 Configuration Tool

The configuration tool is a simple application to generate the XML file that will control the whole application without the need for the user to edit directly the xml file. It will allow users to generate the layout and tell VR Juggler which hardware to use on the Virtual Environment. Its interface was designed to rely on tabs, each one controlling a specific aspect of the system. The hardware Tab allows the user to choose from a list (previously defined by the developer) of equipment supported by pSIVE, dividing them into three classes: Head Tracking, Hand Tracking/Controller and Output. Models tab loads the 3D files giving the possibility of adding information to each file and adjust its position/orientation. The last tab controls the interaction styles to be used on the environment: Navigation speed, Selection style and Manipulation (of the documents). Image 37 presents some views of the configuration tool created.

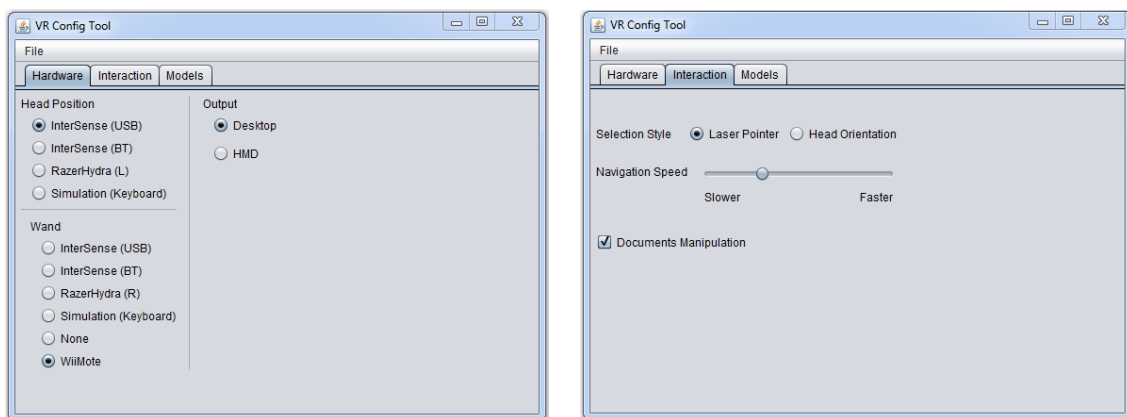


Image 37 pSIVE configuration tool

The list of devices to be used is sent to pSIVE transparently for the user. The configuration tool changes the files that define each device so the correct ones can be loaded by the platform. Those files are defined by VR Juggler, for it is the responsible for abstracting the input and output.

The current configuration tool lacks some flexibility since all options have to be previously programmed into it. A better approach is to adopt an architecture based on plugins, allowing adding new interaction styles and hardware without the need of altering the code. Also, due to the lack of time, the tool does not have an importing option to load previously created layouts.

4 User Evaluation

During the latest stages of development a study was proposed to assess the effectiveness and to find out which selection technique was better applied to different situation, given the importance of object selection on the context of the platform, and the possibility of adding diverse selection methods.

In a Virtual Environment where the selection can be only performed by the specific variations of ray-tracing used on pSIVE, no study is conclusive on which one is better – by direct comparison. For instance, the study conducted by Sanz (Sanz, 2011) that compares the head oriented selection versus the laser point selection but focusing on how they are affected by occlusion and showing how to minimize this specific problem by distorting the object or adding a virtual x-ray lens.

The evaluation performed revisits the experiment conducted by Bowman (Bowman, Johnson, & Hodges, 1999) on which volunteers performed the selection, with different techniques, of a highlighted object among a grid with nine cubes and moved it to place in a specific area. However, this evaluation considered only the selection step.

To assess the selection techniques, a user evaluation was conducted, having as test environment the pSIVE itself as a base, and extending its functionalities to record data and measures. Any device supported by pSIVE could be used to perform the evaluation, but the Razer Hydra was chosen as it is easy to operate and provides 6DOF allowing to easily emulate the natural act of pointing. Since the Hydra is composed of two controllers, one was placed on the back of each volunteer's head, to track its orientation, along with the head mounted display VR2000 since the built-in tracker is not yet supported neither by VR Juggler nor the VRPN. The other controller was given to the volunteer to hold for inputting commands to start the simulation and to trigger the selection. The second controller was also used as a laser pointing, by tracking the position and orientation of the hand.

The evaluation consisted of using the two variations of ray-tracing present in pSIVE to perform the selection of a particular object among a group of objects. As shown on Image 38, the user is presented a grid of blocks, each block 55cm tall, and prompted to select the green one. This work did not take into account the problem of occlusion, since all objects were visible and do not overlap.

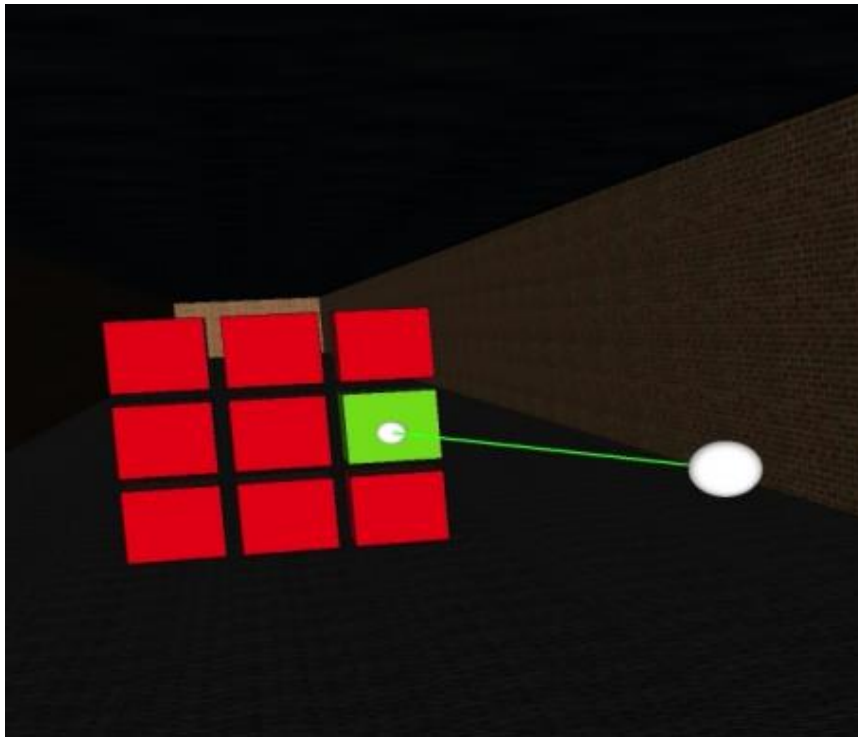


Image 38 Selection technique evaluation environment created with pSIVE

4.1.1 Hypothesis

The techniques present on pSIVE are two variations of ray-tracing, as mentioned before. These variations were widely discussed by Mine (Mine, 1995) who classified them as laser point selection and gaze selection. The concept is similar, to select an object just point it – but while gaze selection describes the act of indicating the object with the orientation of the head, laser point selection (LP) is controlled with the hand as if the user is holding a laser pointer and directed the beam to the desired object. Recently gaze selection refers to the properly gaze been used to indicate the object, by tracking the eyes of the user (Cournia, Smith, & Duchowski, 2003; Jimenez, Gutierrez, & Latorre, 2008). To avoid ambiguities the variation compared here will be referred to

as selection by head orientation (SHO). The main objective of this study is the direct comparison between LP and SHO to assess their adequacy to various situations.

Liang and Green (Liang & Green, 1994) were one of the first to implement the selection analogous to the manipulation of a laser pointer and stress that a known problem it has is the difficulty to select distant or smaller objects, since it requires a high angular precision. Thus, it is expected at first that for distant targets this selection will perform worse when compared to SHO.

The literature also shows that several authors conducted exhaustive tests in order to compare different selection techniques based on ray-tracing, however we do not find comparisons involving controlling the beam with the head orientation and having the same origin. Instead, some of them show techniques where the beam has the origin on the head but the direction is controlled with the hand, as shown by Sanz (Sanz, 2011). However these techniques are less common, since most of the comparisons are made between the laser pointer and the selection made from the eye direction (gaze).

From the analysis of related work and theoretical aspects, the following hypotheses were formulated about this experience:

- H01 – Less errors at long and medium range selections for SHO, in comparison to LP
- H02 – Higher average time taken to perform the selection for LP during the first selections (between 5 and 30 meters) given to the time that the user might need to get the hand in a comfortable position.
- H03 – The method with which the volunteers start won't interfere with the results.

4.1.2 Method

Different variables are used on the evaluation. The first is the position of the element of interest (green cube) on the grid, which is decided randomly after each step. The second is the distance between the grid and the volunteer, starting at 5 meters and increasing 5 meters each step until the final position 70 meters away from the user. These distances were selected to provide feedback for various scenarios, from the selection of close objects to a situation where the objects are smaller defaulting the selection. Also, different combinations of both variables allow the evaluation of cases in which, for instance, the object on the lower part of the grid is easier if it is close or distant.

The output variable (or dependent variable) considered was the performance of the volunteers with both techniques, assessing basically the number of mistakes and the time elapsed. Other data that could present itself interesting is the learning effect for the volunteer according to the initial method (SHO or LP), this being an important secondary variable.

Several measures were automatically recorded by the platform, corresponding to the user's performance: number of errors (one represents a selection performed on the wrong object), time elapsed from the activation of the test and the selection, the distance of the grid and the position of the correct object on each step. The method that the volunteers began with was also recorded (SHO or LP).

Besides quantitative aspects, any relevant additional information provided by the volunteers during the experiment was noted down by an observer, for instance personal comments/opinions, issues found and suggestions. A questionnaire was requested to be filled inquiring about the volunteer profile, its opinions on multiple aspects of the interaction with the test (easy to orientate, pleasant, etc...), the satisfaction rate for each selection method, the favorite method and any comment about difficulties or any other test related subject.

The participants were undergraduate and postgraduate students from courses related to computers and information system at the University of Aveiro. On total, the evaluation had the participation of 16 volunteers (14 male and 2 female) with ages between 19 and 26 years, without experience with Virtual Reality systems but, in 4 cases, experience with computer games (namely first person shooters). Half of the volunteers started with LP and the other half with SHO, however they did the experience with both methods. Corresponding to an experimental design within-groups (Dix, Finlay, Abowd, & Beale, 2003).

Previously to the experiment, the volunteers were presented with a brief explanation on what the evaluation consisted and what they were required to do. An additional training cycle was performed right before each selection cycle (from 5 to 70 meters) so the users could understand the functioning of the system with both selection styles, resulting in a total of 56 selections between training and evaluation, half for each method. The volunteers were responsible for the activation of tests at every step, giving them time to ask questions or make comments without interfering with the acquired measures.

4.1.3 Results

Two participants were unable to perform all selections, the first due to a technical problem during the training cycle that saved corrupted data, and the second due to the impossibility of using the Head Mounted Display while keeping the glasses on, and since he had myopia the volunteer decided not to complete the entire SHO cycle.

Regarding the results obtained, as shown on Image 39, for almost all distances (except one) SHO maintained fewer errors, therefore suggesting the confirmation of H01. However, it is worth noting the fact that, although the results obtained with both techniques are close, LP was better within the range from 5 to 30 meters and that the number of errors obtained with both techniques increased with distance as expected.

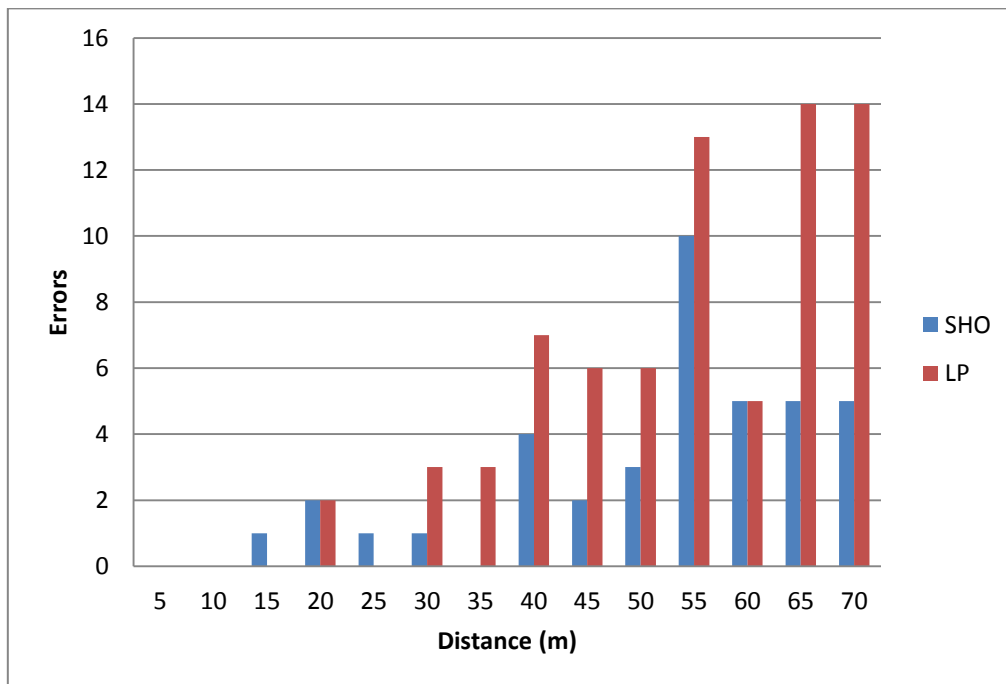


Image 39 Sum of errors by distance

As for the average elapsed time needed to perform the selection on each distance, Image 40 shows that the selection of LP has, in most instances, longer times. However, the original difference tends to soften in the range of 10 to 25 meters. As predicted by H02, the higher times (for LP) in the first selections were also confirmed and explained by the fact that users took some time to find a comfortable position for the hand before starting the selection. Also, this was reported by 8 of the 16 volunteers during the test.

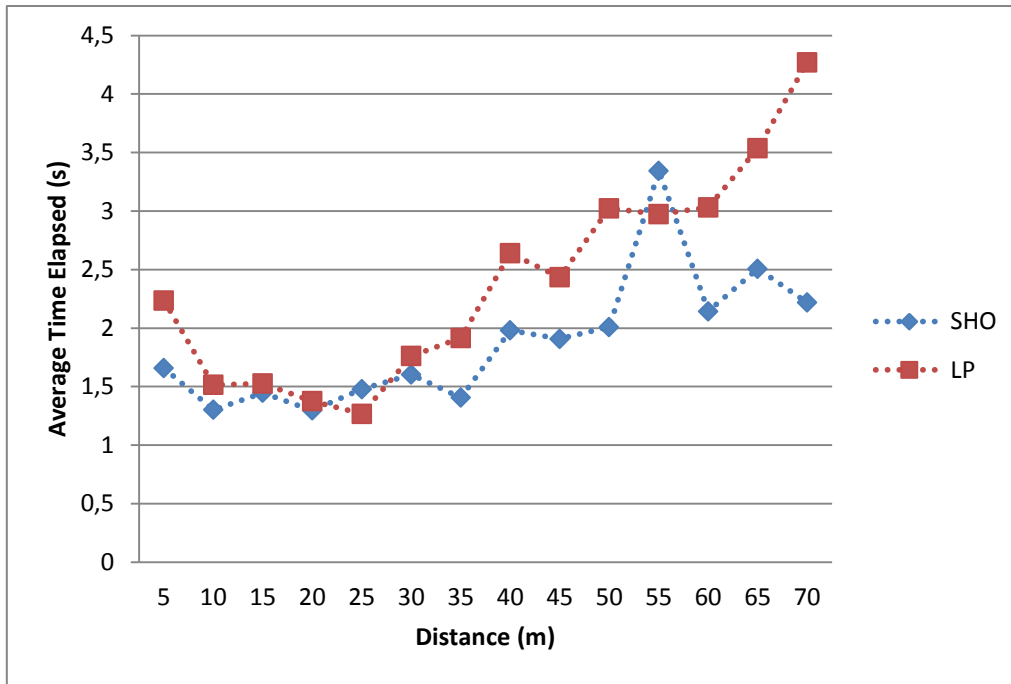


Image 40 Average elapsed time by distance

In contrast to what was supposed by H03, there was a notable difference in both time and the number of errors recorded for those who started using the LP method. Analyzing the results shown in Image 41, it was evident that the number of errors accounted for those who began with SHO and were currently using LP was approximately two times higher than those who started with LP and were currently using SHO. A possible explanation is the fact that both techniques require small and precise movements, which are more easily obtained when controlling the beam with the head. The change from fine movement (with head) for wider movement (hand) or vice versa reflected in the learning rate of the user. So those who started with PL are subjected to a type of control that requires more training to be used, as the users reported (see Table 3), might have focused more to understand how to interact with the system, which resulted in a performance improvement when using even SHO. Participants also indicated that in addition to getting used to the device, they also needed some time to realize what to do and how to do it .

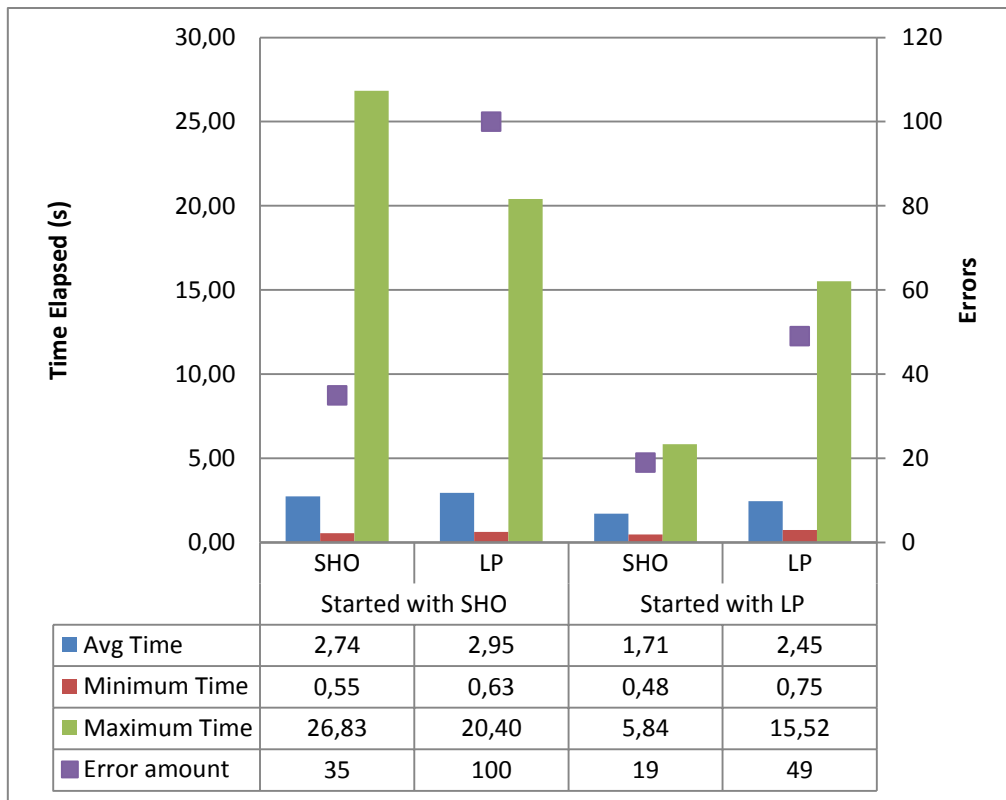


Image 41 Average elapsed time and errors according to the starter method

In comparison, it was noted that LP was worse independently of what was the initial method. These results are reflected on the opinions provided by volunteers. On a scale from 1 to 5, where 1 means strongly disagree and 5 strongly agree: it can be seen on Table 3 that despite the fact that both techniques are pleasant, LP presents irritant features and requires more training than SHO.

Table 3 Questionnaire results (median of each index)

	SHO	PL
Ease of Orientation	4	4
Pleasant	4	4
Annoying features	2	4
Intuitive	4	4
Requires Training	3	4
Useful for near Obj.	5	4
Useful for Distant Obj.	4	3
Satisfaction Rate	4	3

It is also noticed that SHO was the technique preferred by most of users and quantitative data show that it was also more efficient regardless of the position of the object. Additionally, it is worth highlighting that the fact that some participants

possessed gaming experience did not seem to influence the overall result, since there was no significant difference observed.

4.1.4 Conclusions

The results obtained indicate that the size of the object and the distance from the user influence the performance of the selection and present itself as one of major limitations of ray-tracing (Sanz, 2011); however, several techniques can be applied to improve it. For example, working on Fitts' Law and changing the size of the object or distorting it according to distance (Balakrishnan, 2004; Sanz, 2011; Teather & Stuerzlinger, 2011).

The selection by head orientation was more effective both according to quantitative results and the personal opinion of the volunteers, once it presented a lower error rate during most of the experiment (except for distances lower than 25 meters), however the results were very close and the learning rate afforded by laser pointer (LP) selection was significant and thus further research is required to a better understanding of what happened.

After the experiment, it was decided that the head selection should be the most adequate as a primary selection style since it was more precise and pleasant for the volunteers. However, when manipulation of 3D elements comes to mind, a combination of both techniques could be used; for instance the initial selection of an object or document inside the Virtual Environment could be achieved by SOC and any further interaction would be controlled by the LP (laser pointer), allowing 6DOF interaction.

5 Case study

Under the task A.2.3 from the PRODUTECH-PTI Project, a real case scenario was proposed. This task required pSIVE to allow the creation of interactive visits for marketing and virtual training for workers from Portuguese companies on a virtual version of their facilities.

One of these companies is TEGOPI - Metallurgy and heavy metalworking - specialized on the manufacturing of wind power towers. TEGOPI is located in Vila Nova de Gaia, Portugal and parts of its facilities were reproduced into pSIVE to validate the accomplishment of the objectives for the tasks A.2.3.

The following sections present the process of creation and configuration of pSIVE to run an interactive visit to the facility.

5.1.1 Creating layout

As previously explained, some of the companies involved in PRODUTECH already used SketchUp as a tool for modeling their equipment for simulations on the software like SIMIO. TEGOPI is one of those, and kindly provided the 3D models for the components. They were put together with SketchUp representing the real factory (Image 42).



Image 42 TEGOPI factory recreated on Google SketchUp

The virtual facility was created using a free version of Google SketchUp and exported to OpenSceneGraph's format. Currently the only way to add interactive information is by loading models one by one and indicating the information to load within configuration tool for pSIVE. That required each element that should contain information to be exported individually as a single file. To do so, the element should be selected from the overall design and exported individually – This will keep the positional and orientation information, so when different models are loaded on pSIVE, they will stay the same as they were on the original SketchUp model. Once the exportation of elements with contents associated was done, the rest of the layout can be exported as a single 3D model. Image 43 illustrates the process of selection and exportation of a single machine from the design.

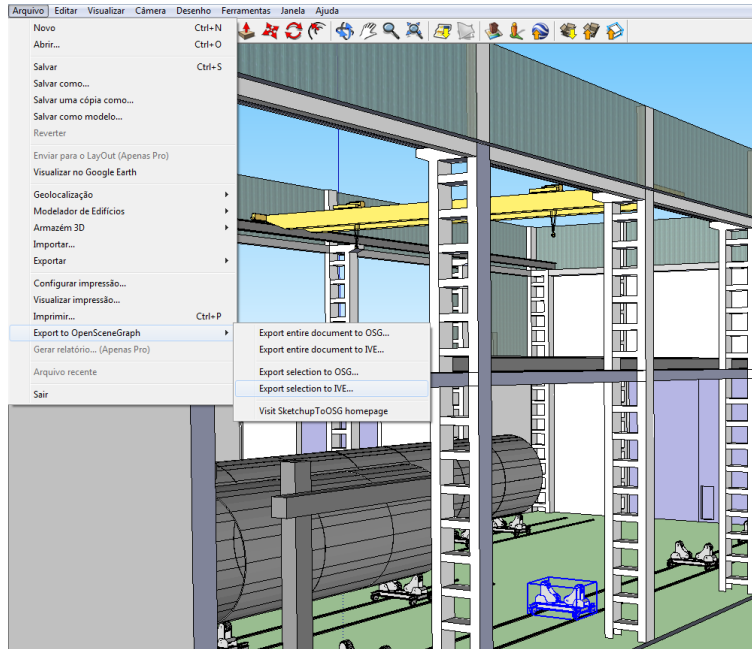


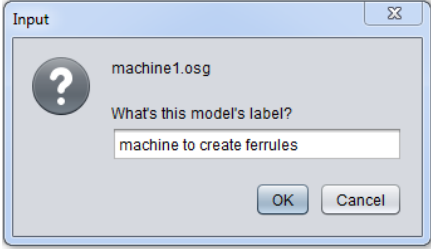
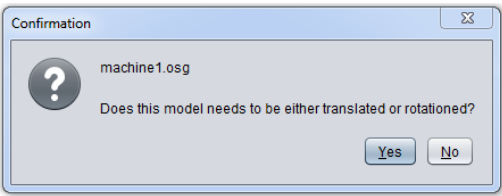
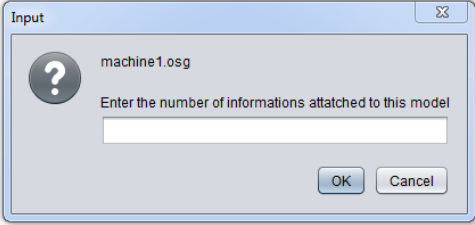
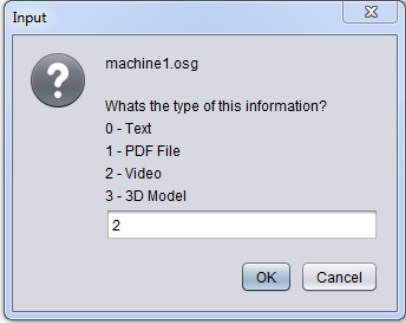
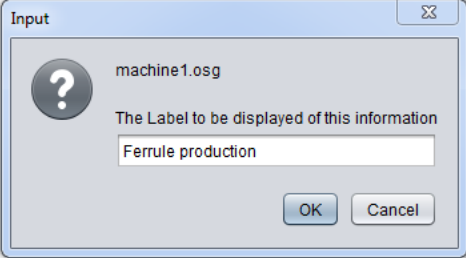
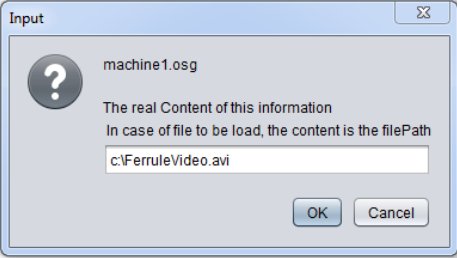
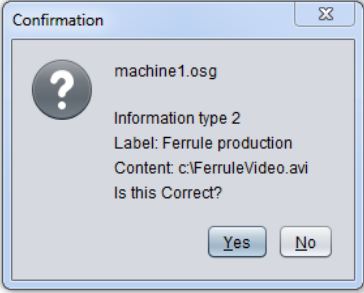
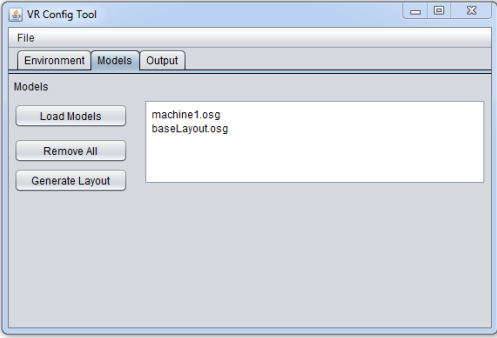
Image 43 Exporting a single machine from the virtual factory

With the 3D models ready, one must use the configuration tool to generate the file that controls and setup the environment. In this case, the interaction style was set to navigate and perform selections with the Razer Hydra (Laser pointer selection), and head tracking with Intersense Bluetooth.

As for the models, one could load them one by one or by adding multiple at a time. Table 4 shows the steps taken to load the models, along with a brief explanation.

Table 4 Configuration of TEGOPI layout

Step 1. Click on Load Models and select the models to be loaded (one or multiple)

 <p>Step 2. Each model will require a label</p>	 <p>Step 3. If the model needs to be translated or rotated, the tool will ask for the values (meters/degrees)</p>
 <p>Step 3. Inform the number of attachments. For each one, steps from 4 to 7 will be repeated</p>	 <p>Step 4. Indicate the attachment type</p>
 <p>Step 5. Give a label to the attachment (to be displayed on the menu)</p>	 <p>Step 6. If the attachment is a file, indicate the full file path, or else enter the text</p>
 <p>Step 7. Verify if the information is correct. If not, the application goes back to step 4.</p>	 <p>Step 8. After loading all models, click Generate Layout and save the file in the same folder as the pSIVE running script</p>

For demonstrational purposes the only element with data associated is the machine highlighted on Image 43, a ferrule machine – it creates ferrules from huge

metal plates. The remaining of the factory was exported as a single base element. Image 44 shows the created configuration file for pSIVE.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<layout>
  <system>
    <config width="1240" height="720" selectionStyle="0"
      pdfAtHand="0" showWand="0" />
  </system>
  <data>
    <model filename="C:\dev\DEMOS\VRJOSG\models\TEGOPI\baseLayout.osg" label="Base" xRotate="0.0" yRotate="0.0" zRotate="0.0"
      xTranslate="0.0" yTranslate="0.0" zTranslate="0.0" context="false">
      <userDatas>
        </userDatas>
      </model>
    <model filename="C:\dev\DEMOS\VRJOSG\models\TEGOPI\machine1.osg" label="Ferrule Machine" xRotate="0.0" yRotate="0.0" zRotate="0.0"
      xTranslate="0.0" yTranslate="0.0" zTranslate="0.0" context="true">
      <userDatas>
        <userData>
          <type>2</type>
          <label>Ferrule production</label>
          <userContent>c:\FerruleVideo.avi.ffmpeg</userContent>
        </userData>
      </userDatas>
    </model>
  </data>
</layout>
```

Image 44 Generated config. file

5.1.2 Running Application

After the configuration is done, it is necessary to make sure all devices are connected. Some of them require a VRPN server running to communicate with VR Juggler (which is the case of Razer Hydra). A pre-compiled version of the VRPN Server is available along with pSIVE. – More on the device configuration and setup is available in Annex 2 of this document.

From this scenario, the first step is to turn on the VRPN server. For the Hydra, the controllers must be placed on their base for the first calibration. After the detection and calibration, the VRPN server will show the message that everything is working properly (Image 45). The devices that are configured directly into VR Juggler do not need the VRPN server, they just have to be connected to the computer that will run the environment.

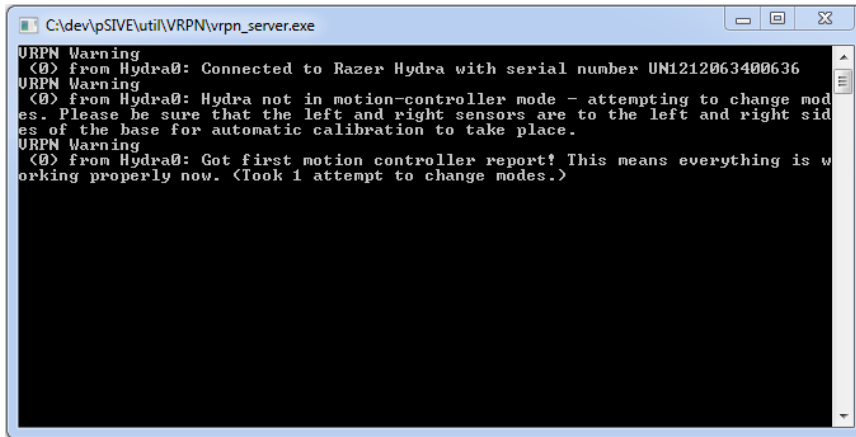


Image 45 Starting VRPN server

The next step is to run pSIVE’s startup script. This is where the devices are handled to the application, VR Juggler’s Gadgeteer will perform a quick verification to make sure they are working (Image 46), and the environment will start according to the settings chosen during the configuration step.

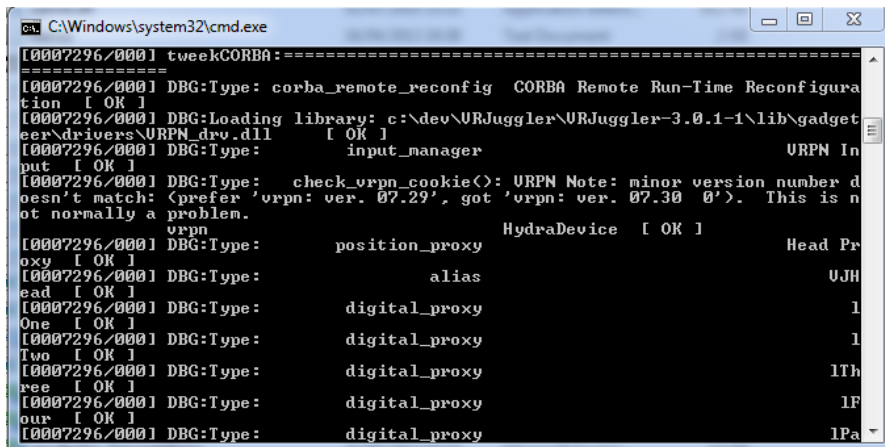


Image 46 Starting pSIVE Virtual Environment

The first view is the factory as a whole (Image 47), as created on Sketchup. The user is free to travel around it by using the Hydra joystick.



Image 47 Tegopi's Factory Overview inside pSIVE

From this point on, it is possible to interact with all elements that had multimedia files attached. For this example, one element with information was created. It is a ferrule machine, as mentioned previously. It is located on the area of the virtual factory without walls. Orientating the head towards it makes its label appear on the screen (Image 48). With the label it is possible to identify elements on the factory without the need of displaying the menu.



Image 48 Ferrule machine highlight inside pSIVE

With the machine on focus, pressing the activate button on the controller will display the menu options for that element (Image 49). In this case, it's possible to read

Tegopi's informative document, or watch two videos: one about the machine operation and the other about an equipment that is used to verify the consistency of the ferrules produced.



Image 49 Menu associated with the ferrule machine

For instance, a factory employer going under training could watch the video about the machine he is about to operate, as seen on Image 50, in an environment that will make him feel as if he was really on the factory.

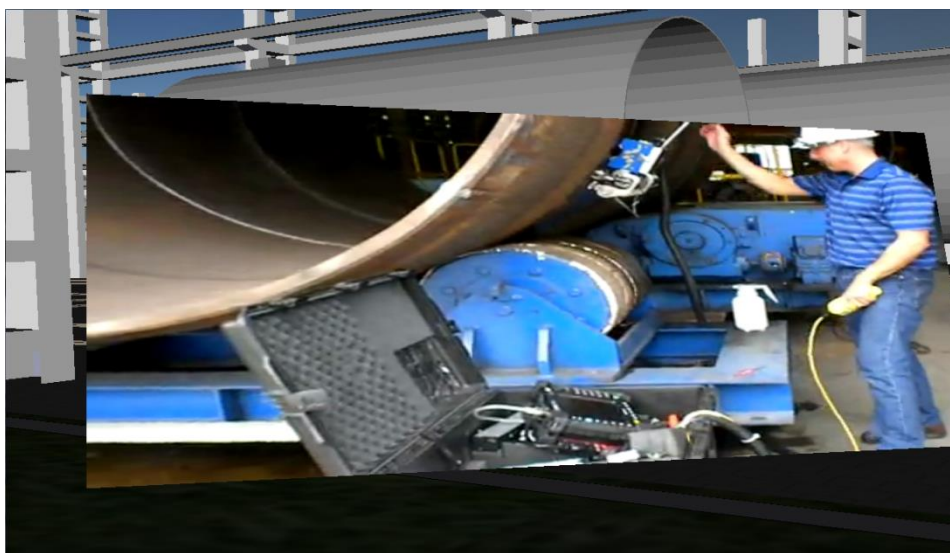


Image 50 Video about the ferrule machine operation

Another scenario corresponds to a virtual visit for marketing purposes; in this case after moving around the facilities it is possible to view a catalog (Image 51) on Steel Towers created by TEGOPI.



Image 51 Steel Towers catalog

5.1.3 Conclusions

From the moment the Sketchup model was putted together, it took, for the system developer, approximately 10 minutes to run a fully functional environment, the steps to export elements, prepare the configuration and run the environment are easily done once the user has a minimum experience with the platform. Since no technical knowledge, but the design of the models, is needed it is believed that with a brief explanation any one could setup an environment like this.

Also, during the creation and testing of demo environments, it was possible to identify some problems with pSIVE and some improvements were done in order to overcome them. On a first visit using the Selection by Head Orientation (SHO), there was no way to know if an object was on the center of the screen and therefore, selected. A cross was added on the screen to give feedback and aid the user centering the object.

Another problem found was the resolution and placement of documents that hindered the comprehension of its content. To ease this, the calculations for the document position were reviewed. However, the resolution is still a concern that is unissued, for low resolution screens and images the legibility is still hindered problem.

6 Conclusions and Future Work

This dissertation presented pSIVE, a platform that abstracts the usage of several frameworks to set up Virtual Environments, allowing even laymen (on subjects about computer programming and Virtual Reality) to work with it. Yet several frameworks already ease the creation of Virtual Environments none of them addressed the problem to include users from different areas without needing to interact directly with code or configuration files that were practically impossible to be used by non-experts.

Even though pSIVE is still on its first stages, it is already possible for a user with minimal information to run an interactive environment with few steps, as described in previous chapters.

The time was a constant issue to finish this work, along with the lack of experience of the developer with C++ and Virtual Reality. It was not possible to fully cover every aspect a Virtual Environment could have, or to offer a better user interface for setting up the files required by pSIVE.

The lack of feedback for the user to know which elements in the Virtual Environment contain information associated without selecting it is a known problem, but still not addressed. Some ideas have already been discussed for implementation, such as adding a glow effect to elements that have information, but still not implemented.

Some improvement was done during the experiment to assess the selection techniques and during the creation of TEGOPI's environment, since they revealed the downsides and problems of decisions and approaches adopted on the platform. With more real cases and with further usage more problems would keep showing as well as better ideas to improve the platform.

During a meeting, with representatives of various companies participant of PRODUTECH-PTI, that took place on July/2013, pSIVE was presented, in order to

demonstrate its capabilities and potential by allowing some of the meeting participants to interact with it, These participants gave very positive feedback on how it could already be used for marketing purposes and to allow virtual visits and training.

The current version of pSIVE is a prototype and still requires further refinement in its structure and modules. Furthermore, it is still limited and does not have the elements to provide a better and more natural usage of the Virtual Environment, for instance adding gravity and allowing interactions with different types of equipment (body tracking with Kinect, haptic feedback with Phantom, etc...). Therefore, it is clear that pSIVE still needs further work to be done to improve it and provide a better tuned version of the platform:

- Add physics and collision detection.
- Add spatial audio support and the possibility of adding an acoustic model for the environment so the sound would be as real as possible.
- Improve the existent interaction techniques or create new.
- Ease the addition of devices not yet listed on the configuration tool.
- Improve the creation and management of GUI elements, since the support provided by OpenSceneGraph isn't supported by VR Juggler. The possible creation of a generic Toolkit would not only benefit pSIVE, but all the VR Juggler and OpenSceneGraph community.

The following publications are resulting from the work presented in this thesis:

Souza D., Santos D., Dias P., Sousa Santos B. “Escolha de uma técnica de seleção para um ambiente virtual imersivo”. Atas da 5ª conferência Nacional sobre Interação - Interação 2013. Universidade de Trás-os-Montes e Alto Douro, Portugal, 7-8 Novembro 2013, pp.77-84.

Souza D., Santos D., Dias P., Sousa Santos B. “Platform for setting up interactive virtual environments”. To be published in Proceedings of the SPIE Electronic Imaging 2014 conference, San Francisco, February, 2014.

Souza D., Dias P., Sousa Santos B. “Choosing a selection technique for a virtual environment”. To be published in Proceedings of the 16th International Conference on Human-Computer Interaction, Crete, June, 2014.

7 Bibliography

- 3Dconnexion. (2013). SpaceMouse Wireless. Retrieved November 18, 2013, from <http://www.3dconnexion.com/index.php?id=352>
- Anthes, C., & Volkert, J. (2006). inVRs—A Framework for Building Interactive Networked Virtual Reality Systems. *High Performance Computing and Communications*, 894–904. Retrieved from <http://www.springerlink.com/index/33827N4429815N27.pdf>
- Balakrishnan, R. (2004). “Beating” Fitts’ law: virtual enhancements for pointing facilitation. *International Journal of Human-Computer Studies*, 61(6), 857–874. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S107158190400103X>
- Bastos, T., Raposo, A., & Gattas, M. (2005). Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual Baseados em Componentes Gráficos. *Anais do XXV Congresso da Sociedade Brasileira de Computação*. Retrieved from http://200.169.53.89/download/cd_congressos/2005/SBC_2005/pdf/arq0241.pdf
- Bastos, T., Silva, J., Raposo, R., & Gattass, M. (2004). Viral: um framework para o desenvolvimento de aplicacoes de realidade virtual. *Symposium on Virtual Reality*, VII, 51–62. Retrieved from http://www.tecgraf.puc-rio.br/~abraposo/pubs/svr2004/ViRAL_final.pdf
- Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., & Cruz-Neira, C. (2001). VR Juggler: a virtual platform for virtual reality application development. *Proceedings IEEE Virtual Reality 2001*, 89–96. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=913774>
- Bierbaum, Aron, & Hartling, P. (2005). Implementing immersive clustering with VR Juggler. *Computational Science and Its Applications*, 1119–1128. Retrieved from <http://www.springerlink.com/index/xl12urj4lea3ycyn.pdf>

- Bowman, D. A., & Coquillart, Sabine, Froehlich, Bernd, Hirose, M. (2008). 3D User Interfaces: New Directions and Perspectives. *IEEE Computer Graphics and Applications*, 20–36.
- Bowman, D. A., Johnson, D. B., & Hodges, L. F. (1999). Testbed evaluation of virtual environment interaction techniques. In *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '99* (pp. 26–33). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=323663.323667>
- Bowman, D. A., & McMahan, R. P. (2007). Virtual Reality: How Much Immersion Is Enough? *Computer*, 36–43. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287241
- Bowman, D. A., & Wingrave. (2001). Design and evaluation of menu systems for immersive virtual environments. In *Proceedings IEEE Virtual Reality 2001* (pp. 149–156). IEEE Comput. Soc. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=913781
- Bowman, D., & Hodges, L. (1999). Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *Journal of Visual Languages & Computing*, 37–53. Retrieved from <http://www.sciencedirect.com/science/article/pii/S1045926X98901112>
- Bowman, Kruijff, E., LaViola, J., & Poupyrev, I. (2004). *3D User Interfaces: Theory and Practice* (p. 512). Addison-Wesley Professional.
- Burdea, G. C., & Coiffet, P. (2003). *Virtual Reality Technology* (2nd ed.). New York, NY, USA: John Wiley & Sons, Inc.
- Burgess, D. a. (1992). Techniques for low cost spatial audio. *Proceedings of the 5th annual ACM symposium on User interface software and technology - UIST '92*, 53–59. Retrieved from <http://portal.acm.org/citation.cfm?doid=142621.142628>

- Costa, V., Pereira, J., & Dias, J. (2007). Tecnologias CAVE-HOLLOWSPACE para a Mina do Lousal. *15º Encontro Português de Computação*. Retrieved from <http://web.ist.utl.pt/~ist62669/uploads/Main/lousal.pdf>
- Cournia, N., Smith, J. D., & Duchowski, A. T. (2003). Gaze- vs. hand-based pointing in virtual environments. In *CHI '03 extended abstracts on Human factors in computing systems - CHI '03* (p. 772). New York, New York, USA: ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=765891.765982>
- Craig, A., Sherman, W. R., & Will, J. D. (2009). *Developing Virtual Reality Applications: Foundations of Effective Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Dassault Systèmes. (2012). 3DVIA Virtools - Dassault Systèmes. Retrieved October 17, 2012, from <http://www.3ds.com/products/3dvia/3dvia-virtools/>
- Dix, A., Finlay, J. E., Abowd, G. D., & Beale, R. (2003). *Human-Computer Interaction (3rd Edition)* (p. 834). Prentice Hall.
- Fakespace. (2004). Tool. Retrieved November 18, 2013, from <http://www.fakespacelabs.com/tools.html>
- Fowler, R., Carrillo, M., Huerta, R., & Fowler, W. (n.d.). Designing a Low Cost Immersive Environment System Twenty Years after the First CAVE. *elrond.informatik.tu-freiberg.de*. Retrieved from <http://elrond.informatik.tu-freiberg.de/papers/WorldComp2012/CGV2636.pdf>
- Frohlich, B., Hochstrate, J., Kulik, A., & Huckauf, A. (2006). On 3D input devices. *IEEE Computer Graphics and Applications*, 15–19. doi:10.1109/MCG.2006.45
- Gadgeteer Device Driver Authoring Guide. (2010). Retrieved December 04, 2012, from <http://vrjuggler.org/docs/gadgeteer/2.0/device.driver.guide/device.driver.guide.html>
- Geomagic. (2013). Geomagic Phantom Overview. Retrieved October 16, 2013, from <http://geomagic.com/en/products/phantom-premium/overview/>

- Gutiérrez, M. A. A., Vexo, F., & Thalmann, D. (2008). *Stepping into Virtual Reality* (p. 214). London: Springer London. Retrieved from <http://www.springerlink.com/index/10.1007/978-1-84800-117-6>
- Henning, M. (2006). The rise and fall of CORBA. *Queue*, (June). Retrieved from <http://dl.acm.org/citation.cfm?id=1142044>
- Hix, D., & Hartson, H. R. (1993). *Developing User Interfaces: Ensuring Usability Through Product & Process* (Wiley) (p. 416). Wiley.
- II, R. T., Hudson, T., & Seeger, A. (2001). VRPN: a device-independent, network-transparent VR peripheral system. *ACM Symposium on Virtual Reality Software & Technology*. Retrieved from <http://dl.acm.org/citation.cfm?id=505019>
- InterSense. (2013). InterSense | Precision Motion Tracking Solutions | InertiaCube3™. Retrieved November 18, 2013, from <http://www.intersense.com/pages/18/11/>
- Iwata, H., Yano, H., Uemura, T., & Moriya, T. (2004). Food simulator: a haptic interface for biting. In *IEEE Virtual Reality 2004* (pp. 51–57). IEEE. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1310055>
- Jimenez, J., Gutierrez, D., & Latorre, P. (2008). Gaze-based Interaction for Virtual Environments. *Journal of Universal Computer Science*, 14(19), 3085–3098.
- Just, C., & Bierbaum, A. (1998). Vr juggler: A framework for virtual reality development. *2nd Immersive Projection ...*, 1–8. Retrieved from <http://www.cvmt.aau.dk/education/teaching/e03/cvg9/vr/vr.juggler.framework.for.vr.dev.1998.IPT98.pdf>
- Kelso, J., Arsenault, L. E., Satterfield, S. G., & Kriz, R. D. (2002). DIVERSE: a framework for building extensible and reconfigurable device independent virtual environments. *Proceedings IEEE Virtual Reality 2002*, 183–190. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=996521>
- Kim, T. (2005). Graphical Menus for Virtual Environments. Retrieved from http://www.uni-koblenz.de/~cg/Studienarbeiten/SA_TaekBongKim.pdf

- Kuck, R., Wind, J., Riege, K., & Bogen, M. (2008). Improving the avango vr/ar framework: Lessons learned. *Workshop Virtuelle und Erweiterte ...*. Retrieved from http://www.avango.org/raw-attachment/wiki/Res/Improving_the_AVANGO_VR-AR_Framework--Lessons_Learned.pdf
- Liang, J., & Green, M. (1994). JDCAD: A highly interactive 3D modeling system. *Computers & Graphics*, 18(4), 499–506. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/0097849394900620>
- Matsukura, H., Yoneda, T., & Ishida, H. (2013). Smelling screen: development and evaluation of an olfactory display system for presenting a virtual odor source. *IEEE transactions on visualization and computer graphics*, 606–15. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6479189
- Melin, E., & Allard, J. (2002). Net Juggler and SoftGenLock : Running VR Juggler with Active Stereo and Multiple Displays on a Commodity Component Cluster.
- Mine, M. (1995). Virtual environment interaction techniques. *UNC Chapel Hill computer science technical report*. Retrieved from http://lsc.univ-evry.fr/~davesne/ens/pub/virtual_environment_interaction_techniqu_129302.pdf
- Mousavi, M., Faieza, A., & Ismail, N. (2011). Selection on Appropriate Departments for Virtual Reality Implementation in Malaysian Automotive Manufacturing Industry. *Proceeding of International Conference on Sociality and Economics Development*, 10, 34–37.
- OculusVR. (2013). Oculus Rift - Virtual Reality Headset for 3D Gaming. Retrieved October 16, 2013, from <http://www.oculusvr.com/>
- Pavlik, R. a., & Vance, J. M. (2012). VR JuggLua: A framework for VR applications combining Lua, OpenSceneGraph, and VR Juggler. *2012 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 29–35. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6231166>

- Razer. (2013). Enter the Hydra. Retrieved November 20, 2013, from <http://www.razerzone.com/minisite/hydra>
- Sanz, F. A. (2011). *Pointing facilitation techniques for 3d object selection on virtual environments*. Retrieved from <http://www.tdx.cat/handle/10803/33383>
- Schroeder, W., Martin, K., & Lorensen, B. (2006). *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics* (4th ed., p. 528). Kitware.
- Shreiner, D., & Group, T. K. O. A. R. B. W. (2009). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1* (7th ed.). Addison-Wesley Professional.
- SouVR. (2013). Logitech Head Tracker Wide Band SouVR. Retrieved November 18, 2013, from <http://en.souvr.com/product/200712/272.html>
- Teather, R. J., & Stuerzlinger, W. (2011). Pointing at 3D targets in a stereo head-tracked virtual environment. *2011 IEEE Symposium on 3D User Interfaces (3DUI)*, 87–94. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5759222>
- Teixeira, L., Trindade, D., Loaiza, M., Carvalho, F. G. De, Raposo, A., & Santos, I. (2012). A VR Framework for Desktop Applications. *2012 14th Symposium on Virtual and Augmented Reality*, 10–17. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6297555>
- Vicon. (2013). Bonita. Affordable motion capture for any application. Retrieved November 18, 2013, from <http://www.vicon.com/>
- Voß, G., Behr, J., Reiners, D., & Roth, M. (2002). A multi-thread safe foundation for scene graphs and its extension to clusters. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (pp. 33–37). Eurographics Association. Retrieved from <http://dl.acm.org/citation.cfm?id=569673.569679>
- VR Juggler Website. (2012). Features. Retrieved November 18, 2013, from <http://vrjuggler.org/features.php>

Wang, F. (2010). Research on Virtual Reality Based on EON Studio. *2010 Fourth International Conference on Genetic and Evolutionary Computing*, 558–561. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5715493>

Wang, R., & Qian, X. (2010). *Openscenegraph 3.0: Beginner's Guide*. Birmingham, UK: Packt Publishing.

WorldViz. (2012). Vizard VR Software Toolkit. Retrieved November 16, 2012, from <http://www.worldviz.com/products/vizard>

Annex 01 – VR Juggler 3.x Basic Installation Guide

Mostly the needed information will be available on the user guides provided on VR Juggler’s website. This document stands as a step by step guide to configure VR Juggler to aid a rapid setup performed by anyone, either by choosing to build it or to use a pre-built version under Windows.

All information hereby presented is present in a detailed form on VR Juggler’s Guides, on its website: <http://vrjuggler.org>.

Building from Source

Available at <https://code.google.com/p/vrjuggler/>

First of all, it is mandatory to have several dependencies on which VR Juggler relies on (Table 5). Unfortunately building VR Juggler can prove itself slow, requiring a bit of knowledge on building third-part code and patience. There are two groups of dependencies, the required and the partially required. As the name says, VR Juggler needs the required to build its core functions at least. The other group is ‘Partially’ required because some components of VR Juggler Suit may rely on them, but they are not mandatory.

For more detailed information please refer to <https://code.google.com/p/vrjuggler/wiki/BuildingFromSvn>.

Table 5 List of Libraries required by VR Juggler

Name	Description/Information	Available at
Required Packages		
Python	Optional – Used to run the configuration tool to help setting up the environment variables	http://www.python.org/getit/
CppDOM	XML parser – The project is stalled and had known issues to build under windows environments. A custom	https://github.com/rpavlik/cppdom

Name	Description/Information	Available at
	branch is maintained by Ryan Pavlik to add CMake Support.	
Boost 1.34+	C++ library providing many powerful utility classes and libraries – Tested with boost 1.46. It's possible to compile Boost from source but windows visual studio builds up to VS2010 are made available by BoostPro ⁷	http://www.boost.org/ http://www.boostpro.com/download/ x64 of versions 1.4x http://www.airesoft.co.uk/boostlibs
GMTL	A generic math library that makes use of C++ templates and STL paradigms – No compilation needed. GMTL is comprised of header files	http://ggt.sourceforge.net/
Partially Required Packages		
Java Developer Kit	The JavaSE SDK (or JDK) is used to compile all the Java code used in the Juggler Project. Without it, none of the Java code can be built. Requires version 1.4 or newer.	http://www.oracle.com/technetwork/java/javase/downloads/index.html
omniORB	A C++ implementation of CORBA 2.3, required for the Tweek C++ API.	http://omniorb.sourceforge.net/

⁷ BoostPro will stop supporting the free installers soon, but up to the time of this writing they were still active, and a message by BoostPro indicates that a community of users will keep them alive.

Name	Description/Information	Available at
Doozer		
VRPN	Virtual Reality Peripheral Network. Implements a network-transparent interface between application programs and the set of physical devices (tracker, etc.)	http://www.cs.unc.edu/Research/vrpn/obtaining_vrpn.html
OpenAL SDK + ALUT SDK		http://connect.creativelabs.com/openal/Downloads/Forms/AllItems.aspx
SDL	Simple DirectMedia Layer	http://www.libsdl.org/
DirectX SDK		http://www.microsoft.com/en-us/download/details.aspx?id=6812

Once the dependencies are ready, it is simple:

1. Chose a version of the source. It could be either a stable release or a developer version from GIT repository.
 - a. If it is from a stable release, download the zipped file and extract it
 - b. If it is from the GIT, make sure you have GIT installed. Find an appropriate folder and execute:
 - i. git clone <https://code.google.com/p/vrjuggler/>
2. Browse the folder and run the application called build_windows.py. It will prepare the environment and guide the user setting the environment variables required to build VR Juggler.
 - a. Set the install folder and point the dependency folders according to the console output. Once the input is finished it will launch Visual Studio.
 - b. If any error occurs, either typo or something else, simply delete the file “options.cache” so it can start over.
 - c. After launching Visual Studio, it will prompt to install VR Juggler. This must wait until the Step 3 is complete.

3. On Visual Studio, everything is ready, under the menu build, select Batch Build, check the wanted setup and hit Build. Some projects may fail due to missing include files. This is normal, since not all of them are mandatory.
 - a. After Building, Return to the Console of build_windows.py and continue with the installation by entering “y” or just hitting Enter.
 - b. Input Y again so it can install all dependencies as well.
 - c. Steps 3.a and 3.b are optional; the installation is a mere copy of all files to a specific directory selected by the user on the first steps of the environment setup.
4. Proceed to the **First Steps** part of this annex

Pre-Compiled Version

Available at <http://www.danilo-souza.net/VRJuggler-3.0.1-1.rar>

The compressed file contains everything is needed to run VR Juggler Applications, with and all the dependencies as well. Simply extract the files. The content is divided into VR Juggler which, as the name says, is the actual VR Juggler Libraries and configuration files. And the VR Juggler Deps which is the folder containing all the dependencies.

First Steps

Sample projects available with the package above or with the source code

To start working with VR Juggler the step is to set up the environment variables to point to its files and dependencies. The variables are:

- VJ_BASE_DIR – The folder of VR Juggler Files
 - ie.: C:\dev\VRJuggler\VRJuggler-3.0.1-1
- VJ_DEPS_DIR – The dependencies folder
 - ie.: C:\dev\VRJuggler\VRJuggler-3.0.1-1-deps
- VJ_CFG_PATH – The path to the configuration files located within the installation folder.

- ie.: %VJ_BASE_DIR%\share\vrjuggler\data\configFiles
- VJPATH – the path to all binaries of VR Juggler and its plugins
 - ie.:
 - %VJ_BASE_DIR%\lib;%VJ_DEPS_DIR%\bin;%VJ_DEPS_DIR%\lib;
 - %VJ_BASE_DIR%\lib\gadgeteer\drivers;%VJ_BASE_DIR%\lib\gadgeteer\plugins;%VJ_BASE_DIR%\lib\jccl\plugins;%VJ_BASE_DIR%\lib\vrjuggler\plugins
- Finally Add “%VJ_BASE_DIR%\lib;%VJ_DEPS_DIR%\lib” to Path

Instead of creating a new project and setting everything up, it is much easier to adapt an existing project to the needs of the user. Along with the VR Juggler Suite, many different sample projects are available, one for each graphics engine that is compatible with it.

In case of using the pre-compiled version provided in the beginning of this guide, a solution for Microsoft Visual Studio 2010 is available on the folder “samples”. Otherwise, it is necessary to use cmake to configure the source code and generate the solution file.

Some samples were designed as a starting point for VR Juggler. As recommended on VR Juggler’s Getting Started Guide the following projects are considered as for starters:

- simpleInput: No graphics rendered, this project demonstrate how to obtain input from the devices
- simpleApp: OpenGL application that draws a cube in space
- contexApp: Extends simpleApp by allowing interaction with the cube through a tracker.
- MPApp: OpenGL application to demonstrate multi-processing in VR Juggler.

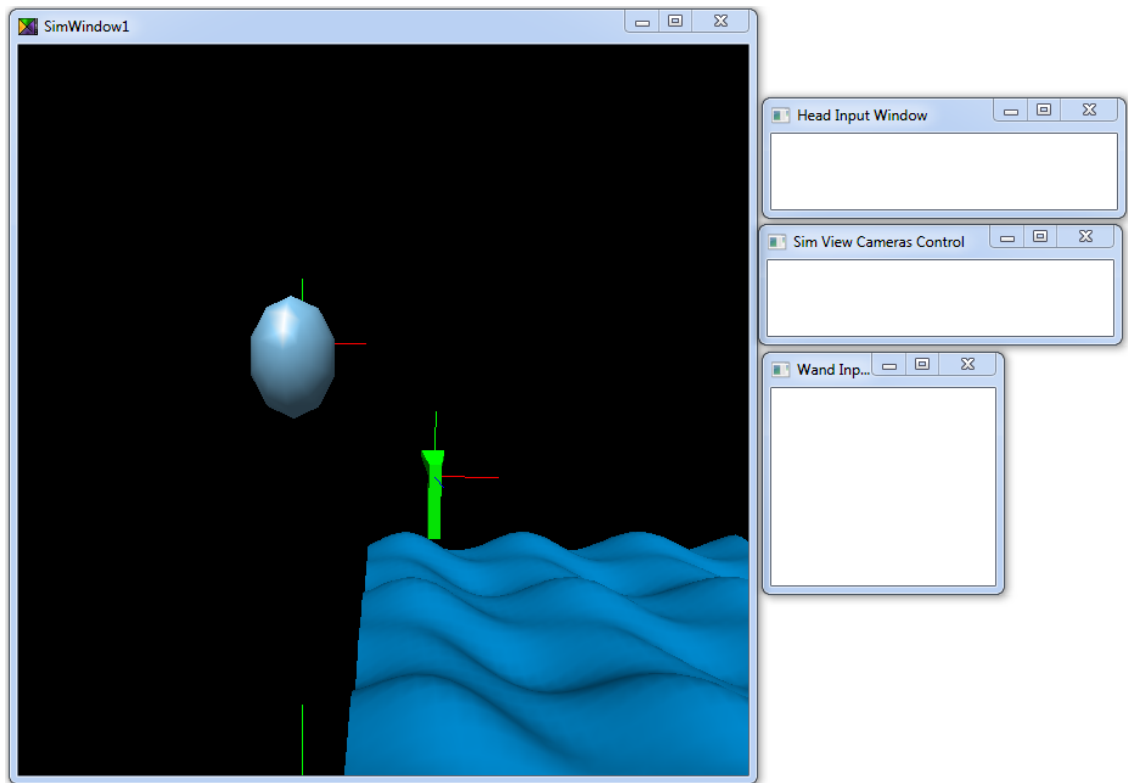
These projects can be compiled right away, and as long as the environment variables are set, no problem will happen.

To run an application it is necessary to tell it the configuration files of the devices (input and output) that are going to be used (A brief explanation on the device setup are given on Annex 03 – Please refer to it before running the applications).

Some sample simulator configuration files are available along with VR Juggler, they simulate the behavior of both input and output devices through mouse and keyboard for input and a single screen as output of even 6 displays systems.

An interesting example is the “sim.wand.mixin.jconf”, that simulate a tracker with positional data and several buttons along with “sim.base.jconf” to provide a viewport for the rendering to occur along with simulated head tracking by using the keyboard.

Once the compilation of the previously mentioned projects was completed successfully, run the application “MPApp”, using as configuration the both files referred before. To do so, type “MPApp.exe sim.wand.mixin.jconf sim.base.jconf” in a command prompt on the build destination folder. The result should be as follow:



The simulator window will show a moving water-like element and two simulations of head and wand. By pressing Z and X the wand will tilt left or right and the input windows will grab the pointer and use it to simulate a positional tracker.

Annex 02 – VRPN Guide

The VRPN is available

@

http://www.cs.unc.edu/Research/vrpn/obtaining_vrpn.html

VRPN stands for Virtual Reality Peripheral Network. It is a free cross-platform open source tool that handles many devices, usually used for Virtual Reality Systems. It is used by many Virtual Reality software, both commercial and free solutions. It acts as a server, enabling the access to hardware through an abstraction layer that standardizes their data. Any device made available through VRPN will be outputted as a generic hardware that can contain three types of data:

- Tracker – Any device/module that contains position and orientation
- Analog – Any type of axis (Joysticks for instance)
- Button – Any type of binary button

For instance, each Razer Hydra gamepad has a Tracker, 2 analog channels (x and y axis for the joystick) and a set of buttons. The current supported hardware as well as technical information and guides are available on its web-site: <http://www.cs.unc.edu/Research/vrpn/index.html>.

This guide will exemplify the configuration and connection to obtain data from a mouse using a pre-compiled VRPN Server for Microsoft Windows⁸. Other devices follow the same idea is mostly the same, however each device will have its characteristics and sometimes different configuration parameters.

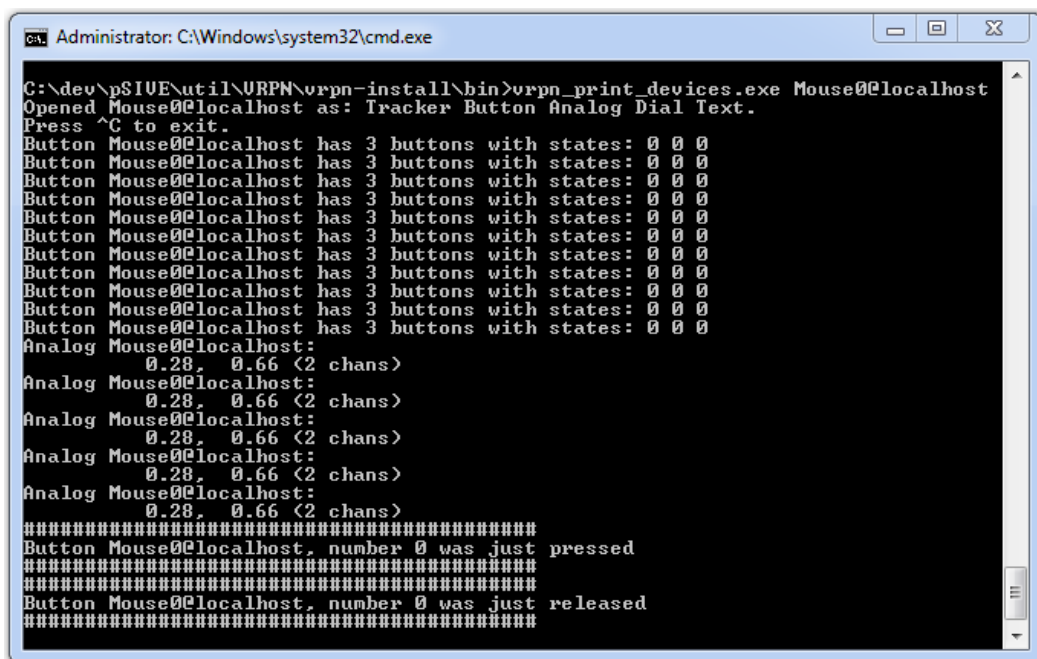
For the VRPN a mouse is composed of 2 analogs and 3 buttons. Each device is defined into the ‘vrpn.cfg’ file that is inside the “bin” folder, along with the server’s executable. The configuration file contains also short instructions for using the different devices supported by VRPN. To add the mouse locate the line “#vrpn_Mouse Mouse0” and uncomment it by removing the “#” symbol. “vrpn_Mouse” is the device identifier for the server to load and “Mouse0” is the label that identifies the device externally.

After saving and closing the file, the server is ready to run. To do so, simply execute the file “vrpn_server.exe”. Some devices print some information that is

⁸ Updated versions of VRPN are nightly built and are available at <http://public.vrac.iastate.edu/~rpavlik/downloads/vrpn-visual-studio-snapshots/>

obtained during their connection or calibration (if needed), the mouse will not, so once the server is opened and no error is shown, everything is working fine. Also, each device may require a series of libraries to be accessed, the mouse for instance requires the XINPUT library that is provided by the DirectX, if no DirectX is installed, an error message will be shown alerting about the lack of the XINPUT dll.

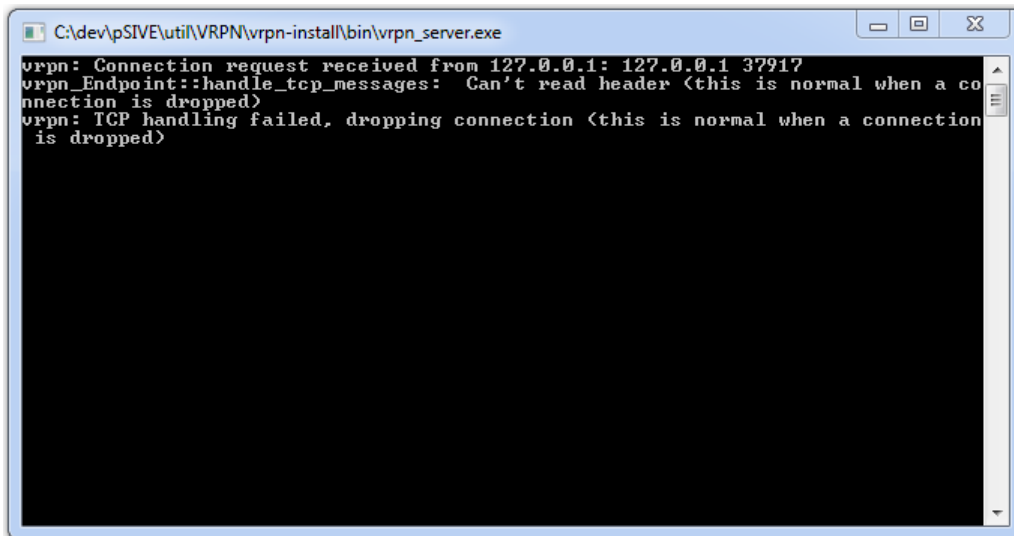
To perform a quick check, it is possible to use the “vrpn_print_devices” application. It runs by receiving the name of the device and the ip address of the server to connect, for this example, run “vrpn_print_devices Mouse0@localhost”. The application will show the data obtained from the mouse as on Image 52:



```
Administrator: C:\Windows\system32\cmd.exe
C:\dev\pSIVUE\util\VRPN\vrpn-install\bin>vrpn_print_devices.exe Mouse0@localhost
Opened Mouse0@localhost as: Tracker Button Analog Dial Text.
Press ^C to exit.
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Button Mouse0@localhost has 3 buttons with states: 0 0 0
Analog Mouse0@localhost:
  0.28, 0.66 (2 chans)
Analog Mouse0@localhost:
  0.28, 0.66 (2 chans)
Analog Mouse0@localhost:
  0.28, 0.66 (2 chans)
Analog Mouse0@localhost:
  0.28, 0.66 (2 chans)
Analog Mouse0@localhost:
  0.28, 0.66 (2 chans)
#####
Button Mouse0@localhost, number 0 was just pressed
#####
Button Mouse0@localhost, number 0 was just released
#####
```

Image 52 VRPN showing mouse status

At the same time, information on the connections active are displayed on the server window (Image 53).



```
C:\dev\pSIVE\util\VRPN\vrpn-install\bin\vrpn_server.exe
vrpn: Connection request received from 127.0.0.1: 127.0.0.1 37917
vrpn_Endpoint::handle_tcp_messages: Can't read header (this is normal when a connection is dropped)
vrpn: TCP handling failed, dropping connection (this is normal when a connection is dropped)
```

Image 53 VRPN Server messages

A VRPN client performs the connection to the server and receive the information from it as the device change its state (button press/release, analog or tracker values change). A simple client in C++ can be built as the code snippet bellow:

```
#include "vrpn_Analog.h" //VRPN Analog device Header

#include <iostream>
#define DEVICE "Mouse0@localhost" //Device to connect
//Callback to recieve the device updates
void VRPN_CALLBACK analog_callback( void* userData, const vrpn_ANALOGCB
analogDevice ){
    int nbChannels = analogDevice.num_channel; //Get the number of analog
channels available
    std::cout << "Data Received: ";

    for( int i=0; i < analogDevice.num_channel; i++ ) {
        std::cout << analogDevice.channel[i] << " ";
    }

    std::cout << std::endl;
}

int main(int argc, char* argv[]){
    vrpn_Analog_Remote* vrpnAnalog = new vrpn_Analog_Remote(DEVICE); //Connect
to the device's analog component

    vrpnAnalog->register_change_handler( 0, analog_callback ); //Set the
callback for analog input

    while(1) { //Update
        vrpnAnalog->mainloop();
    }

    return 0;
}
```

To build the code, it is required to add the folders include and lib to the compiler and linker respectively, also link with vrpn.lib. In this guide, the code was compiled using Visual Studio 2010. The result should look like Image 54.

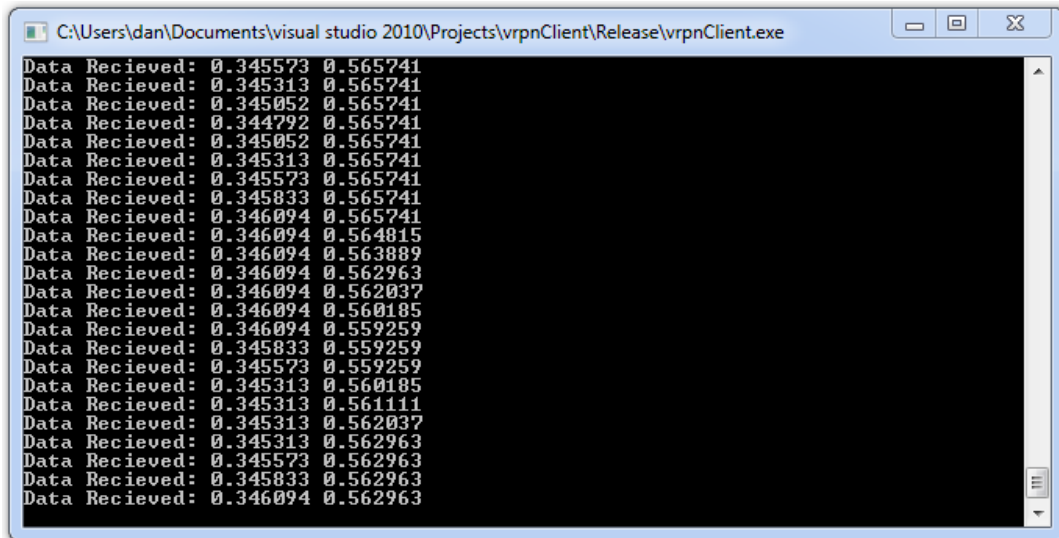


Image 54 VRPN demo client

Buttons and Trackers are read in the same way, just requiring a new callback for each one and a new connection. The following code adds the mouse buttons to the client:

```
#include "vrpn_Analog.h" //VRPN Analog Header
#include "vrpn_Button.h" //VRPN Button Header
#include <iostream>
#define DEVICE "Mouse0@localhost" //Device to connect

//Callback to recieve the device updates
void VRPN_CALLBACK analog_callback( void* userData, const vrpn_ANALOGCB
analogDevice ){
    int nbChannels = analogDevice.num_channel; //Get the number of analog
channels available
    std::cout << "Data Recieved: ";

    for( int i=0; i < analogDevice.num_channel; i++ ) {
        std::cout << analogDevice.channel[i] << " ";
    }

    std::cout << std::endl;
}

/*The buttons will all be handled the same way,
They will evoke this callback.
Each one has the identifier and its current state
*/
```

```

void VRPN_CALLBACK button_callback( void* userData, const vrpn_BUTTONCB
button ){
    std::cout << "Button '" << button.button << "': " << button.state <<
std::endl;
}

int main(int argc, char* argv[]){
    vrpn_Analog_Remote* vrpnAnalog = new vrpn_Analog_Remote(DEVICE); //Connect
to the device's analogs
    vrpn_Button_Remote* vrpnButton = new vrpn_Button_Remote(DEVICE); //Connect
to the device's buttons
    vrpnAnalog->register_change_handler( 0, analog_callback ); //Set the
callback for analog input
    vrpnButton->register_change_handler( 0, button_callback ); //Set the
callback for button input

    while(1) { //Update
        vrpnAnalog->mainloop();
        vrpnButton->mainloop();
    }

    return 0;
}

```

The result should be something like Image 55

```

Administrator: C:\Windows\system32\cmd.exe
Data Recieved: 0.138542 0.297222
Data Recieved: 0.138281 0.29537
Data Recieved: 0.138021 0.294444
Data Recieved: 0.13776 0.291667
Data Recieved: 0.1375 0.290741
Data Recieved: 0.1375 0.289815
Data Recieved: 0.13724 0.287963
Data Recieved: 0.13724 0.287037
Data Recieved: 0.136979 0.286111
Data Recieved: 0.136979 0.283333
Data Recieved: 0.136719 0.283333
Data Recieved: 0.136719 0.282407
Data Recieved: 0.136719 0.281481
Data Recieved: 0.136719 0.280556
Button '1': 1
Button '1': 0
Button '0': 1
Button '0': 0
Button '2': 1
Button '2': 0
Button '0': 1
Button '0': 0
Button '1': 1
Button '1': 0
Button '2': 1

```

Image 55 VRPN demo client improved

For implementations of the client using another language than C++, it is possible to build the wrappers from the source code for Java and Python, as well as a port to Android.

Annex 03 – Devices Configuration on VR Juggler

For a full guide refer to

http://vrjuggler.org/docs/vrjuggler/3.0/configuration.guide/configuring_vr_juggler.html

This guide will briefly explain the structure of VR Juggler configuration files for input devices by showing a step-by-step configuration of the Razer Hydra through the VRPN.

VR Juggler, just like VRPN, will abstract the physical devices and classify them into a set of categories so for the application any device that fits its requirement, can be integrated without the need of altering the code.

The VRPN server used on Annex 02 is already capable of provide access to the Hydra as is. The only required step is to adding “vrpn_Tracker_RazerHydra Hydra0” to the VRPN.cfg file or by uncommenting the line that contains “vrpn_Tracker_RazerHydra” the important point is to remember the label given to it (in this case is Hydra0).

After the configuration is done, opening the VRPN server will output a message warning that the calibration is been executed. Leave the controllers on their base, and in the correct side of the magnetic emitter until the message that everything is working is displayed shows up.

For the VR Juggler part, there are two ways of creating a new configuration file: first is to manually edit a text file, and the other is to use the configuration interface. For this guide, we will stick to the manual text file editing since the configuration interface is not trivial and will be more trouble than help.

A configuration file must have:

- The driver mapping telling which driver to load from VR Juggler’s core
- The device itself (Physical or Simulated)
- Device Proxies to provide the abstraction between device and application
- Proxy Alises to add an extra layer of abstraction, for instance to refer to a button on a gamepad as “Up Button” and so the application will just need to refer to it as its label.
- Position Filters to correct placements and coordinate systems.

The starting point to any input device configuration is the driver mapping. On this case, the driver VR Juggler needs is “VRPN_drv”. It is added on the configuration as:

```
<input_manager name="VRPN Input" version="2">
<driver>VRPN_drv</driver>
</input_manager>
```

Since it is coming through VRPN, the Gadgeteer (module responsible for I/O in VR Juggler) will act as a VRPN client. That way any device supported by it can be integrated into VR Juggler. However, the devices from VRPN are available “disassembled”, so it is necessary to combine the different elements (buttons, analogs and trackers) to have the full device available on the Juggler side. It is also possible to use only part of the elements available.

The Hydra consists of two units with where each one has 1 tracker, 8 buttons and 3 analog axis. To correct the coordinate system, a pre-rotation is required and to add both units the settings must be as follow:

```
<vrpn name="HydraDevice" version="2">
<tracker_server>Hydra0@localhost</tracker_server>
<button_server>Hydra0@localhost</button_server>
<analog_server>Hydra0@localhost</analog_server>
<tracker_count>2</tracker_count>
<button_count>16</button_count>
<analog_count>6</analog_count>
<min>-1.0</min>
<max>1.0</max>
<position_filters>
<position_transform_filter name="Position Transform 0" version="1">
<pre_translate>0.0</pre_translate>
<pre_translate>0.0</pre_translate>
<pre_translate>0.0</pre_translate>
<!--X-->
<pre_rotation>0.0</pre_rotation>
<!--Y-->
<pre_rotation>180.0</pre_rotation>
<!--Z-->
<pre_rotation>0.0</pre_rotation>
<custom_scale>1.0</custom_scale>
<device_units>1.0</device_units><!--Meters -->
<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<!--X-->
<post_rotation>0.0</post_rotation>
<!--Y-->
<post_rotation>0.0</post_rotation>
<!--Z-->
<post_rotation>0.0</post_rotation>
</position_transform_filter>
</position_filters>
</vrpn>
```

Device proxies are required to add an abstraction layer, but in this case they also split the device into two independent units. On this configuration, the right unit will become a wand, to control navigation and input of buttons data, while the left will be a head tracking device. However, it is just a scenario. Any other setup is possible of accomplishing as long as the application can handle.

Again another position filter is applied so the right unit can be placed on the hand like a remote control, horizontally. The left unit is adjusted to stay vertical, to be placed on the back of the head.

```

<position_proxy name="Wand Proxy" version="1">
<device>HydraDevice</device>
<unit>1</unit>
<!--Right controller-->
<position_filters>
<position_transform_filter name="Position Filters" version="1">
<pre_translate>0.0</pre_translate>
<pre_translate>0.0</pre_translate>
<!--X-->
<pre_rotation>0.0</pre_rotation>
<!--Y-->
<pre_rotation>0.0</pre_rotation>
<!--Z-->
<pre_rotation>0.0</pre_rotation>
<custom_scale>1.0</custom_scale>
<device_units>1.0</device_units>
<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<!--X-->
<post_rotation>90.0</post_rotation>
<!--Y-->
<post_rotation>0.0</post_rotation>
<!--Z-->
<post_rotation>0.0</post_rotation>
</position_transform_filter>
</position_filters>
</position_proxy>
<position_proxy name="Head Proxy" version="1">
<device>HydraDevice</device>
<unit>0</unit>
<!--Left controller-->
<position_filters>
<position_transform_filter name="Position Filters" version="1">
<pre_translate>0.0</pre_translate>
<pre_translate>0.0</pre_translate>
<!--X-->
<pre_rotation>0.0</pre_rotation>
<!--Y-->
<pre_rotation>0.0</pre_rotation>
<!--Z-->
<pre_rotation>0.0</pre_rotation>
<custom_scale>1.0</custom_scale>
<device_units>1.0</device_units>

```



```

<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<post_translate>0.0</post_translate>
<!--X-->
<post_rotation>0.0</post_rotation>
<!--Y-->
<post_rotation>0.0</post_rotation>
<!--Z-->
<post_rotation>0.0</post_rotation>
</position_transform_filter>
</position_filters>
</position_proxy>

```

Finally, by adding the alias, a label is associated to the proxies so the system can refer to.

```

<alias name="VJWand" version="1">
<proxy>Wand Proxy</proxy>
</alias>

<alias name="VJHead" version="1">
<proxy>Head Proxy</proxy>
</alias>

```

As for the buttons and analogs, the process is basically the same without the need of position filters. To define the axis of the joystick, two analog proxies are created along with the respective alias:

```

<analog_proxy name="rAnalogXAxis" version="1">
<device>HydraDevice</device>
<unit>3</unit>
</analog_proxy>
<analog_proxy name="rAnalogYAxis" version="1">
<device>HydraDevice</device>
<unit>4</unit>
</analog_proxy>

<alias name="rUpDown" version="1">
<proxy>rAnalogYAxis</proxy>
</alias>
<alias name="rLeftRight" version="1">
<proxy>rAnalogXAxis</proxy>
</alias>

```

The full configuration file is available along with pSIVE. Also, for a deeper understanding the official guides are highly recommended.

Annex 04 – pSIVE Installation Guide

As pSIVE is based on VR Juggler and OpenSceneGraph, they must have been set up prior to the installation. For VR Juggler the basic steps are listed on Annex 01, however the configuration files created for the equipment available on IEETA must be within VR Juggler's folder.

The files are along with pSIVE and they must be placed inside the "VJ_CFG_PATH" folder, so the VR Juggler can find them. The files are inside the "VRJ Config" folder.

OpenSceneGraph can be downloaded ready to use, but it will not have the plugin required by pSIVE to load video files. The framework has to be built from scratch with the ffmpeg plugin as well. Instructions for building and installing the framework are available on its website (www.openscenegraph.com).

Once OpenSceneGraph is built and installed, some environment variables have to be created:

- OSGHOME – OpenSceneGraph's root folder.
 - Ie.: C:\dev\OpenSceneGraph\OpenSceneGraph-3.0.1
- OSGPATH – The path to all binaries
 - Ie.: %OSGHOME%\bin;%OSGHOME%\data;%OSGHOME%\lib
- Add %OSGPATH% to the Path

Another pSIVE requirement is JAVA for the configuration tool, it is recommended to have JAVA SE 7u45.

Currently pSIVE is available as a Visual Studio 2010 Solution (image xxx), however it is a future goal to adapt it to cmake and allow it to be prepared for any compiler. With the environment set, the pSIVE just have to be built.

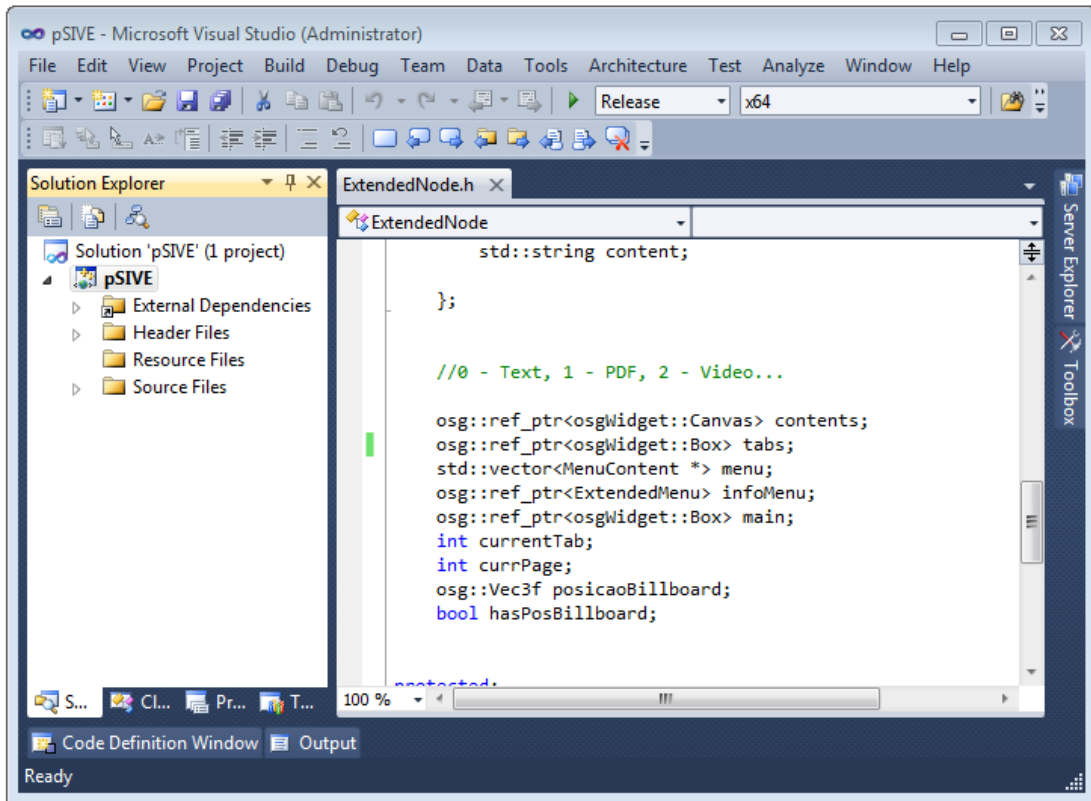


Image 56 pSIVE project on Visual Studio

Once pSIVE is built, it have to be placed within the folder bin (image) of the platform to replace the previous built and to be started from the script – that is controlled by the configuration tool.

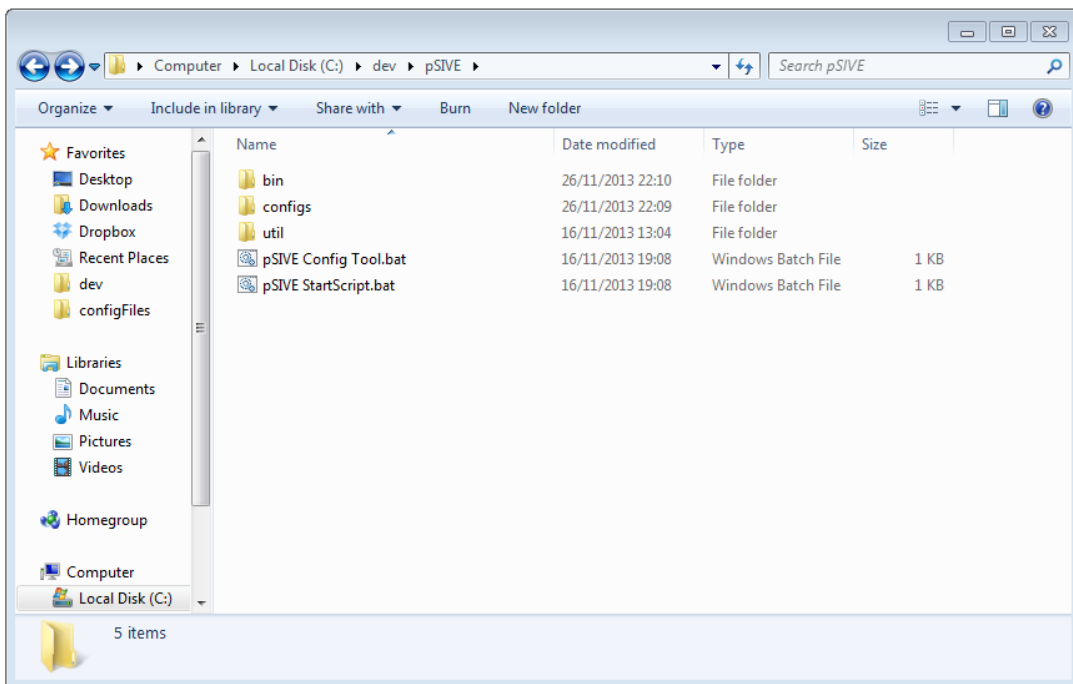


Image 57 pSIVE folder