Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Visualization and Interaction in a Simulation System for Flood Emergencies

Rui Pedro da Silva Nóbrega
Nº 26373

Orientador
Prof. Doutor Nuno Manuel Robalo Correia

Dissertação apresentada na
Faculdade de Ciências e Tecnologia da
Universidade Nova de Lisboa para obtenção do
grau de Mestre em Engenharia Informática

Lisboa

Julho 2008

Acknowledgements

I would like to thank to everyone who helped and supported me in the last year during the dissertation. Special thanks to Nuno Correia for the advices and directions, to my project leader Armanda Rodrigues and to my friend and colleague André Sabino. I also thank to all my colleagues and friends for the help, the fun and the companionship especially to Duarte Gonçalves, Diogo Cabral, Rui Jesus, Filipe Grangeiro, Rossana Santos, Carmen Morgado and Rute Frias. Finally, I thank Guida for being so sweet, to my parents for the great support and to my younger sister Claudia.

Keywords

Palavras-Chave:

- Interacção Pessoa-Máquina
- Gestão de Emergências e Disastres
- Sistemas de Informação Geográfica
- Multimédia
- Computação Gráfica

Keywords:

- Human Computer Interaction
- Disaster Emergency Management
- Geographic Information Systems
- Multimedia
- Computer Graphics

Resumo

Esta tese apresenta um sistema de visualização e interacção para simulação de emergências em caso de cheias fluviais. Apresenta-se ainda um estudo detalhado sobre formas de apresentação gráfica de elementos críticos de emergências. Todos estes elementos são conjugados numa aplicação baseada em informação geográfica e simulação de agentes. Uma grande maioria dos objectivos desta tese estão interligados com o projecto Life-Saver. Este projecto tem por objectivo o desenvolvimento de um simulador de resposta a emergências, para o qual é necessario criar um sistema de visualização e interacção.

A tese explora as áreas da visualização em situações de emergência, criação de interfaces multimédia e interacção entre o utilizador e o sistema.

A componente aplicacional desta tese representa o cenário da simulação com os seus múltiplos agentes e respectivas acções. Foram efectuados vários estudos para criar uma interface intuitiva. São estudadas e usadas novas formas de interacção multimédia como, por exemplo, quadros interactivos sensíveis ao toque ou painéis *multi-touch*. É possível o carregamento e consulta de informação geográfica no cenário. A arquitectura resultante é usada para visualizar uma simulação de uma situação de emergência e inundação no caso de rebentamento da barragem do Alqueva na bacia do rio Guadiana.

Abstract

This thesis presents an interaction and visualization system for a river flood emergency simulation. It will also present a detailed study about forms of visual representation of critical elements in emergencies. All these elements are currently assembled in an application based on geographic information systems and agent simulation. Many of the goals in this thesis are interconnected with project Life-Saver. This project has the goal to develop an emergency response simulator, which needs a visualization and interaction system.

The main goals of this thesis are, to create a visualization system for an emergency, to design an intuitive multimedia interface and to implement new forms of human-computer interaction.

At the application level there is a representation of the simulation scenario with the multiple agent and their actions. Several studies were made to create an intuitive interface. New forms of multimedia interaction are studied and used such as interactive touch sensible boards and multi-touch panels. It is possible to load and retrieve geographic information on the scenario. The resulting architecture is used to visualize a simulation of an emergency flooding situation in a scenario where the Alqueva dam in Guadiana river fails.

Contents

List of Figures

Acronyms

AR	Augmented Reality
ANPC	Autoridade Nacional de Protecção Civil
CEGUI	Crazy Eddie's Graphic User Interface
DBEM	Dam Break Emergency Management
EDIA	Empresa de Desenvolvimento e Infra-estruturas do Alqueva
FCT	Fundação para a Ciência e a Tecnologia
GIS	Geographic Information System
IWB	Interactive White Board
LNEC	Laboratório Nacional de Engenharia Civil
OGRE3D	Object-Oriented Graphics Rendering Engine 3D
PLSM	Paging LandScape Manager
RTS	Real-Time Strategy
TUI	Tangible User Interfaces

1.Introduction

In a world full of disasters and unexpected situations, disaster management has become one of the key topics of today. Visualization and interaction are main aspects in Emergency Management. Timely decisions depend on having the right tools, available in a way that is easily perceived by the users. Instead of waiting for the disaster, it is of most importance to be prepared for it. What to do? Where to move? Who should we call? These are questions that need to be answered in advance. Decision makers must be aware of the dangers and plan ahead.

The main motivation for this work was to build and study a multimedia system that could accurately and interactively represent an emergency scenario. For that we use several techniques that were developed in areas like games, digital media or in the entertainment industry and apply them in real world hazard scenarios.

Emergencies can have natural causes such as earthquakes, fires, hurricanes or floodings. They can also be caused by wars or terrorist attacks. The emergency plans must handle large amounts of data such as maps, locations, response personnel available or building configurations. Lines of action have to be drawn, all the agents involved must be aware of their roles and everything should be tested to its full extent with drills and role playing.

Unfortunately some emergency situations are impossible to simulate in real life. It is impossible to break a river dam and see the flood resulting from it or set fire to a building and

watch the reaction of the people inside. Sometimes the simulation is too costly to be done in the real world. This is where computer simulation comes in. Given an emergency scenario, a computer can simulate the disaster and the actions taken by the agents involved and several models can be tested. There are even real-time simulations were the users take the role of real agents.

Modeling the emergency scenario [32] is essential for the process of making a simulation tool but if the users cannot easily interact with the system they usually tend to ignore it. In order to make a successful system to validate emergency plans it is necessary to have a visual comprehensive interface for the users to interact.

1.1 Context

The Visualization module that is being proposed is supported by a model for a disaster scenario that is being simulated. This thesis is part of a larger project called Life-Saver, that has the goal to build a flood emergency simulator with a visual multimedia interface.

Life-Saver was created in the Portuguese context of Dam Break Emergency Management (DBEM). The management of flooding emergencies resulting from problems related with dams is planned through the guidance of existing special emergency plans. The purpose of the simulator is to test the efficacy of these plans.

According to a LNEC (Laboratório Nacional de Engenharia Civil) study [9] there were 200 serious dam failures in the world in the 20th century causing the loss of 8 000 lives. Important key failures were the Malpasset dam (421 deaths, France, fig.1.1) in 1959, the accident at Vajont dam (2 600 deaths, Italy) in 1963 or the failure of the Teton dam in 1976 (U.S.A.).

Fig. 1.1– Malpasset dam, failed in 1959 in France

In Portugal, there were in 2003 almost 150 large dams and 35% of them were built between 70 and 35 years ago. According to the Portuguese dam safety legislation, all major dams must have a dam break analysis and technical procedures aimed to increase public safety must be implemented. All dams must have inundation maps, risk-zoning and emergency plans. Taking this into account the project Life-Saver was proposed and funded by FCT (Portuguese Foundation for Science and Technology). To develop the project there is a partnership between LNEC (Laboratório Nacional de Engenharia Civil) and FCT-UNL (Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa).

The objective of this project is to develop a system that can effectively validate existing emergency plans, through simulation of the flooding natural phenomena and of all the actors intervening in the situation. To achieve this, the project is separated in three main components: Simulation, Visualization and Controller. In the work that is being presented here the main focus will be the Visualization module although a brief description of the other components will be presented.

The simulation module is supported by the available spatial data from the downstream valley of the dam. This information is handled by an agent-based simulation engine, which automatically feeds relevant spatial information to the visual interface component as needed. The simulator will be able to define emergency scenarios which will include available DBEM (Dam Break Emergency Management) resources, actors and roles. The system dynamics are visualized and manipulated with a graphical interface representing the emergency scenario, while parameters characterizing this dynamics are registered during the simulation period, for later analysis. All the simulation is configured and initiated by the controller component. Figure 1.2 illustrates the relation between the three modules. This system design allows a complete independence between each module and allows the existence of several visualization modules, each with its special features.

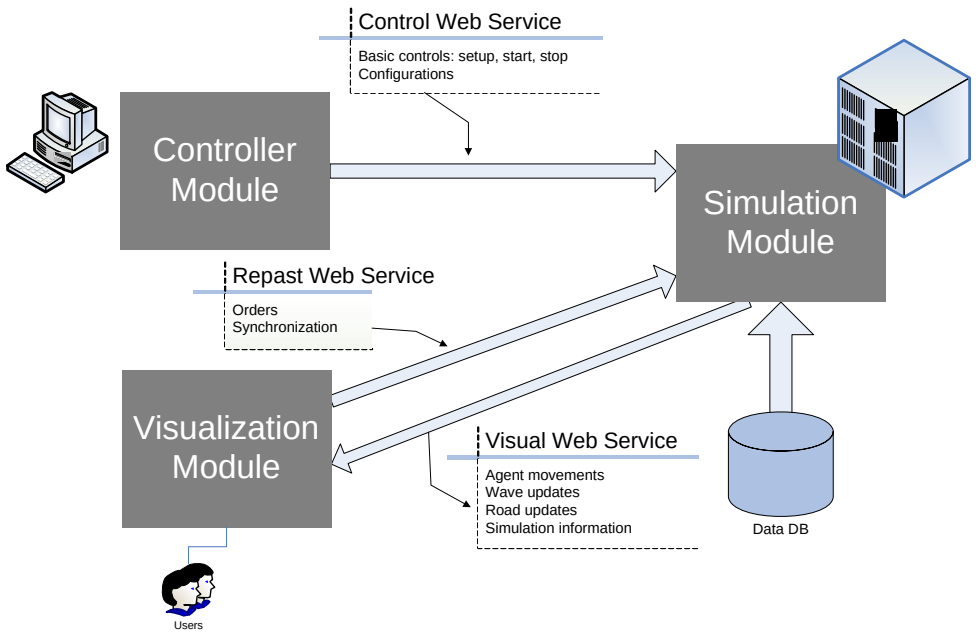

Fig. 1.2– Life-Saver component model

1.1.1 Case study

A case study scenario was created in order to test the interaction and visualization system. The simulator would run an evacuation plan from the Portuguese authorities where the Alqueva dam collapsed and created a flood. The Visualization system should show the disaster, all the operations involved and collect data for future simulations.

The Alqueva dam is placed in the Guadiana River, about 150 km north of its base level, at Vila Real de Santo António. It is the largest dam in Europe, holding the Europe's largest artificial lake.

In a catastrophe scenario, the collapse of the dam's wall, there is a set of events that can be predicted. First of all, there's a formation of a wave 100 meters high, starting from the dam's position. The wave is expected to reduce in strength, while it progresses through the valley, eventually becoming a flood hazard. According to studies on the wave, the civil protection infrastructure can only initiate evacuation operations after the first 30 minutes of flooding. This leaves every person placed in the territory, which can be reached by the wave before that time, the initiative to escape to pre-determined safe spots. Sirens are placed in the valley to alert the populations in the case of emergency.

The agent-based simulation [3] system (fig.1.3) integrates GIS data with an Object Oriented, Individual-Based Modeling approach providing a successful simulation platform for the emergency plan validation process. The simulation model is mainly concerned with the location-based and location-motivated actions of the involved agents, providing data describing the likely effects of a specific emergency situation response, upon which the acceptance of the plan can be supported or changes suggested.

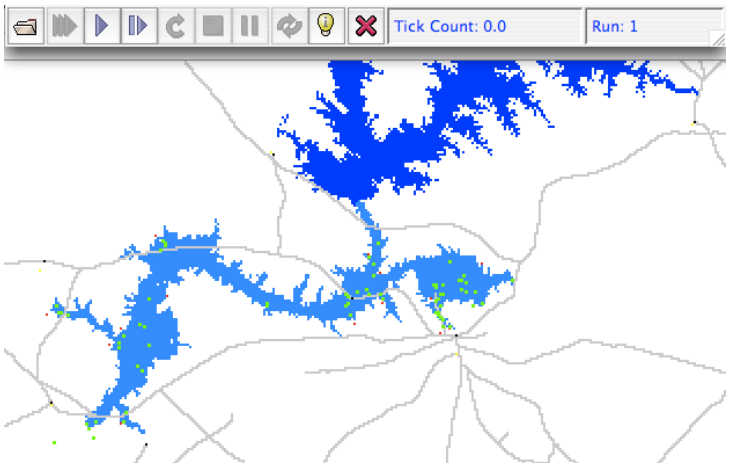

Fig. 1.3– Repast Simulation Component

The simulation receives orders from the Controller module and feeds the Visualization modules with information. Additionally, it can receive feedback from the visualization when the user gives specific instructions through the interface.

1.2 Objectives

Considering the scenario and background above, the main goals of this thesis are:

(i) Create a visualization system for an emergency.

(ii) Design an intuitive user interface.

(iii) Study and implement new forms of human-computer interaction.

The theme of the thesis involved a research in a wide range of areas such has multimedia [20], geographic information systems (GIS), augmented reality [41], game design [25], interface design [7] and interaction [2].

To create the emergency system it was necessary to develop the Visualization application for the Life-Saver project, using a 3D game engine. This Visualization component is responsible for displaying the information that comes out of the simulation with an intuitive interface. One of the goals of the thesis was to make experiments with new forms of interaction such as interactive white-boards or multi-touch devices. New paradigms like

multipoint interaction, multi-touch interaction, augmented reality or tangible interfaces were tested.

Finally, there was the intention to submit the work to conferences in the area and demonstrate it to the partners LNEC, EDIA and ANPC (Protecção Civil).

1.2.1 Contributions

The main contributions and results of this thesis are:

- An application that can connect to the simulation module to retrieve data and present it.
- A multi-touch device supported with computer vision techniques and was connected to the application.
- Two C++ generic libraries with graphic primitives created to support the application.
- An interaction system application using an interactive whiteboard.
- A large amount of GIS information processed to fit the project needs.
- Background research in many areas including game architecture design and multi-touch technologies.

1.2.2 Publications

The following publications about the system have been submitted and accepted:

Nóbrega, R., Sabino, A., Rodrigues, A., Correia, N.:
Flood Emergency Interaction and Visualization System
Proceedings of the 10th International Conference on Visual Information Systems Visual 2008,
Salerno, Italy, 11-12 September 2008

Sabino, A., Nóbrega, R., Rodrigues, A., Correia, N.:
Life-Saver: Flood Emergency Simulator
Proceedings of the Information Systems for Crisis Response and Management Conference ISCRAM 2008,
Washington D.C., USA, May 4-7, 2008

1.3 Dissertation structure

This document presents the work done during the master thesis elaboration. Chapter two presents a survey of the related work showing other perspectives and approaches. Chapter three describes all the modeling and implementation of the main application. Chapter four explains the decisions that were made at an interface level, and the fifth chapter details the interaction systems that are used. The conclusion summarizes the work that resulted from this thesis and the main directions for future work.

2. Related Work

This chapter is a survey of other research work that is related or was used as source of inspiration for this thesis. The main sources of information are papers that were published in conferences, books, articles and internet web sites. The topics will be approached in two main areas:

- Multimedia emergency plan simulation and visualization: here are explored other existing simulation systems and are showed visualizations of emergency situations using game engines and GIS features
- Interaction with the system: new forms of interaction including tangible interfaces, multi-touch systems and similar technologies available in this area.

In the next pages a wide range of topics related with this work will be presented including several case studies developed by different teams in different countries, working projects in other universities and research centers and related commercial applications and interaction paradigms.

2.1 Dam risk studies in Portugal

Previous hydrological studies were carried out by LNEC in the Guadiana river valley to determine the flooding zones in several scenarios. Various degrees of threat were considered. The scenarios included the total failure of the Alqueva dam and/or the failure of the next dam,

the Pedrogão dam. The resulting wave from the rupture of the dams was computer simulated using hydraulic models that took into account the height of the dam and the width of the river in each particular spot. With this simulation important information was obtained:

- The maximum flooding area in each scenario.
- The time in seconds, with a precision of 500 meters, that took the wave to reach a particular spot since the failure of the dam.
- The height of the wave, with a precision of 500 meters, at each spot.

This data was then introduced in the simulation model so that agents in the terrain take into account the disaster wave.

Between 1994 and 2000 LNEC conducted a research in the context of a NATO funded project intended to develop Dam Break Flood Risk Management in Portugal [9]. In this project a large international team was assembled with the objective to create the technical and policy guidelines necessary for the safety and risk management in river valleys. Several problems related with potential abnormal floods induced or amplified by man-made structures along river valleys were studied.

In the book describing the project, a case study was used to illustrate all the techniques and methodologies involved. For the case study a valley in the South-Western region of Portugal was chosen with two dams upstream two urban areas, the Arade river valley. The river is 45 km long and has a surface of 800 km^2. It crosses six districts Silves, Portimão, Loulé, Monchique, Lagoa and Almodôvar. The nearby population is about 100 000 inhabitants.

The objective of the case study was to make a dam break risk assessment along the Arade valley, in an extension of 29.3km between the Funcho dam, passing the Arade dam and into the sea. The Funcho dam was built essentially to provide water to the populations. It is a concrete arch dam 36m high with a 210m long crest and a reservoir capacity of 43.4hm^3. The

Arade is an earth fill dam, 50m high with a 246m long crest and 28hm^3 of capacity. Several studies were conducted in this area taking into account:

- Dam and valley safety
- Hydraulic analysis and computational simulations
- Emergency planning
- Computer-aided decision support systems
- Risk management from a social point of view

Arguably, there was a careful procedure followed regarding public information. In order to not induce fear or anxiety among the population, the level of knowledge released to the public was kept very low. This project was essential to create the knowledge base that was necessary later on when the project Life-Saver was proposed.

2.2 Multimedia emergency plan simulation and visualization

There are many emergency related projects and here special focus was placed on projects that more directly provide a good background to this thesis and have a multimedia interface with interesting features.

2.2.1 Frameworks for simulation

There are many simulation tools available for emergency response applications. Jain and Mclean [32] propose a framework to integrate several different simulation tools to provide a coherent whole picture to decision makers.

By providing a framework for modeling and simulation for emergency response they are not focused on the simulation itself but on the means to create a generic interface by which several independent applications can cooperate in order to achieve a goal.

The context of this work is related with the events of September 11, 2001 in the United States and focuses on homeland security emergency response. It states the increasing importance of preparedness in order to avoid future life and economic hazardous.

According to the authors the main solution to this problem is to win the “interoperability challenge”. Building simulation models from scratch is a “high expertise requirement and time-consuming process”. To build an interoperable framework the paper identifies several common areas and subjects present in almost all emergencies such as Disaster Events, Entities of Interest, Applications, Planning, Vulnerabilities Analysis, Identification and Detection, Training and Real-Time Response Support.

One of the proposed measures is a visualization system that is independent from the simulation itself. This recommendation is used in the architecture of the Visualization application that was implemented.

2.2.2 Disaster simulation systems

The Tangible Media Group [35] from MIT has made some interesting proposals in the disaster simulation area. The Tangible DSS (Disaster Simulation System) seen in figure 2.1 is

Fig. 2.1– Tangible Disaster Simulation System.

a tool for planning disaster measures, based on disaster and evacuation simulation using Geographic Information Systems. They use a tool called Sensetable [18] which is a table where an image is projected enabling interaction with viewed objects. This system is specially designed to support collaborative work and discussion since the physical input can track multiple users on the table. The input system and the Sensetable will be detailed in the interaction section.

Another interesting project is the HazMat: Hotzone [16] from Carnegie Mellon University. This is a simulation that uses computer game technology to train fire fighters against chemical and hazardous emergencies.

The application looks and feels just like a 3D multiplayer first-person shooter game as seen in figure 2.2. The only difference is that it has an instructor that has complete control over the scene and the objective is to train communication, observation and critical decision making. The instructor has the ability to pause the game or trigger unexpected actions and secondary events.

In order to create people awareness regarding the issues of disaster management the United Nations created an online game [34] where users can administer and control zones that

Fig. 2.2– Snapshot of the HazMat: Hotzone simulator

have been devastated by natural disasters. This is a grid-like simulation game that intends to train people that live in risk-zones to be aware of what to do in a disaster situation.

2.2.3 Multimedia approach in Valencia's metro

Paper [22] explains thoroughly the multimedia system implemented to manage the Valencia's metro emergency plans. Just like the river dams, the metro needs to have an emergency plan. This must be submitted to and approved by the corresponding authorities.

Emergency plans usually are large complex documents that integrate all the information necessary to save lives. In large buildings, the plan contains maps with highlights at the emergency exits and at all safety equipment such as fire extinguishers. It explains how to route people through the corridors or through the alternative routes when the others are blocked. In subway systems emergency plans are far more complex.

Responses to emergencies are normally coordinated from a centralized location where safety managers use the emergency plan as a decision-making guide. "Regardless of the complex procedures it describes (…), the plan's success is always measured by how effective the evacuation is" [22].

In the Valencia subway context, the problem was how to integrate a large amount of emergency information on a single spot and make it easily available for the decision makers. In a critical situation there is no time to consult random paper documentation. Their solution was turning the emergency plan into a multimedia software system (fig.2.3) that would integrate text, audio, video, 3D models, and animation for handling emergencies in underground metropolitan transportation.

Fig. 2.3– Interface of the emergency management system of the Valencia metro

The multimedia system was initially composed by a set of hypermedia pages that were linked together. Later on, several ubiquitous sensors were installed in the underground tunnels. These sensors and cameras were used to enhance the hypermedia pages. If something happens in the metropolitan, the controller can immediately evacuate the trains and the stations, consult the procedures that should be taken, analyze the escape routes and safety equipment and monitor real life events through the cameras.

2.2.4 Using computer game engines

Modern computer games make use of technologies from almost all areas of computer science: graphics, artificial intelligence, network programming, operating systems, languages and algorithms. These applications used for enjoyment and entertainment are usually very powerful regarding visualization and interaction techniques.

In [8] it is discussed the utility of using 3D computer game engines for information visualization. In the initial survey, the paper describes a large number of applications that take

advantage of game engines; these include architectural design, military simulations management, landscape planning and even an interface for UNIX process management.

The authors final objective was to create a source code comprehension tool that used a 3D environment metaphor so that users can organize and explore the code using spatial memory. Perhaps what is more important in this work is that they considered that commercial or open-source game engines can have a place in a scientific environment as a mean of showing rich content and context to users. Additionally, the paper makes a summary of the main game genres and suggests a large number of available game engines that are worth to try.

One of the game genres described is Real Time Strategy (RTS). In a RTS game the world is seen from above, the units work as agents with limited artificial intelligence and information about each unit or building can be retrieved by clicking on it. Many RTS games have sound alerts and provide several shortcuts that enable a quick response by the user to any emergency. In the context of emergency management the idea is to make the emergency visualization interface to look as close as possible to a RTS game (Real-Time Strategy game). Good examples of RTS games are the series Age of Empires, Warcraft or Command and Conquer.

In the work described in this dissertation a computer game engine library called Ogre3D [28] was used. This is a C++ open source scene-oriented library that is widely used to create 3D games and it will be detailed later in section 3.2.

Before the Ogre3D was chosen, other alternatives were considered. The other main candidate was Virtools [40] a product from Dassault Systèmes'3DVIA. Virtools is a framework solution to create 3D applications. The main difference between Ogre and Virtools is that the later is not a library meant to be used in a language such as C++. Virtools is a graphical program which allows a visual integration of all the elements in a 3D scene.

Additionally, it has a scripting language for extra customization of each object on the scene. Although it is a very powerful tool, it is paid software, with a small community using it and it is more difficult to integrate with external solutions.

Beyond these two solutions several other game engines exist. There are many open-source projects such as Crystal Space, Irrlicht or The Nebula Device 2. There are also the Doom, Doom2, Quake and Quake2 engines which have been open sourced by Id Software. The best [10] game engines available are commercial closed-source libraries used especially in the FPS game genre. Among these are Doom3 and Quake 3 engines from Id Software, Half-Life 2 from Valve Software and Unreal Tournament and Unreal Tournament 2004 from Epic Games. All of these closed-source engines are fully-featured and for each exists more than one complete game based on it.

2.2.5 Merging computer games and simulation

One of the first persons to blend in the concept of computer simulations and games was Will Wright [7]. He is the founder of the Maxis game company and the creator of well known titles like SimCity. In 1983 he released his first game called Raid on Bungling Bug. This was a simple shut-m-up where the player flew around some islands and dropped bombs. The more relevant aspect about this game was that it required the creation of a landscape editor capable of drawing terrain, roads and buildings. Will decided to improve the editor, studied theories about urban dynamics and traffic models and in the end he made a little toy city. In 1985 it was launched as SimCity. It was a huge success and was followed by several titles like SimAnt and SimEarth. Perhaps the more famous of all the initial Sim's series was SimCity2000 (fig.2.4).

Fig. 2.4– Disaster in Sim City 2000

From a visualization point of view SimCity2000 introduced new and refreshing ideas. It represented the terrain in a grid form just like its predecessors but this time it was in an isometric view. This gave the game a whole new 3D sensation. Additionally, the terrain had relief, with the different heights affecting the behavior of the world. There were different levels of infrastructure with subways, pipes and streets. The city had context and interacted with the nearby cities. There were natural disasters that occasionally would occur and the user would have to do the emergency management.

In terms of emergency simulation many agent-based frameworks for simulation still use today a grid form to represent the world were their agents live on. This is done mostly because it is easier programmatically to deal with a discreet world than it is to deal with a space continuum.

2.2.6 GIS visualization

There are many GIS software packages in the market, and one of the most used is the ArcGis [4] suite from ESRI. As part of this suite there is ArcMap, one of the best traditional applications to work with maps, geographic data and relational databases.

In the last years the appearance of Google with its Google Earth [11] has brought the GIS to the common computer users. The new software was a complete simulation of the planet Earth covered with textures taken from satellite pictures and airplanes. The main advantages of Google Earth when compared with the traditional GIS is that it is fast, it is free, supports custom user content, retrieves local information and allows a series of customizations like building 3D houses or creating flyby animations. Traditional GIS such has ArcMap are better suited for scene querying and analysis, do not have much render capabilities but compensate by having a sophisticated database system.

Another interesting geographic API is the Google Earth's KML API [12]. This is an XML format that can be used to represent almost any feature on the Google Earth's application.

There are many aspects of these two approaches that can be mixed to create a better visualization system. Additionally, there are studies made that test different configurations of 3D Views and viewports on a GIS visualization system. In this particular system described in [26] special emphasis was given to the representation of context symbols such as North indication, camera view direction or lines linking the camera viewports to the actual place in the world where the camera is.

2.3 Interaction approaches

As mentioned, beyond designing an intuitive visual interface one of the objectives of this thesis is to study new interaction paradigms and apply them to the dam emergency break case study scenario. The next sections describe different interaction approaches.

A new interface for handling geospatial information is proposed in [17]. The study indicates that typical GIS are not suited for multi-user access and high-level abstract queries. The main problem detected is that currently decision makers do not get the information they need in real time and often depend on GIS analysts to produce maps at their request. The challenge here was to create a multimodal interface design that was not analyst-driven, menu-controlled or keyboard and mouse operated.

The solution presented relies on computer vision and speech processing as a means of interpreting and integrating information from two modalities, spoken words and free hand gestures. The users stand in front of a large display panel where they can see the maps and the GIS. A camera pointed to the users detects motion and a microphone listens to commands spoken. Speech is best suited for direct actions, expressing orders, pronouns and abstract relations. Speech is not self-sufficient and therefore gestures are used for expressing spatial relations.

A prototype using these techniques was made to test this new paradigm. This application follows a client/server architecture. The client side includes the input devices and the handlers that receive images, gestures and speech. There can exist multiple clients receiving information from multiple sources. The clients communicate with a central server across the network. The server collects all input given by the clients, processes it and reacts accordingly. If necessary, the server communicates back to the clients to change the information display with new maps or some feedback. The main goal of the system is to achieve a multi-user collaborative interactive tool that can handle GIS faster and in a simpler way.

Visual interaction is a topic that has been around for some years. One of the main tools that have been used to interpret visual information is OpenCV [29] or Open Source Computer Vision Library. This is a tool meant to process information from images or cameras in order to detect motion, moving objects, tracking color blobs or helping algorithms that use visual interpretation. Using this library many applications recognize human gestures or the relative position of the body. It is also useful to detect certain objects with certain characteristics. Another way to track objects is through fiducial markers as described next.

2.3.1 Augmented reality

Augmented Reality (AR) is a technology that allows "real time superimposition of synthetic objects on real images, providing an augmented knowledge about the surrounding world" [37]. This typically implies that an application using AR will have a camera where it captures the real world and the captured image is enriched with artificial content. This artificial content can be placed taking into consideration the user position or special markers in the real world. The users can even interact with a computer system, changing the position or rotation of these markers [6].

One example of the use of AR technologies is ANTS – Augmented Environments [37]. This project builds an AR infrastructure so that users can explore physical and natural structures for environmental management purposes. The system proposed uses a Head Mounted Display were the viewer sees the world with extra information that helps him move around. The system also supports mobile phones and PDA's.

One of the tools that has been widely used in tangible and augmented reality interfaces is the ARtoolkit [6]. This toolkit is used to detect special markers in the images captured from a camera, and giving them virtual coordinates in a 3D word. This coordinates can then be used

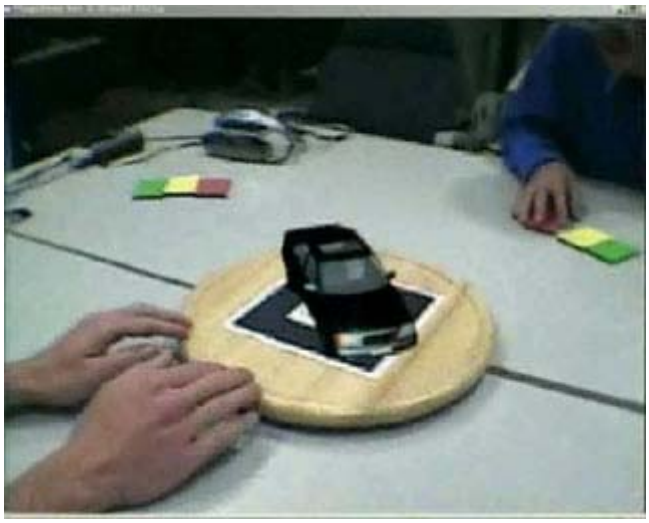

Fig. 2.5– An Augmented Reality example using AR toolkit

to create virtual worlds on top of the real image creating an augmented reality. An example can be seen in figure 2.5.

2.3.2 Tangible interfaces

People are already used to interact with computers through a small window display, which is always in front of them, “you sit for so many hours at a time, staring at that small rectangular display of information: it is always the same focal length, with no relief for the eyes; it makes no use of your peripheral vision; it is so dim that you have to control the surrounding lighting conditions to see it properly” [7]. A new approach for this problem is to make use of other senses such as touch and gestures.

To use touch, new input devices with tactile feedback should be designed. The Tangible Media Group [35] is a research group led by Hiroshi Ishii at the MIT Media Group. In this group they make experiments on how to make objects, which have ubiquitous sensors, to interact in an intelligent way using computation. One of the more interesting features of their work is that many times the person doesn’t really perceive that it is interfacing with a computer.

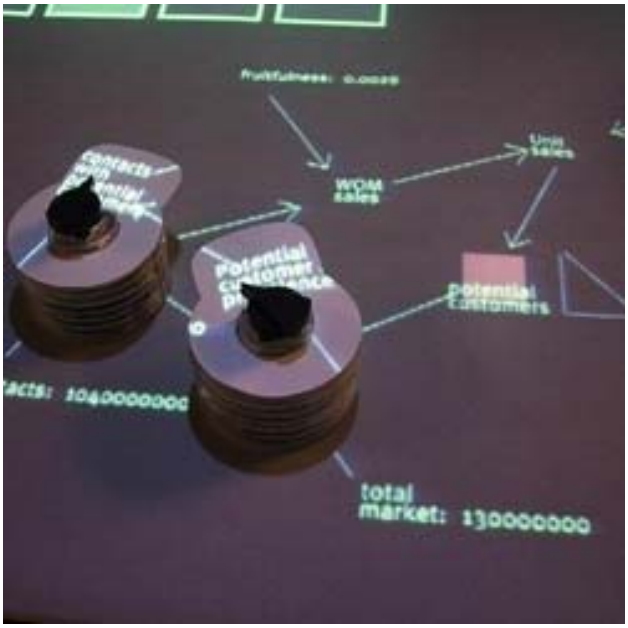

Fig. 2.6– Tangible Sensetable

One example is the Disaster Simulation System (DSS) that was already mentioned before. The Tangible DSS takes advantage of an input device called Sensetable. The Sensetable [18] is a wireless object tracking platform that has been used for Tangible User Interfaces (TUI) [39]. This is a system that electromagnetically tracks the positions and orientations of multiple wireless objects on a tabletop display surface (fig.2.6). There is a great difference between this system and others that rely on camera and gesture tracking approaches. This system tracks objects quickly and accurately at a high refresh rate without susceptibility to occlusion or changes in lighting conditions. Additionally, the tracked objects have state that can be modified by attaching physical dials and modifiers. The system can detect these changes in real-time. The Sensetable has a tabletop configuration. It has a table with a projector, directly above pointing to it. This displays graphical information on the table and on the tracking objects. The interface is tangible with multipoint interaction suitable for two-handed manipulation and collaborative work. One interesting use of the Sensetable is the Reactable [21,24], using the tabletop as a music instrument.

The Reactable is a music edition project using an interactive multimedia multi-touch table to modulate the sound and change the music in a collaborative way. It uses an

implementation of the TUIO [23] protocol called ReactVision [31]. This implementation has servers, clients and also a virtual simulator. This simulator allows testing TUIO applications without actually having a physical multi-touch table. This was later used in the implementation of the prototype.

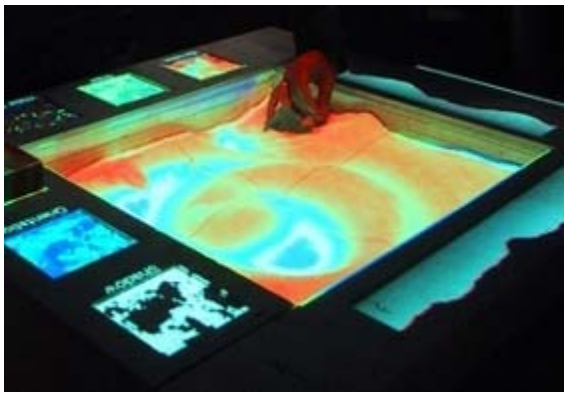

Fig. 2.7– Tangible Sandscape technology

The Sandscape [7] is also a very interesting interface that consists of a sand box that is being regularly scanned for any change in its relief (fig.2.7). It is a “tangible interface for designing and understanding landscapes through a variety of computational simulations using sand. Users view these simulations as they are projected on the surface of a sand model that represents the terrain” [35]. There is a variety of different simulations that highlight the height, slope, contours, shadows, drainage or aspect of the landscape model. The users can alter the form of the landscape model by manipulating sand while seeing the resulting effects of computational analysis, generated and projected on the surface of sand in real-time.

2.3.3 Multi-touch interfaces

Multi-touch technologies have been around for some years but mainly in research centers and universities. Only now, with the appearance of the Apple IPhone, Microsoft Surface and Jeff Han’s interface, the general public started to show some interest for this interaction technique.

In his site Bill Buxton [6], an expert in human-computer interaction, explains some concepts in the area and clarifies several concepts related with multi-point, multi-touch, touch screens and touch tablets. He also states that the multi-touch devices still have some time to reach maturity when he says “Remember that it took thirty years between when the mouse was invented by Engelbart and English in 1965 to when it became ubiquitous, on the release of Windows 95. Yes, it was released commercially on the Xerox Star and PERQ workstations in 1982, and I used my first one in 1972 at the National Research Council of Canada. But statistically, that doesn’t matter. It took thirty years to hit the tipping point. So, by that measure, multi-touch technologies have five years to go before they fall behind”.

Multi-touch interfaces greatly expand the gestures and actions that can be used to interact but they may not completely substitute the mouse. This is because the mouse is a mature and precise input device and for some tasks it is the best. In the end, the users must use the best devices for the task. Additionally, Bill Buxton makes some important remarks about multi-touch interfaces:

- Touch interfaces will always be worst then the real objects. A graphical keyboard will always feel worst than a real one.
- Not suitable for blind people.
- Handhelds that rely on touch screens for input virtually all require two hands to operate.
- The finger size matters no matter the size of the screen
- We don’t do finger painting because it is not as effective as using a pencil.
- Sometimes it is difficult to see the color on LCD displays when you’re outside trying to use the device.

One of the more interesting recent interfaces is the one proposed by Jeff Han. In his most notorious publication [15] he explains how to build a low cost Multi-Touch sensing interface

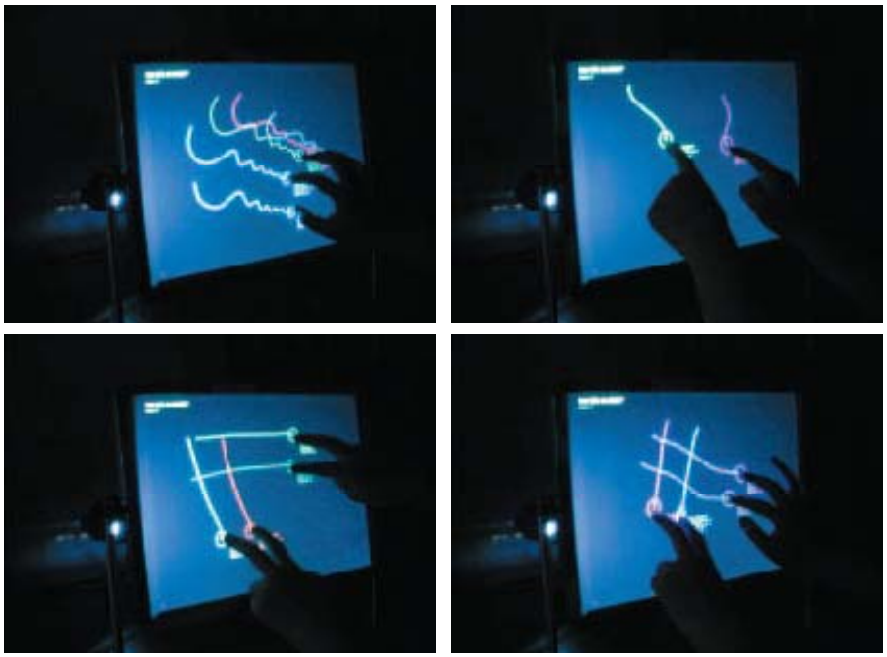

Fig. 2.8– Jeff Han's multi-touch interface

using a technique called Frustrated Total Internal Reflection (FTIR). This technique consists on using a display made of acrylic with some diffuser material in one of the sides and rear-project the images on it (fig.2.8). On the edge of the acrylic are placed several infra-red LEDs, and when the user touches the acrylic screen some light is reflected. An infrared camera on the back of the display captures the light. The captured image will have several points and blobs that can be computer tracked and used as input for applications, as seen in the image 2.8.

More important than the physical device are the applications that can be done using multi-touch. In a presentation at TED Talks [19], Jeff Han shows a number of applications of the technique. The main advantage of using multiple fingers is spatial navigation and collaboration; being able to move things around in the desktop while another person sits next to you and interacts also with the screen; rotate things with two fingers or scale using the two hands. The interface becomes invisible and the user just has to do the system defined gestures to perform actions.

In 2007 Microsoft released a tabletop called Surface. Although the system has only been shown in conferences it is the first commercial desktop experience with multi-touch. At the same time Apple released the IPhone which has a multi-touch screen.

3. Design Architecture

This chapter presents the design options to build a solution capable of representing the simulation scenario presented in the case study. This chapter explains the several steps that were necessary to implement it.

3.1 Concept

The initial proposal was to create a prototype capable of representing the case study scenario in the Alqueva Dam. This application constitutes the Visualization component of the Life-Saver project. The Visualization component is responsible for displaying the information that comes out of the simulation and must have an intuitive interface.

In order to visualize the scenario case, the multimedia interface must fulfill certain requirements. It must show a full-fledged three-dimensional terrain with a complete and accurate representation of the valley. Important structures and geographic features like the dam, the river or the retained water should be represented. Additionally, there is the need to represent vehicles, people, buildings and other objects. All these elements must communicate through the network with the simulator, which defines their actions. The interface must be intuitive, easily manageable by people with no special skills in computer interaction.

The computer games [8,25] area provides good insights on matching some of these requirements. The idea is to make the emergency visualization interface to look as close as

possible as a RTS game (Real-Time Strategy game). In a RTS game the world is seen from above, the units work as agents with limited artificial intelligence and information about each unit or building can be retrieved by clicking on it. Many RTS games have sound alerts and provide several shortcuts that enable a quick response by the user to any emergency.

In terms of interaction the application should support several input devices like the mouse, keyboard, tablets, interactive touch boards and multi-touch devices. All the infrastructure required to do that will be detailed later in the interaction chapter.

The final requirement is to build an application that uses a game engine library to render the objects. The application should be built in a high performance object oriented language such as C++. Every piece of code or component that can be generalized should be placed in a separated library, for later reuse.

3.2 Choice of technologies

As described, the prototype application uses a game approach, and was implemented with an open source game engine called Ogre3D [28]. This C++ library supports most of the features and effects currently available in games. The main advantage of using a game engine is that it is easier to create 3D content. Ogre3D has a set of exporters and plug-ins for almost all the major 3D manipulation programs such as Blender, 3DS Max, Maya and MilkShape 3D. In this way it is possible to create all the objects and scenes using external applications and load them whenever it is necessary into the program.

The Ogre3D (Object-Oriented Graphics Rendering Engine 3D) library has a complete set of classes and functions to control the scene object-graph. The objects are placed in a tree of nodes. Each node can be rotated, translated or scaled to position objects and create animation effects.

It includes a complete scripting system, where the developer can write scripts for materials, screen overlays, fonts and particle systems. This allows making small changes to the application without recompiling it again.

Ogre3D is a scene-oriented, flexible 3D engine written in C++ designed to make it easier and more intuitive for developers to produce applications using hardware-accelerated 3D graphics. The class library abstracts all the details of using the underlying system libraries, like Direct3D and OpenGL, and provides an interface based on world objects and other intuitive concepts, available as classes.

Although it contains a very good graphical support, Ogre3D does not have several important features usually needed like networking, sound management, physics, GUI or artificial intelligence. It has however a good reference and documentation [13] about how to integrate Ogre with other libraries which can provide the necessary functionalities.

Since it is an active open source project there is a large community that posts many problems and discussion on the internet and extends almost daily the documentation of Ogre3D. Another good point is that it has many plug-ins written by the community. The main drawback of this has to do with sometimes the functionality and code not being so well documented.

To complement Ogre3D functionalities the PLSM2 [30] was used. It is a plug-in that can create a large, almost endless, terrain and load it using a fast octree model.

To create the internal GUI in the application the CEGUI library was chosen mainly because of its easy integration with Ogre3D.

The prototype application uses a large amount of geographical information. This information was provided by the Life-Saver's project partner LNEC. All the data collected is on ESRI ArcGIS[4] format.

Additional technologies were used and will be detailed more precisely in the implementation. Most of these technologies are free or open-source. Among these technologies are:

- GSoap[14], for using the web services in the communications between modules.
- FMod[10], for sound effects.
- ShapeLib[33], for reading ESRI's proprietary file format ShapeFile.

3.3 Architecture

After establishing the requirements and choosing the technologies it was necessary to make an analysis study of the future application. A Use Case model was employed to define the main functionalities and a preliminary class diagram was created. This diagram was iteratively modified during the implementation with some improvements. These improvements addressed issues that were not predicted or were changes to circumvent problems. From the preliminary class diagram several modules in the application were identified:

- A set of classes that abstracted Ogre's functionalities.
- A module to facilitate the construction of primitive objects and lines.
- A set of classes that would take care of all communications with the simulator.
- A module that would take care of all the GUI windows.
- One input-output module.
- A geographic elements representation module.

The entire application was built with C++, using an object-oriented and event-based approach. As mentioned, the programming infrastructure is provided by the Ogre3D game engine.

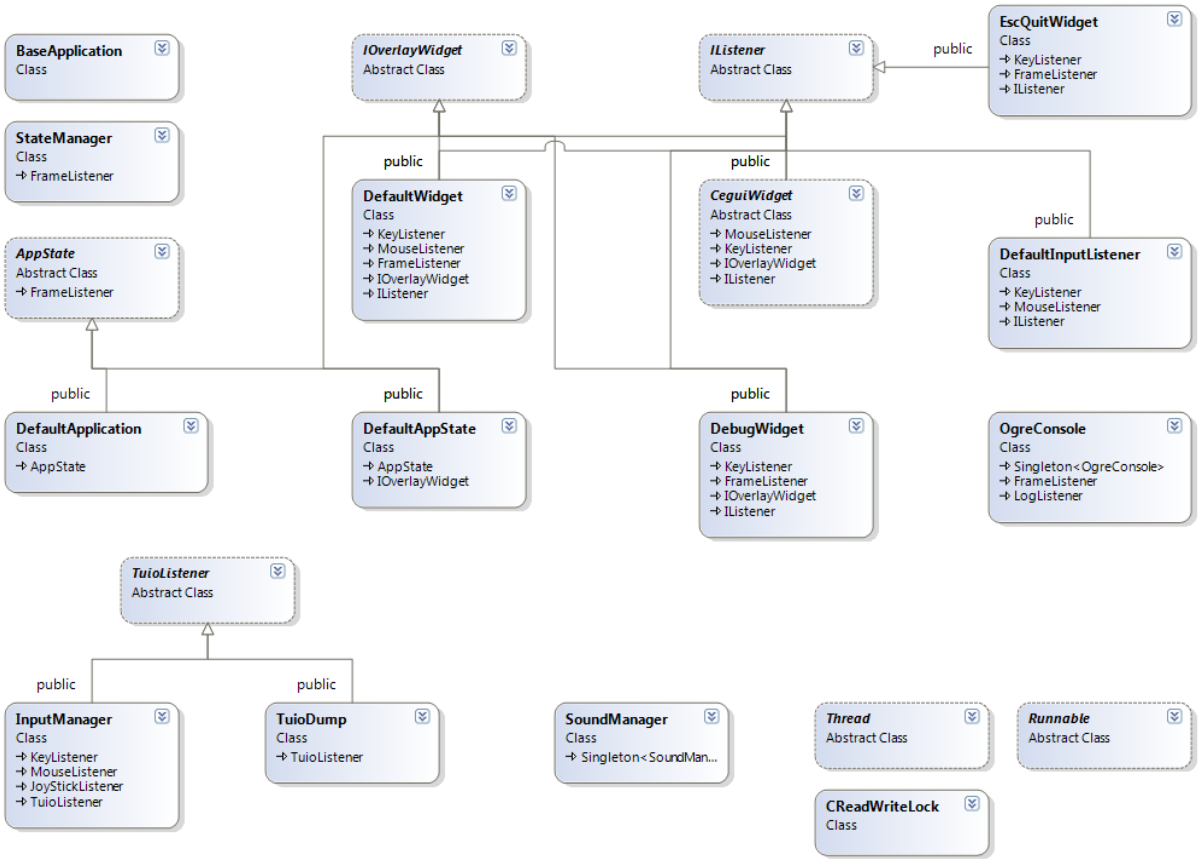

Fig. 3.1– BaseProject library: class diagram

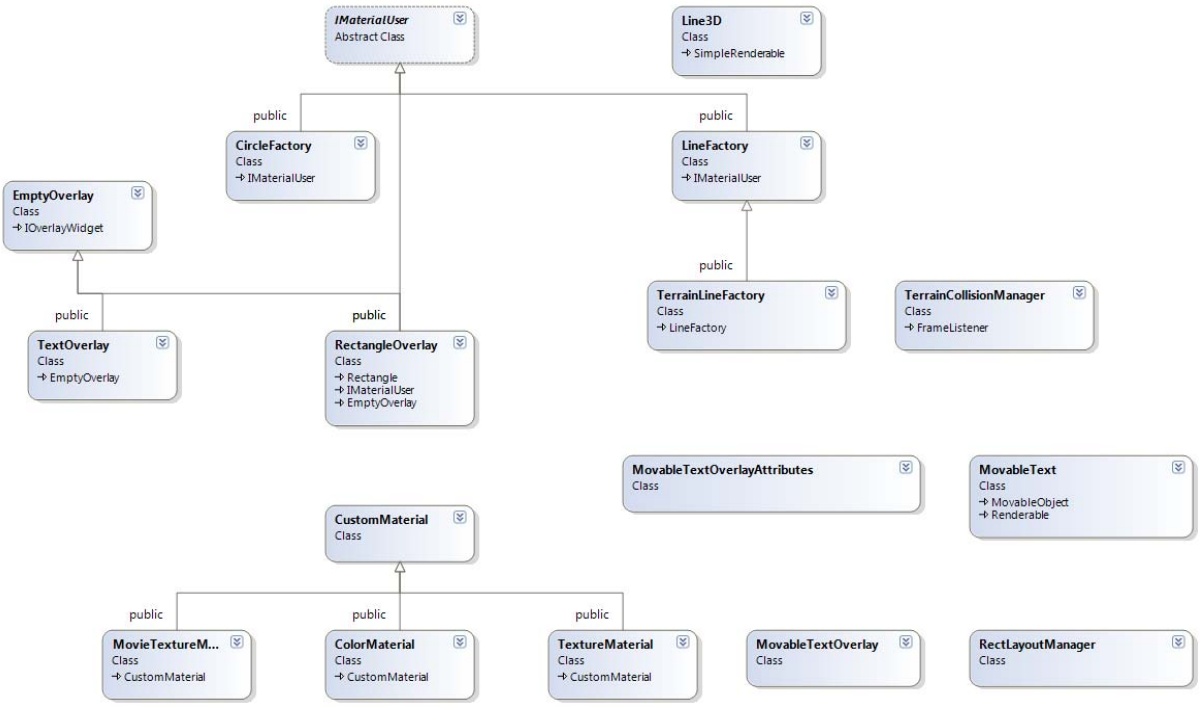

Fig. 3.2– UtilityProject library: class diagram

All the modules identified were divided between two C++ libraries and the application. One called “BaseProject” (fig. 3.1) acts as a framework that takes care of all Ogre3D initializations, IO management, scene and level context and sound management. The other library is “UtilityProject” (fig 3.2), an utility toolkit used to create simple custom objects, custom lines, custom materials and additional useful resources. The application “TerrainStudio” takes care of all specific code that will not be reused.

The code was organized with a set of Manager classes. These follow the “Singleton” code pattern, which means that for each class there is only one instance of the class running. This is important when, for example, it is necessary to have a global InputManager class that must be accessible from many parts of the code and has to be unique in the running program.

One of the manager classes is the StateManager. This class runs the main render loop of the program. The main advantage of this class is that it has a stack of Scenes. Each Scene is a state of the program and it can be a new level or a configuration scene. Each scene can be “pushed”, “popped” or redirected to other scenes. This allows the fast creation of Scene flows. One Scene is a class that implements the AppState abstract class seen in figure 3.1.

Another important method was the use of an event based approach. When an object interacts with the mouse it is registered as a listener in the InputManager and then the desired handlers are implemented in the object’s class. This allows a decentralized approach, where each object has its input code, as opposed to having all the input treated in one place. The same happens with the graphical user interface.

This application is part of the Life-Saver project which includes a set of applications that communicate with each other through the Internet. The communications with the simulator are done using XML Soap Web Services. This is supported by using the GSoap [14] library.

To study the application’s interface, several informal inquires were made to five volunteers. Most of them agreed with the visualization as a RTS Game. The main locations of

the elements that constitute the interface were decided during this process. The interface was refined using several input devices and always validated with informal talks and experimentation.

3.4 Implementation

This chapter puts in evidence some of the details and difficulties that were found during the implementation of the prototype application. The implementation of the project was made using Microsoft Visual C++. In figure 3.3, it is possible to observe the library hierarchy that was implemented to create the visualization system.

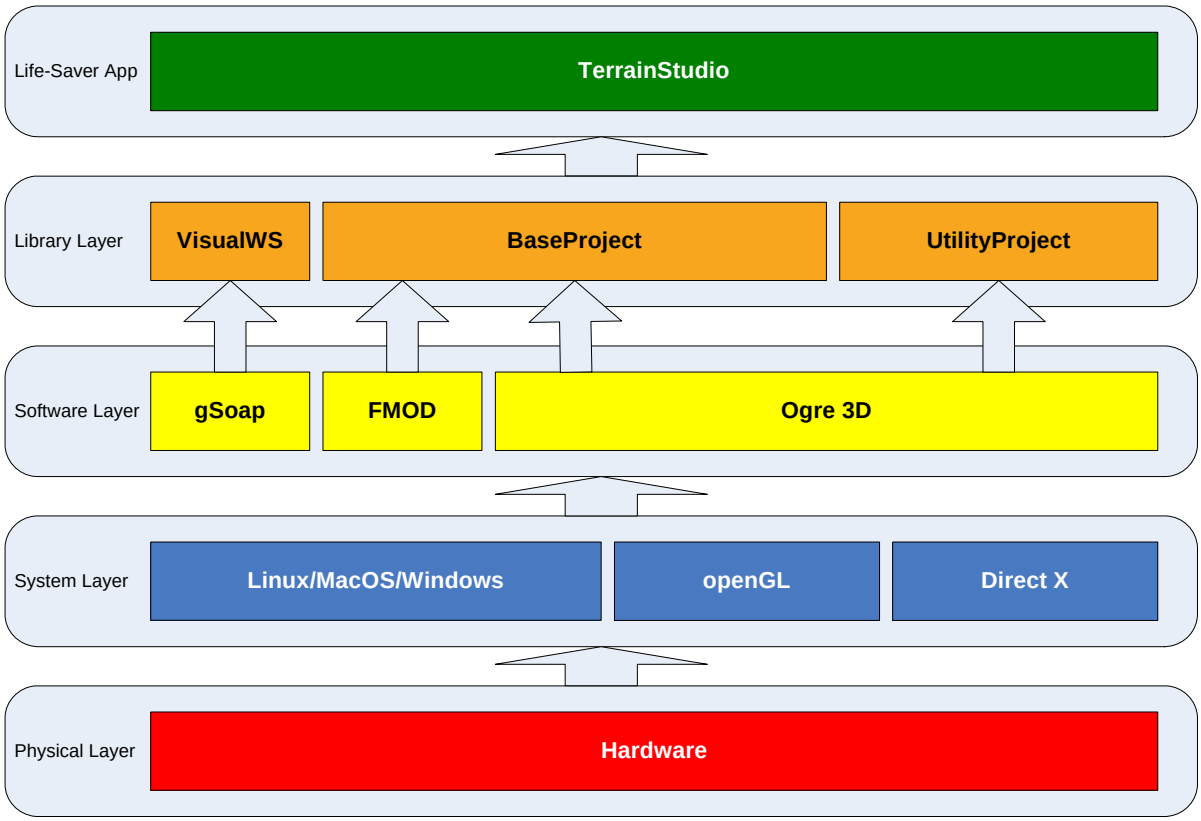

Fig. 3.3– Diagram of the Visualization Component. This thesis work focus on the two top layers.

3.4.1 Terrain creation

The creation of the terrain was essentially made in three steps. On the first step the terrain data was manipulated. On the second the information was converted into height images and prepared to be loaded into the application. Finally, in the third step the information is loaded and displayed in the program.

The Paging Landscape Scene Manager (PLSM2) [30] is an Ogre3D plug-in to render terrains. The main advantage of this plug-in is that it can render almost endless amounts of terrain and still render the scene fast (above 30fps). The system uses an octree system to manage the level of detail displayed for each square of terrain. Taking into account the camera position, the system loads and unloads information from the octree. If a certain area is not in the scope of the camera, that area is unloaded from memory. If an area is too far from the camera, then that area is rendered with a lower level of detail.

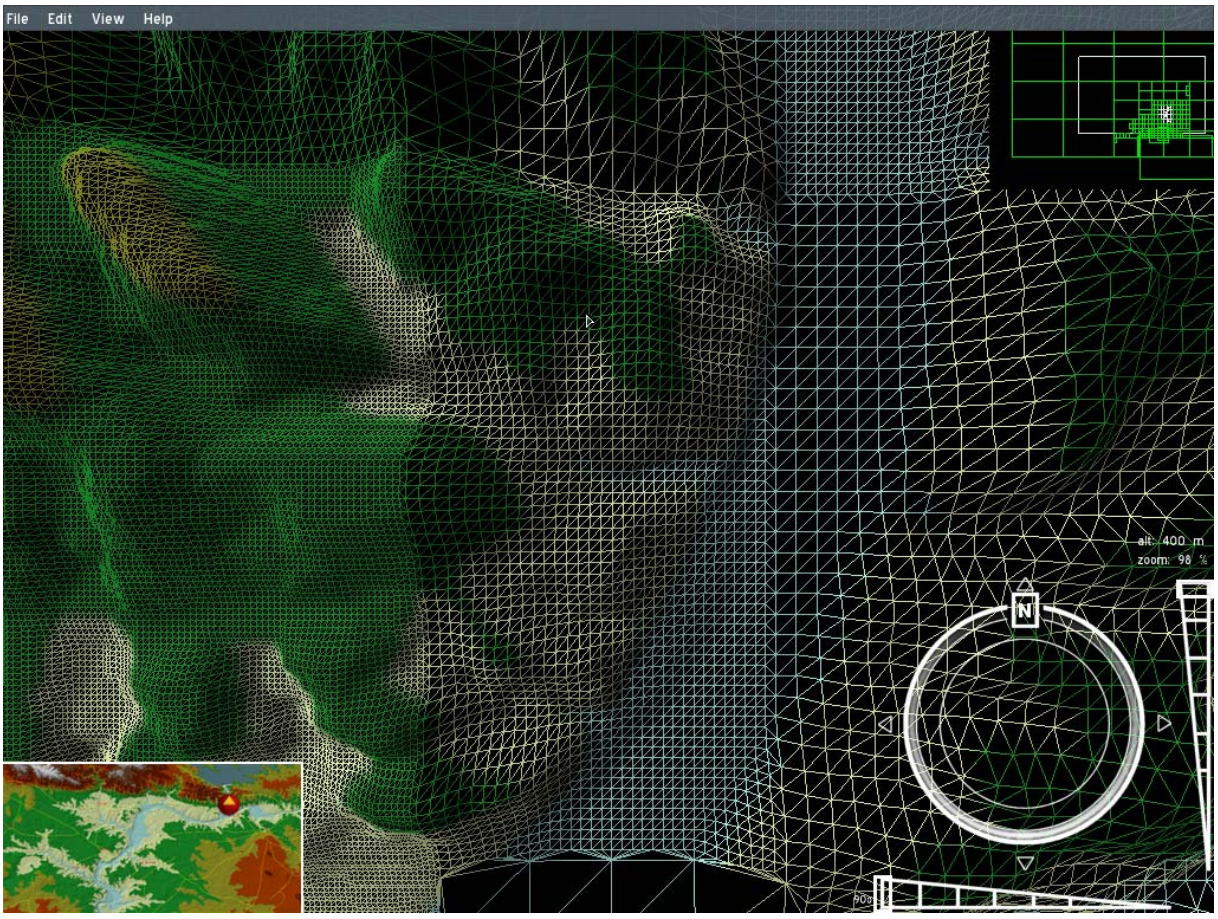

Fig. 3.4– PLSM: different levels of detail of the terrain. Octree diagram in the top-right corner.

PLSM requires two sets of images to work. The first is a set of grayscale square pictures that define the height of the terrain. Black is deeper and white is higher level terrain. The size of the squares defines the highest level of detail. If they are small, they are fast to load, but there is more overhead in computing. If they are large they are slower to load. In this application it is used a size of 512 by 512. It is recommended that the size should be a power of 2.

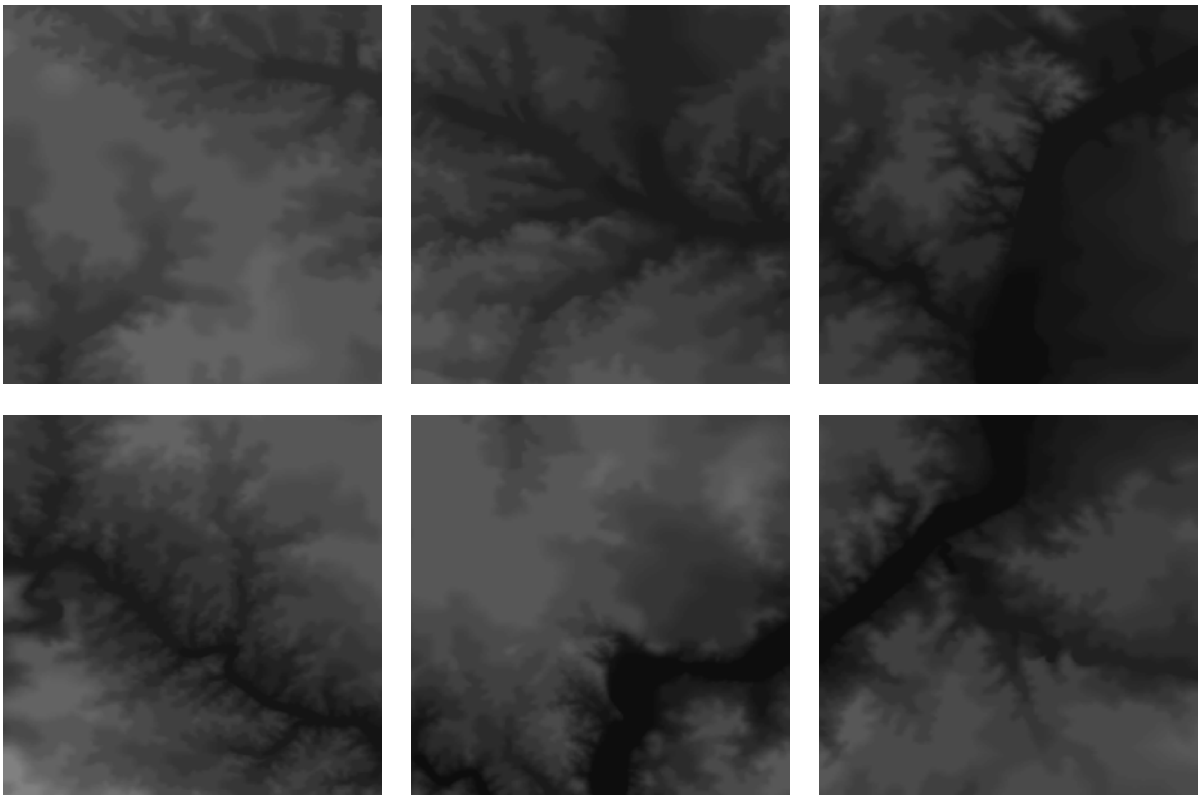

Fig. 3.5–Height map divided in squares.

The second set of pictures are the textures that will be mapped to the polygons that were created with the height map as seen in figure 3.5. These pictures are generated to create a full representation of the Guadiana valley. First of all, it was necessary to get information of the terrain. With the help of LNEC a large amount of data was collected. The data included regional information of the Guadiana valley such as: ArcGis ShapeFiles and ArcMaps, military charts and aerial ortho-photos.

The military charts had the contour lines of the terrain. Using ArcGis to manipulate the charts it was possible to extract those lines and construct a 3D model of the terrain. Additionally, there was information in the Shapefiles about the height of some topographic points. Using the contour lines and the points two models were created that represented the different heights of the terrain. The first model was a grayscale map with 256 levels of height; this was used as the height map in the PLSM. The second model was a false color model to serve as texture. The two models were rendered into two large high-resolution images. Finally, these two images were splitted into 512 by 512 squares by an application called MapSplitter that comes with the PLSM. To use the PLSM it is necessary to write a small script where the developers can tweak some details about the terrain. To load it inside an Ogre application it is necessary to call a special Scene Manager as in this example:

```
CustomPagingLandScapeListener *paginglandScapeevents;

mSceneMgr = mRoot->createSceneManager("PagingLandScapeSceneManager");
//call the terrain configuration file
mSceneMgr->setWorldGeometry(Ogre::String("paginglandscape2.cfg"));
mSceneMgr->setOption ("ConfigGroupName", &String("PLSM2"));
paginglandScapeevents = new CustomPagingLandScapeListener (mSceneMgr);
```

To set options of the terrain there is the function `setOption`. Unfortunately many options are undocumented and can only be understood by looking into PLSM's code. Here are some examples:

```
int MaxAdjacentPages = 4;
int MaxPreloadedPages = 6;
Real DistanceLOD = 6;
mSceneMgr->setOption( "MaxAdjacentPages", &MaxAdjacentPages);
mSceneMgr->setOption( "MaxPreloadedPages", &MaxPreloadedPages);
mSceneMgr->setOption( "DistanceLOD", &DistanceLOD);

//load terrain NOW!
mSceneMgr->setOption ("LoadNow", mCamera);
```

These are the settings that are currently being used although they change with the zoom of the camera.

3.4.2 Shapefile manipulation

Much of the information about roads, house locations or river flooding lines was in ShapeFiles. The ShapeFile format is a proprietary open format from ESRI. With the success of the ESRI product ArcGis wich manages geographic databases and information systems and its widely adoption, the ShapeFile became a standard *de facto*. Although it is a proprietary format it is open, its specification and documentation are public. This allows that several applications and libraries use this format. The ShapeLib is a C library used to read the geometric information of the ShapeFiles (.shp) and their databases (.dbf). The ShapeFiles contain information about points, lines or polygons. Each file can contain several entities; each entity is composed of several parts, each part contains all the vertex coordinates that compose that part. Associated with each entity there is an entry in the database that can contain several fields with information regarding that entity. This is an example of how to extract all the information from a shapefile:

```
int entities;
SHPObject *shp;
SHPHandle fp = SHPOpen( filename , "rb");
SHPGetInfo( fp, &entities, &type, NULL, NULL );
//for each entity
for( int e = 0; e < entities; e++ )
{     shp = SHPReadObject( fp, e );
      //for each part
      for( int i=0; i < shp->nParts ; i++ )
      {     if( i == (shp->nParts - 1) )
                  end = shp->nVertices;
            else
                  end = shp->panPartStart[i+1];

            for( j = shp->panPartStart[i] ; j < end ; j++ )
            {
               //Using one vertex
               Vector3( shp->padfX[j] , shp->padfY[j], shp->padfY[j]  ) ));
               //(...)
            }
      }
      SHPDestroyObject( shp );
}
```

One of the main problems encountered in the database was the differences between the character sets of the ShapeFiles and the one used by Ogre3D. The solution to convert all the

characters to Portuguese recognizable letters was to map all the characters to the small UTF8 character set. This code converts each character by ignoring all bit codes above the 8th bit:

```
int i;
Ogre::UTFString UTFString;
Ogre::UTFString::code_point cp;
for (i=0; i<(int)String.size(); ++i)
{
   cp = String[i];
   cp &= 0xFF;
   UTFString.append(1, cp);
}
```

This example only works in languages that are fully supported by the UTF8 character set.

3.4.3 Main loop and threading support

Threading support was fundamental to implement the prototype. The main program thread typically starts with all the initializations and scene loading. When it finishes, it starts the main loop. The main loop is responsible for rendering all frames. If we want to do some action before or after rendering a frame it is necessary to create a Frame Listener. A Frame Listener is a class that extends the Ogre::FrameListener class and implements the following methods:

```
virtual bool frameStarted(const FrameEvent& evt);
virtual bool frameEnded(const FrameEvent& evt);
```

The Root node of Ogre is used to add the Frame Listener class:

```
mRoot->addFrameListener(myFrameListener);
```

Taking this into account, the main loop will call all the `frameStarted` functions of all the listeners, then the scene is rendered and afterwards all `frameEnded` are called. This is very good for moving objects; all the translations can be made before the frame is rendered. With the help of the FrameEvent it is possible to know how much time elapsed since the last frame. This allows creating constant movements using the equations of movement.

Another advantage with this main loop is a fast unbuffered input-output. With this method the input is detected on each frame. It is not very recommendable with low frame rates because the signals are not buffered and some might be missed.

The main problems happen when it is necessary to implement blocking Web communications or buffered event-based input output. The solution to this problem is to implement a system with several threads where each thread receives the information continuously from the web services and stores it in a shared memory (fig.3.6). The main loop thread then fetches the information on each frame when it is possible. To handle the producer/consumer problem generated by this solution read/write locks are used.

The implemented solution used the Windows threads library but the code is well isolated so it is simple to change for other systems.

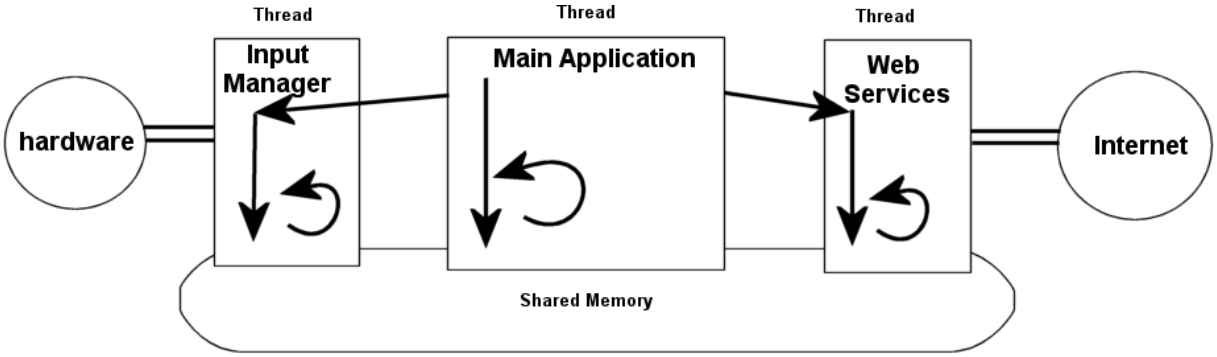

Fig. 3.6– Thread solution.

3.4.4 Web service comunications

The communications between components were decided within the scope of the overall Life-Saver project. It was decided that the connections would be made through Soap web services. The web services would not be, as they usually are, served by a permanent HTTP server such as Apache or IIS but by stand-alone applications. This stand-alone applications create the Web services on start-up. Clients connect and interact with the server. In Life-Saver the main Web server is called VisualWS as depicted in the Introduction chapter in figure 1.2.

In this Web service the Server is the Visualization Component and the client is the Simulator Component. Although this may seem counter-intuitive it makes sense from an action perspective. The Visualization is waiting for actions to happen. When the Simulation has something new it calls the Server (the Visualization) and informs it of the new information.

The Web service server was created using the GSoap [14] C++ library. This is a simple library capable of generating stand-alone applications to serve soap requests. Below is a sample of the implemented server:

```
struct soap soap;
struct soap *tsoap;
soap_init2(&soap,SOAP_IO_KEEPALIVE,SOAP_IO_KEEPALIVE); // initialise gSOAP
m=(int)soap_bind(&soap, NULL , 8082 , 100);     // http://localhost:8082
while(!done) {

      s=(int)soap_accept(&soap);
      if(s<0) {
            soap_print_fault(&soap, stderr);
            break;
      }
      else
      {
            tsoap = soap_copy(&soap);
            //Call functions that implement the we service
            if( soap_serve(tsoap)!= SOAP_OK){
                  (...error... )
            }
            soap_destroy( tsoap );
            soap_end(tsoap);
            soap_done(tsoap);
            soap_free(tsoap);
      }
   }
```

The code above would create a web service on the machine in the address `http://localhost:8082` . Whenever a client connects and calls a function, the call to `soap_serve()` will pass that call to the function that implements that functionality. For example a call to function void `setTick(int tick)` by the client would be redirected to a function such as the following:

```
int __ns1__setTick(struct soap*, ns1__setTick *ns1__setTick_,
                   ns1__setTickResponse *ns1__setTickResponse_)
{
      int tick = ns1__setTick_->arg0;
      //(...)
      return SOAP_OK;
}
```

This function has to be implemented by the developer.

3.4.5 Tridimensional labeling

After creating the terrain, and loading the shapefile information, one of the problems encountered was how to display text information in the 3D world. One example is how to display the name of the cities above their location spots. This problem must take into account that the user may zoom, rotate and tilt the display camera and there can be many labels in the same spot. To solve this problem two approaches were considered.

The first approach was an invisible 3D rectangle created above the spot. This rectangle has a texture with the label text. The 3D rectangle is always updated on each frame to always be facing the camera. This solution works (fig.3.7), but has some drawbacks. When the user is very near the letters seem too big, too high above the ground or blurry. Far away the letters seem small or blurry if a scale is used to increase their size.

Fig. 3.7– 3D rectangle label: near and far

The second approach makes use of overlays. Overlays are 2D elements that are directly printed on the screen with 2D coordinates on top of the 3D scene. The idea here is to built a 2D rectangle with the label text. This rectangle is placed on the screen above the desired point on the terrain. Because the overlay is printed on the screen, it is always facing the camera. On each frame the only thing that is necessary to do is to update the rectangle position to match the point in the 3D world (fig.3.8).

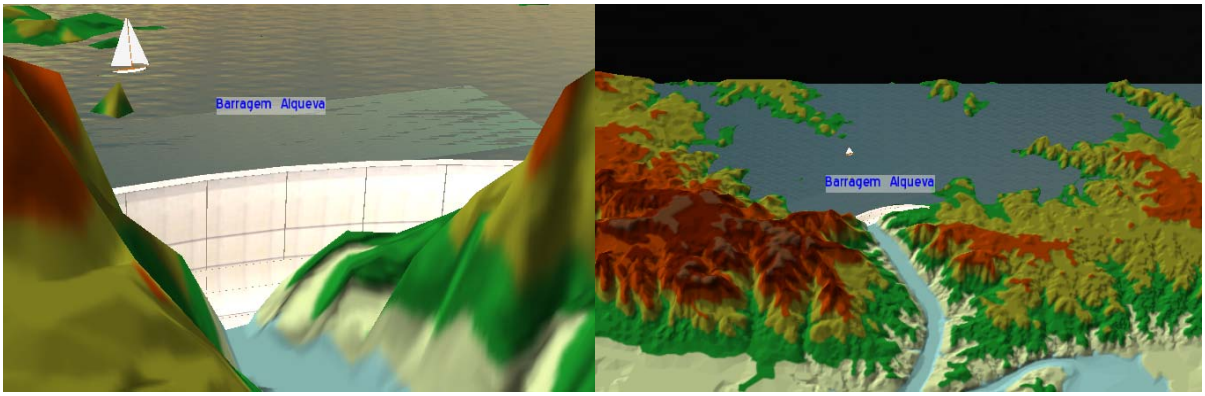

Fig. 3.8– 2D rectangle label: near and far

The second method prevents heavy 3D calculations to rotate the label to the camera and because the rectangle is 2D it is computationally fast to calculate and avoid overlaps with other labels.

3.4.6 Mouse object selection

To create a usable geographical representation application it is required that the users can select objects on the terrain and be able to interact with them. Ogre3D has a set of tools that help to detect objects. The main technique consists in detecting the mouse position and using ray-tracing to detect the objects on the scene. Each object on an Ogre3D scene has a bounding box. When the ray that comes from the screen crosses that bounding box a collision is signaled. This works fine if the object is massive and fills its bounding box. In a collision with

an irregular object the selection will look unexpected to the user if he clicks on an empty space and just because it is within the bounding box the object is selected (fig.3.9)

Fig. 3.9– Object bounding box.

The proposed solution uses two types of collisions. Terrain object such as city pins, houses, cars or people are selected directly through their bounding box. Irregular objects such as roads, flooding lines or wave targets are selected through a collision map. This collision map is a grid that is filled in the beginning of the program. Each square on the grid represents a small area of the map. If the user clicks on that area then a collision with the associated irregular object will be signaled (fig. 3.10). In summary, the main algorithm has the following behavior:

```
init:
      create map collision
      create map of irregular objects


Mouse Click triggered event:
      raytrace from screen(x,y) to scene(x1,y1,z1)
      for each collision
            if terrain
                  stop
            endif
            check map of irregular objects
            if irregular
                  ignore
            else
                  signal collision!
            endif
      endfor
```

```
if no collision found
      check collisionMap (x1,z1)
      if has associated irregular
            signal collision!
      endif
endif
```

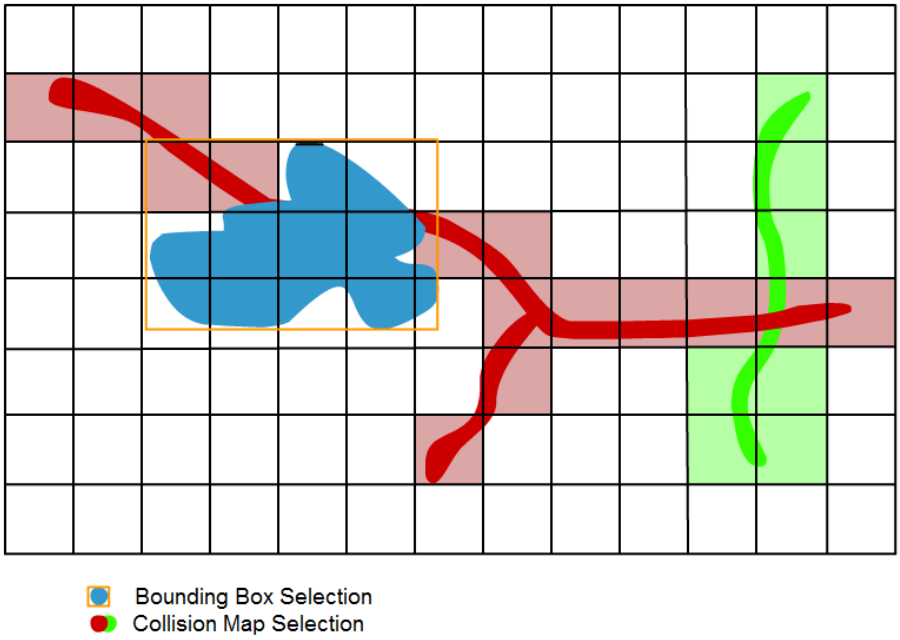

Fig. 3.10– Selection through Bounding Box and Collision Map

With this algorithm, priority is given to the objects in the scene and less to irregular object such as roads.

3.4.7 Simulation recording

The objective of the simulation component is to simulate the behavior of the authorities and inhabitants intervening in the flood scenario. The simulator feeds the Visualization component through the Web services with information about the wave status, the agent status and the people status. Each time an agent moves, the Visualization component is informed with its new position and the new position is updated on the scene.

To record the entire simulation it is necessary to record all status change of all agents. The state of an agent is defined to be A = {id, tick, pos {x,y}, rotation, status}. The simulation runs by ticks. A tick is the minimum time unit of the simulation.

The implemented approach saves only the state A of each agent when it changes. To go to a specific point in time T it is necessary to get the last change of each agent before tick T. This is a much more efficient solution than to save the state A for each agent for all the ticks. If the simulation had 100.000 ticks and about 100 agents it would cause heavy memory consumption (above 200MB).

3.4.8 Camera navigation

The navigation on the scenario can be made with four basic operations: (1) move, (2) rotate, (3) zoom and (4) pitch.

1) The move operation is triggered when the user presses the mouse button on any part of the scenario that it is not occupied by the user interface. The concept is that when the user drags the mouse the terrain appears to move along with the cursor. In reality the terrain doesn't move, what really moves is the camera. For the movement to appear realistic the camera must move on the opposite direction of the mouse movement. This is a common computer graphics trick [25]. The real question is how much it is necessary to move the camera to create the illusion that the user is pulling the terrain.

For the sake of simplicity the following equations are just for one dimension. In a two dimensional screen it is necessary to repeat the calculations for the two coordinates.

With X being the necessary amount of translation of the camera, Rel the relative movement in pixels of the mouse, Alt the current altitude of the camera and k a movement constant dependent on the screen resolution:

$$X = -Rel \cdot Alt \cdot k \quad \textbf{(3.1)}$$

Since the altitude of the camera can be changed with the zoom operation, it is necessary to compensate the amount of camera movement that will be larger when the camera is high.

The constant k is obtained with experimentation. For example using an 800 by 600 resolution the value that shows better visual results is $k_0 = 0.00136$. Using this result it is possible to calculate all k values for all screen resolutions because the relation between the constants and the resolution is inversely proportional. The larger is the resolution, the smallest amount of movement is necessary per pixel. The final k equation implemented has the following constants: $k_0 = 0.00136$, $S_0 = 800$, S is the current screen resolution (the same dimension is considered in S_0 and S).

$$\frac{k}{k_0} = \frac{S_0}{S} \Leftrightarrow k = \frac{S_0 \cdot k_0}{S} \qquad \textbf{(3.2)}$$

The following diagram represents the camera translation in pixel and world units:

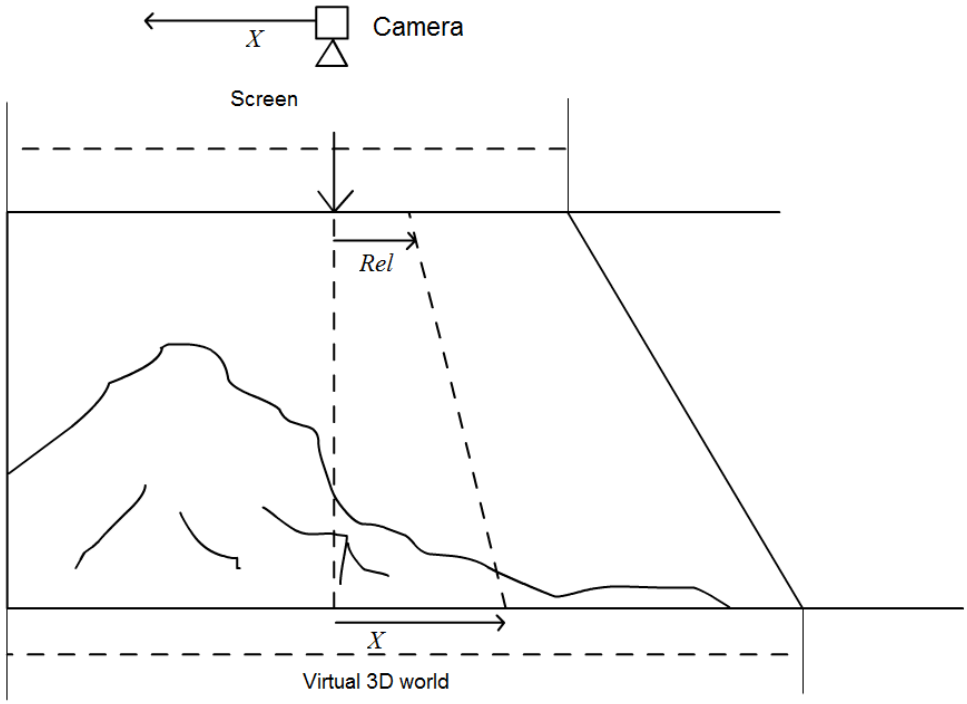

Fig. 3.11– Camera translation.

2) The rotate operation is triggered when the user rotates the interface knob on the bottom right corner of the screen. The mouse position is converted into a vector with origin in the center of the knob. It is then compared with the previous vector of the last move. The angle that the two vectors make is considered the angle of rotation.

3) The zoom is a simple function where the camera is translated up and down.

4) Finally the pitch operation is used to change the pointing direction of the camera. Initially the camera makes a 90º angle with the terrain. With the pitch operation it is possible to reduce this angle. This is made with a simple scene graph node rotate transformation.

Additional details about the navigation interface can be seen in chapter 4.

3.4.9 Interaction devices

To consider the different interaction options, it was necessary to study a wide variety of input devices. The prototype implementation had to take that into account. There are basically three types of input devices that were supported in the application:

- Keyboard: mainly for debug and shortcut keys
- Mouse related devices: mouse, interactive white boards, tablets
- Tangible interface devices: AR toolkit, multi-touch device.

All these devices and technologies will be detailed in chapter 5. The important point here is that the application through its InputManager generates three types of events:

- Keyboard events: key pressed/released.
- Mouse events: mouse pressed/released/moved.
- Multi-touch events: object id in/moved/out.

All these events are generated simultaneously, so it is possible although not practical to use the mouse, the interactive white board and the multi-touch device at the same time.

3.4.10 Custom 3D line creation

Many of the elements in the terrain are better represented with lines. Objects like roads or every line that is glued to the terrain are difficult to represent. One approach used in many RTS games is to stamp the road elements on the texture of the terrain. Unfortunately due to the internal structure of Ogre and the terrain plug-in PLSM it was not possible to do this dynamically. This was important because the user can load ShapeFile elements at any time and it may be necessary to represent them with lines.

The alternative solution was to create custom lines from 3D polygons. Ogre3D supports the dynamic construction of custom polygons. The main problem can be stated as: given a set of points and a certain thickness, how should the polygon that represents the line be constructed?

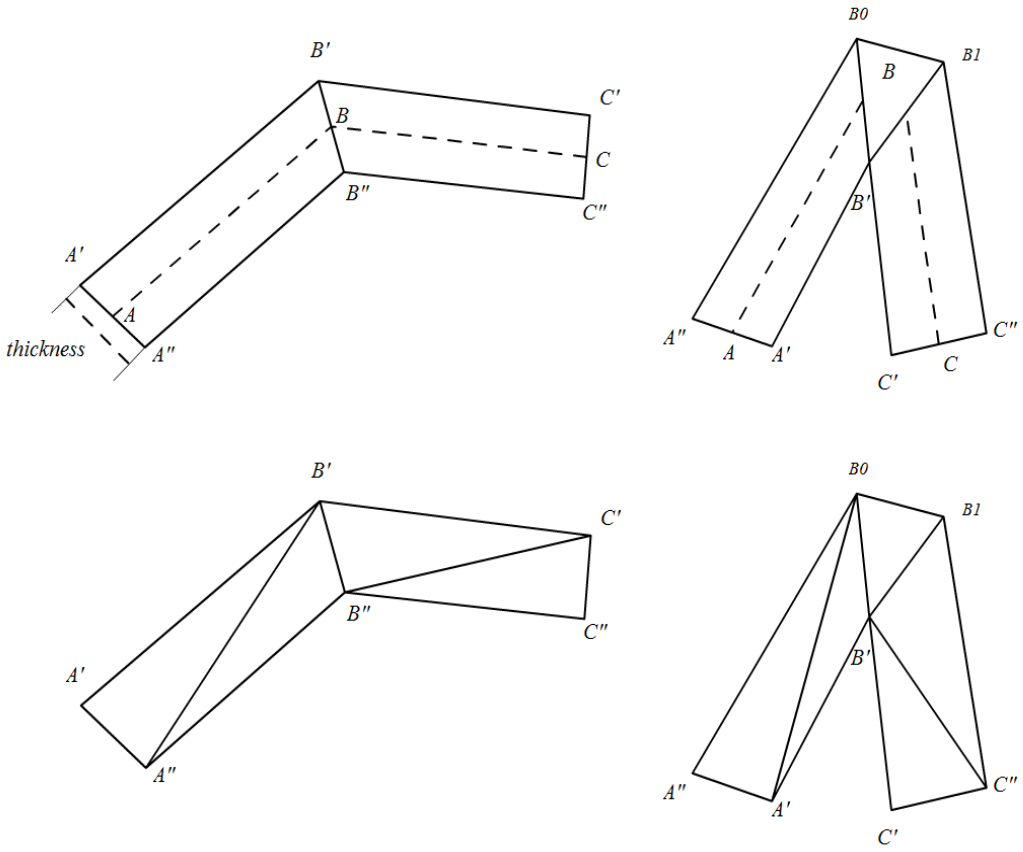

Fig. 3.12– Line creation with triangles

The main polygon is constructed triangle by triangle. Looking to the example on figure 3.12, there is one main problem that is to find if the angle formed by three points is less or more than 90º.

If the angle is equal or larger than 90º then only four triangles are required to generate the line. This is done by geometrically calculating the positions of *B'* and *B"*. The segment s is parallel to $\overline{AB}$ with a distance of half thickness. *B'* is the intersection of *s* with *t*.

If the angle is less than 90º then the above method would produce very sharpen unrealistic corners. If the angle is too short *B'* has a tendency to infinity because *s* and *t* take too long to intersect. To solve this situation another method was used. In this new method a new triangle is used to cut the corner. For this method it is necessary to find 3 points *B"*, B_0 and B_1 that are at a distance of half thickness from B. The implemented prototype generalizes this solution treating all corner points with the previous and next points.

To glue the lines into the terrain, the lines were interpolated into smaller divisions, and on each point the height of the line was the height of the terrain. The result was a smooth line that followed the terrain's relief as seen in image 3.13.

Sometimes there were small anomalies caused by abrupt terrain obstacles. These flaws

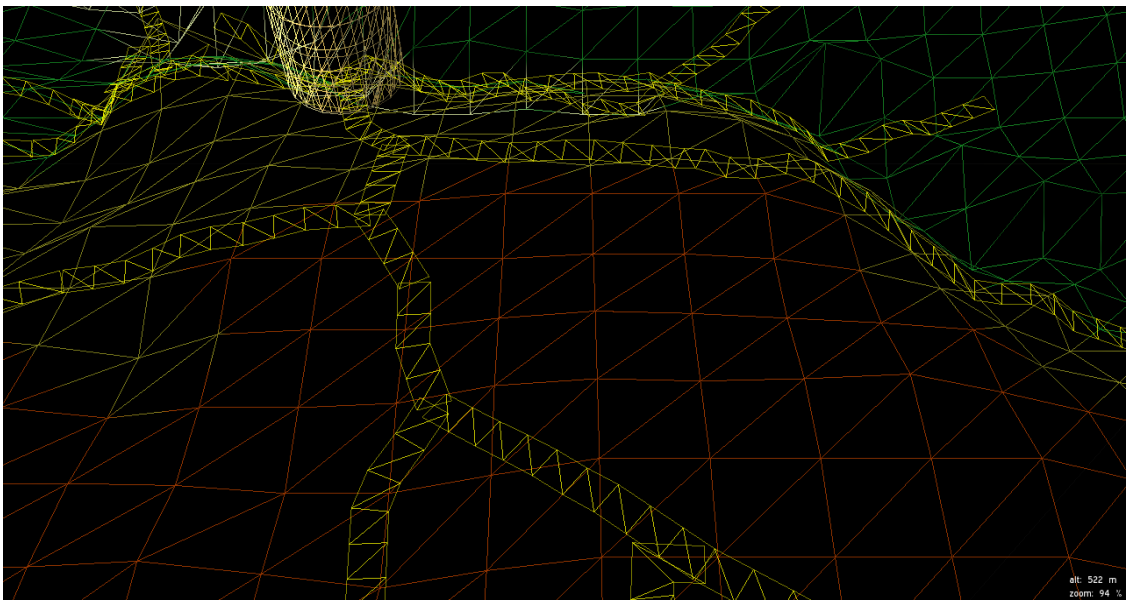

Fig. 3.13– Wireframe of the lines that follow the terrain (yellow lines)

were disguised by fooling the depth buffer used to paint the scene. Additional priority was given to the material of the lines, so that in small distances Ogre would always paint the lines above the terrain.

3.4.11 GUI and overlay manipulation

The interface of the prototype application was created with a mix of window menus and 2D overlays. Overlays are simple 2D graphics that are printed directly on the screen in front of the 3D scene. Ogre3D overlays are built using a configuration script that states the dimensions, material or text of the elements. They are very good for displaying images and simple text. They are essentially used for the mini-map and for the control console.

Overlays are not suited for complex window controls. For that it was used a proper GUI library called CEGUI. This is a library that can be integrated with Ogre3D to manage windows and widgets above the 3D Scenario. These widgets are not ordinary OS windows as they are rendered inside the application. The main advantage of that is the full customization that it allows. It supports all the widgets that are expected in a GUI such as buttons, menus, dropdown lists or checkboxes. The widgets are organized in layouts. Each layout has an XML representation where all windows and widgets are defined with their default properties. The implemented logic was programmed in the code using CEGUI events. For each XML layout there is a C++ class to control it.

4. Visual Interface

One of the main objectives of this thesis was to propose a new interface, especially one that takes advantage of new forms of interaction. To create the interface the following design steps [2] were followed:

- Study the user requirements of the system.
- Build a prototype of the interface.
- Evaluate the prototype.
- Build the final interface.

Currently the program loads a three dimensional terrain. It loads geographic information from GIS data, namely ShapeFile data from ArcGis. It includes a navigation system, with sliders and a mini-map. The user can turn on and off several layers of information. The interface allows the selection of agents and the definition of actions for them. Currently it supports and presents a simulation of a flooding wave (fig. 4.3 and 4.9) and of the emergency agents. This chapter presents the necessary steps to create the interface and its main features.

4.1 Interface study

The human-computer interaction study started with several interviews conducted to collect all the necessary requirements of the Life-Saver project. These interviews were essentially held in LNEC with the Life-Saver project partners. These resulted in the requirements for the main

components of Life-Saver. Additional interviews were made specifically related with the Visualization component, which is the object of this thesis. According to these interviews the main requirements for the Visualization were:

- A better visual GIS than the traditional ArcGIS.
- An interactive multimedia system.
- A tridimensional rendering of the simulation system
- An intuitive look that would appeal to future users, many of them with no computer skills.

As usual, when defining interfaces, the requirements were quite vague and required several iterations of analysis, design, prototype and validation to reach the final version.

4.1.1 Preliminary interface

After the technology was mastered, a preliminary interface was designed. The main objective of this interface was to build an infrastructure where the system could be temporarily tested and debugged. Other goal was to start testing new ideas with real users.

The first interface had the 3D terrain as background; the navigation controls were above it and occupied the entire screen. Additional functionalities were triggered by keyboard keys, and a large amount of information was written to a console. The system toolbars provided technical details such as: the number of frames per second, triangle count and minimum and maximum time elapsed by frame.

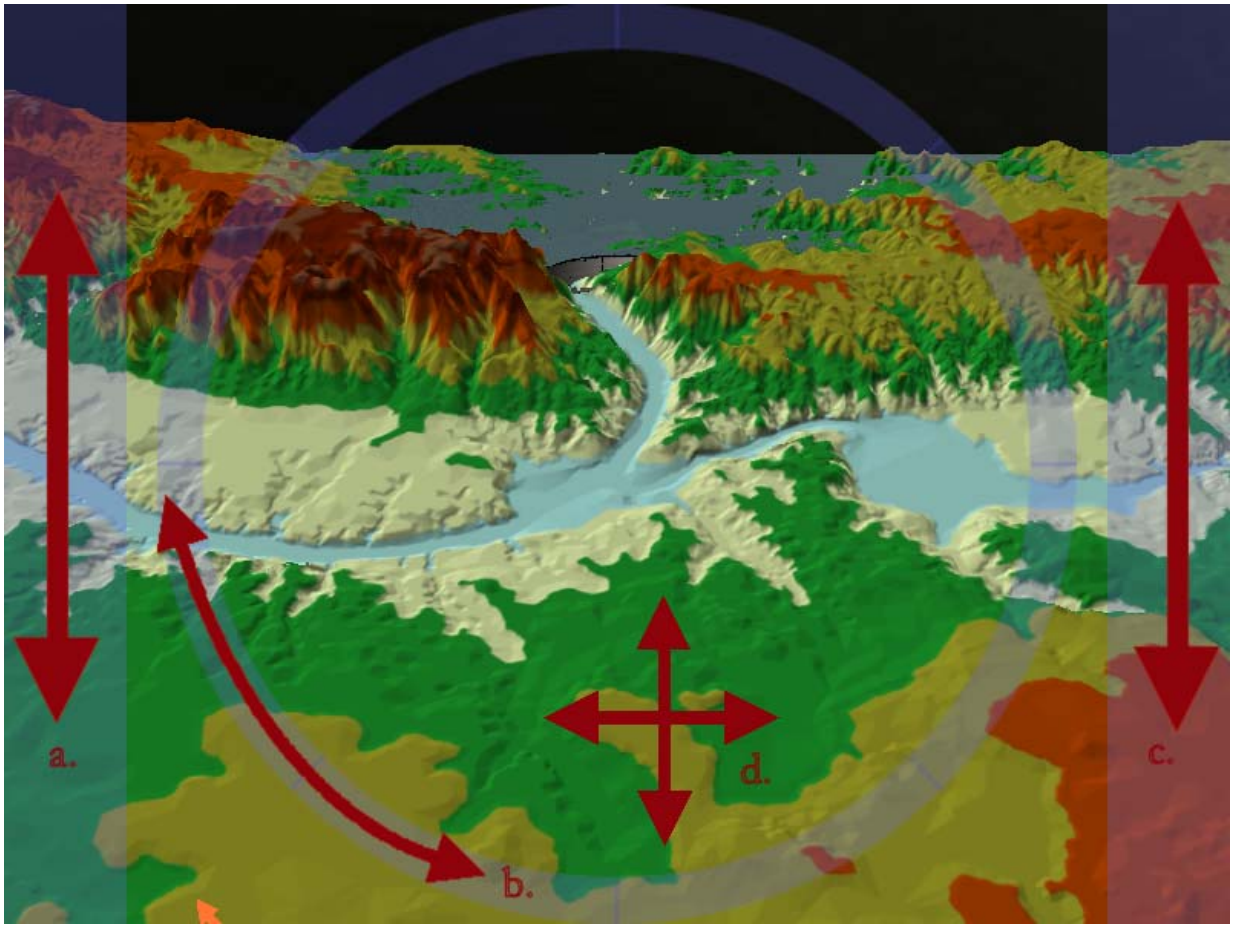

Fig. 4.1– Interface of the Visualization module. Touch screen movements: a) zoom b)rotate c) pitch d) move.

The main concept behind this first approach was to provide large areas were the user could easily click. At this time the main input interface that was being tested was the touchable interactive white board. The users controlled the interface with their fingers. Large draggable areas were used to help manipulating the interface without having to worry if the users were touching in the exact place that they wanted.

The application had two large bars on the sides and one large wheel on the center (Fig. 4.1). The left bar was used for zooming. Dragging up would zoom up, dragging down would zoom down. In the same way, the right bar tilted the world, dragging down reduced the angle between the camera orientation and the horizontal terrain. In the other direction, it increased the angle until it reaches 90 degrees. The wheel was used to rotate the world. The degree of rotation was related to the cursor rotation around the center of the wheel. To move the world

the user had to touch any other area and drag. Initially the bars on the left and on the right were smaller and didn't stretch till the top and bottom of the screen. This was changed following a suggestion of some users because sometimes it was easier to zoom if the user was not worried with the vertical end of the bar. Another suggestion that was implemented was to put stripes on the bars. They helped to create a visual feeling of movement and precision.

To validate this initial interface some informal tests were held with five users. The tests were done with an IWB (Interactive White Board). The tests pointed several positive and negative aspects. The positive aspects were:

- Good visual representation of the terrain.
- Easy access to the navigation bars. Easy to touch even if the touch screen is not so well calibrated (sometimes there were problems with correspondence between the finger and the cursor).
- Good navigation, the users could easily go to some place if told to.

The main negative aspects were:

- The interface occupied most of the screen.
- The rotate wheel was too large, and its use was tiring especially in the upper regions of the board.
- In some users there was some confusion about the place where they touched, expecting to do something and in reality another action happened. This was especially true between the move function and the others.
- No identification or hint about what each bar would do.

This interface was still used during part of the development of the Visualization prototype, but was later replaced with a more complete interface.

4.1.2 Final interface

To study the application's final interface it was necessary to learn from the mistakes of the preliminary approach. First of all it was decided to make a user test of what was expected from an application with these requirements. Several informal inquires were made to five people. To each was given this Visualization problem. They had to interpret it and try to draw an interface for the application. The majority of the test participants had a computer science background and had good user level experience with computers.

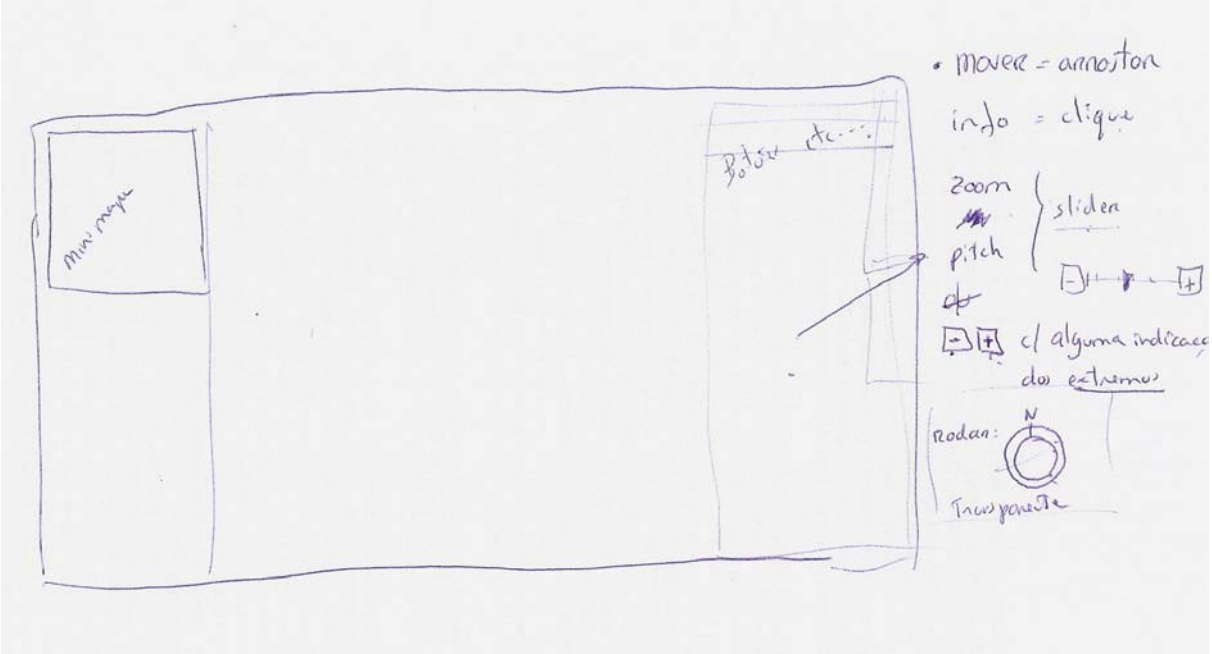

Fig. 4.2– Prototype drawing created by a volunteer participant

The main noticeable influence in the drawings (fig. 4.2) were computer games and Google Earth [11]. Although there were some original differences, the main aspects were consensual:

- The most common interface elements should be in the lower side of the screen because it is easier to use in a touch screen.
- The right side should be privileged because most people are right handed. If the menu is on the left, some elements are blocked by the user's arm. In the future this could be changed with a configuration menu.

- The navigation menu could be smaller.
- Optional menus should be hidden

Borrowing ideas from the drawings, the final interface was created. Figure 4.3, shows the final result. The detailed functionalities and features will be presented in the next chapter.

The final interface was validated through meetings with LNEC partners and EDIA.

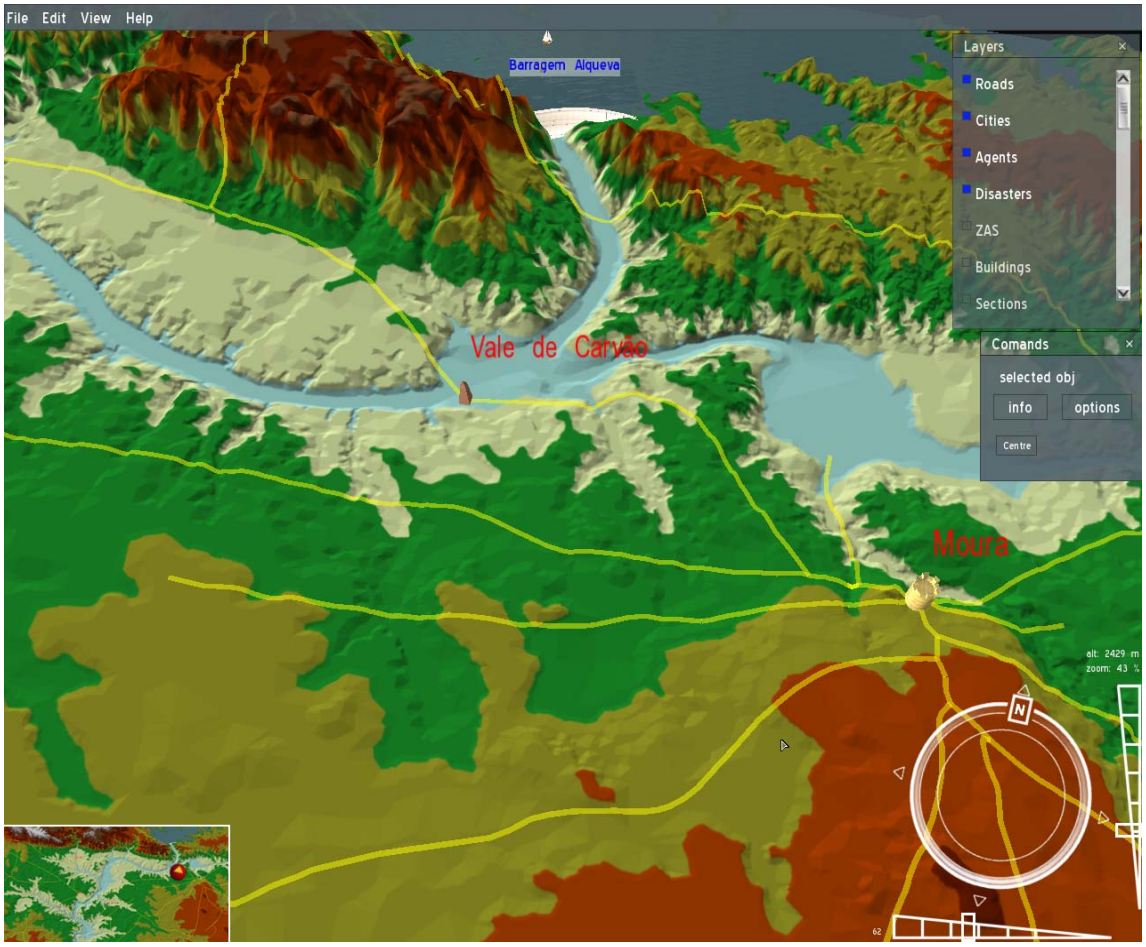

Fig. 4.3– Final prototype interface

4.2 Features

As mentioned the Life-Saver Visualization application features several components, most of them inspired by RTS games or by geographic information systems. The current application interface is essentially centered on the world navigation and on the simulation agent interaction options. Since it is a project related with emergencies planning and simulation the prototype application is designed to be used in a control center where many people interact

collaboratively. Because of that we elected as our primarily input device the Interactive White Board (IWB) and the interface reflects that. All menus can be accessed using only the left click (equivalent to the screen touch).

4.2.1 Navigation and mini-map

The navigation (fig. 4.4) is made using the bottom right interface. The knob rotates the world indicating the North, the vertical slider changes the zoom and the horizontal slider changes the pitch of the viewer enabling a 3D perspective. To move the world the user only has to drag the terrain. This interface has similarities with the Google Earth configuration.

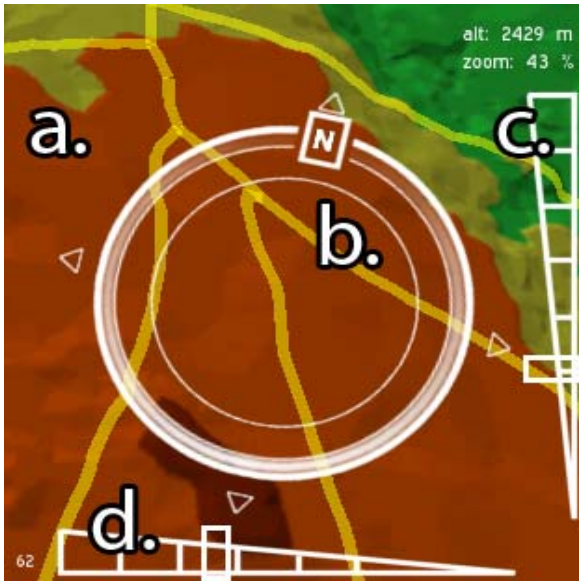

Fig. 4.4– Navigation console: a) move dragging the terrain, b) rotate, c) zoom and d) pitch

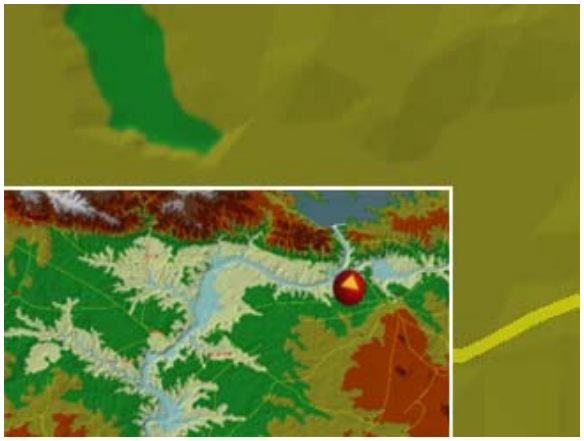

Fig. 4.5– Mini-map

The left mini-map (fig. 4.5) allows a fast recognition of where the user is. If the user clicks on it the viewing area it is redirected to that area. This is a similar behavior to the RTS games.

4.2.2 Roads and cities

The roads are an accurate representation of the information retrieved from ArcGIS ShapeFiles. Car agents usually follow these lines when they are rescuing people in the simulation(fig.4.6 and 4.9).

The roads are organized as a graph with arcs and nodes. Each arc has a certain amount of information associated with it. This information includes the official designation of the road, the state of conservation, the kilometers and the scale in which they were drawn. The cities are represented in the terrain with towers and labels. The towers are different accordingly to the importance of the settlement.

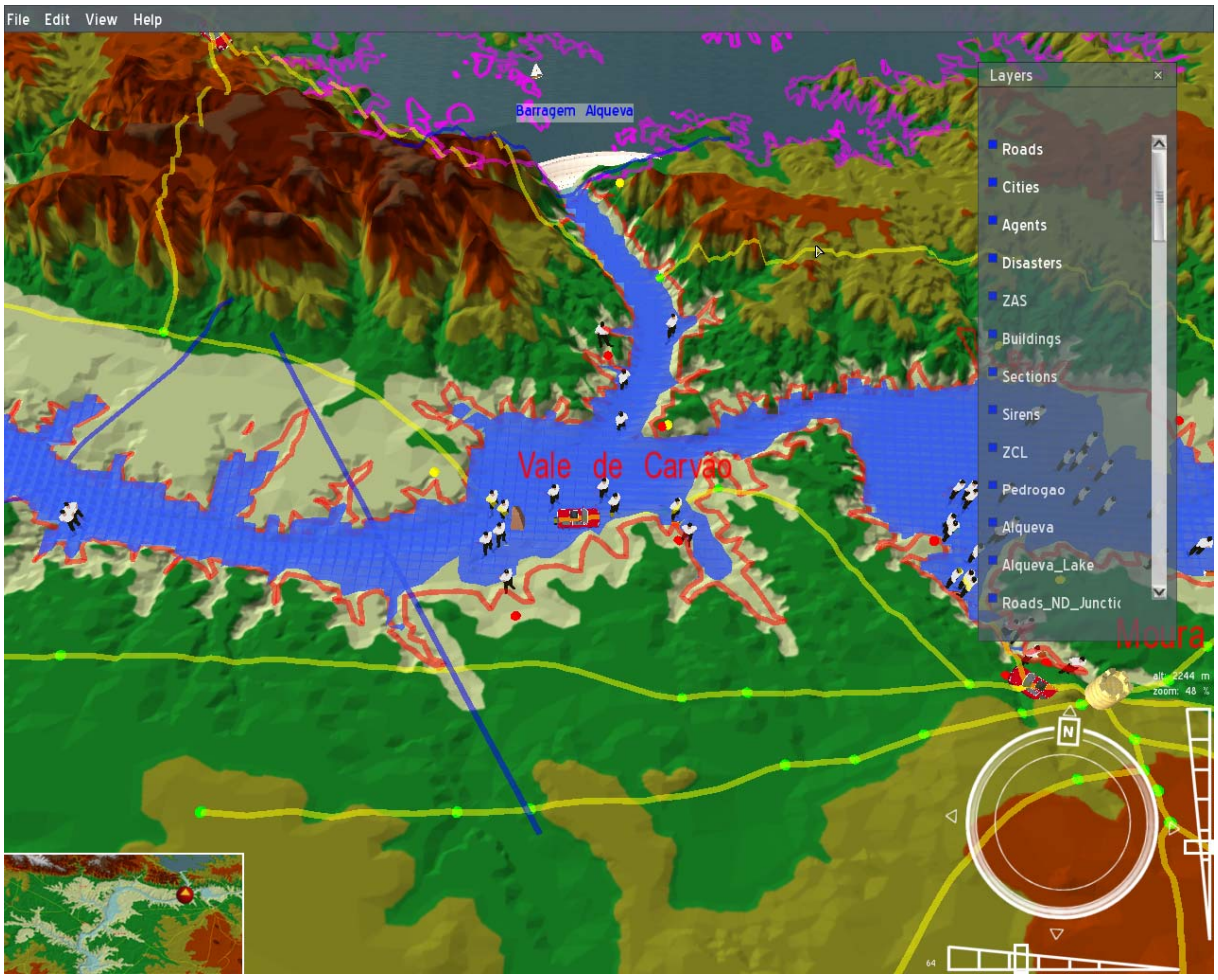

Fig. 4.6– Layer representation. The right menu allows toggling the layers visibility.

4.2.3 Layers

One of the features is the capability to load information layers from external systems, including shapefiles from ArcGIS [4]. This facilitates loading information like building locations, flooding areas, road blocks, bridges and all the map information that can be represented by points, lines or areas. The user is able to switch on and off each layer, using the layers menu as seen in figure 4.6.

4.2.4 Menus

The main menu can be seen on the top of the screen (fig. 4.7). It has four main options:

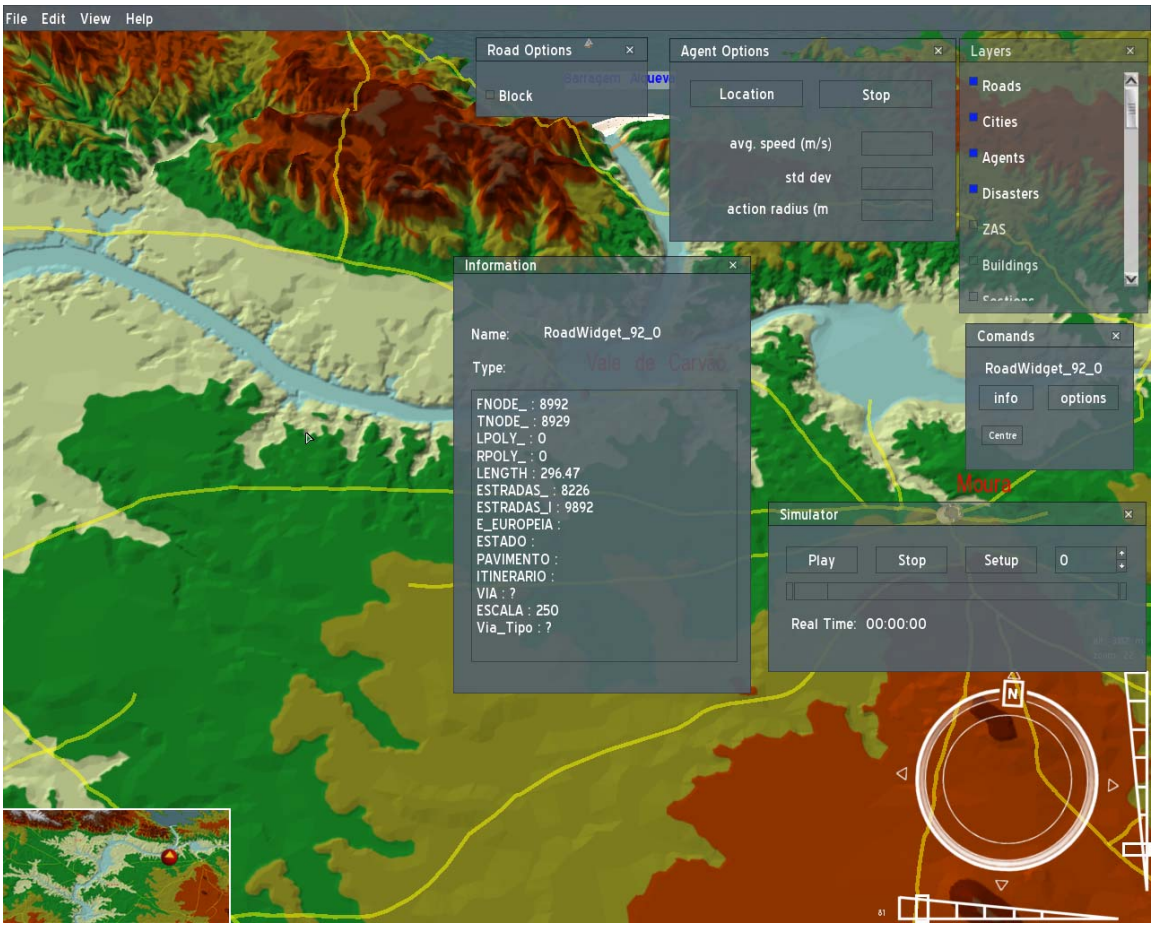

Fig. 4.7– GUI system of the prototype.

- File: It only has the exit button.
- Edit: The user can edit the preferences of the application and of the simulation.
- View: The user can show and hide all control windows.
- Help: The user can obtain information about the authors.

4.2.5 Object information

With the command window (fig.4.8) on, it is possible to click on world objects and select them. Selecting one object shows relevant information in the information window. It also shows all the enabled actions for the object. Information, like the identification of the object, is shown directly on the command window.

Fig. 4.8– Object selection. The Information window on the right displays details about the selected object.

4.2.6 Simulation control

The simulation control is only partially implemented. There are plans in the Life-Saver project to fully control the simulation through the Visualization component but that feature is not yet implemented. What is implemented is a recording system that saves the state of all the simulation. While the simulation is running or when it finishes it is possible to move the

horizontal slider in order to move the simulation back and forward (fig.4.9). This allows going back to any point in the simulation.

Fig. 4.9–Simulation control a) horizontal time slider and b) current simulation tick.

5. Interaction

There are several devices that depart from the traditional mouse and keyboard. Some were tested in order to be used in the prototype Visualization application. What was considered about each device is its availability, degree of supported interaction and amount of people that can interact at a given time with the application. Currently, the mouse, digitizer pen and the touch-sensitive board are supported. Additionally, several studies were made to use collaborative multipoint interfaces.

5.1 Interaction devices

The Interactive Whiteboard (IWB) is a large white interactive board where the computer image is projected and that enables navigation on the screen, by touching it. It only supports one touch at any given time. Whenever the board is touched, it assumes that the touch position is where the mouse cursor is. The idea is to use the IWB as the central piece of the control room, placing it on a wall or on a table (fig.5.1).

The IWB is a very practical hands-on device but has some limitations. It only handles absolute position, when the user touches the board, and only handles relative position when dragging on the board. Mainly, it only supports left clicking. A right click button is available, but it is not intuitive and easy to reach. Another handicap is that users usually interact with

their fingers that are less precise than a pen or a mouse. There are also some synchronization problems between the position touched by the user and the cursor's position.

In this project, the IWB is used with large buttons and draggable areas, where the user can do all the world moving operations with his hands. The same interface that works with the IWB is naturally usable in a Tablet PC or with a digitizer pen. For demonstration purposes the Tablet is used due to its portability.The keyboard and the mouse are also supported but mainly for debug and demonstration when the other devices are not available.

Fig. 5.1– Visualization simulation component with an Interactive White Board interface.

5.2 Multi-touch research

Multipoint interaction was one of the topics that were studied in this thesis. Special focus was put in techniques that people can use to enhance their work by using multipoint and multi-touch technologies.

The main disadvantage from the IWB is that it only allows a single touch point. This makes the interaction rather limited. Using a multi-touch interface will lead to more invisible interfaces. All the operations are handled with finger gestures. It is possible to rotate and zoom with two fingers, by dragging them on the board [15].

To create a multipoint interface several solutions were studied [18,21,27], almost all requiring computer vision techniques as described next.

5.2.1 AR toolkit approach

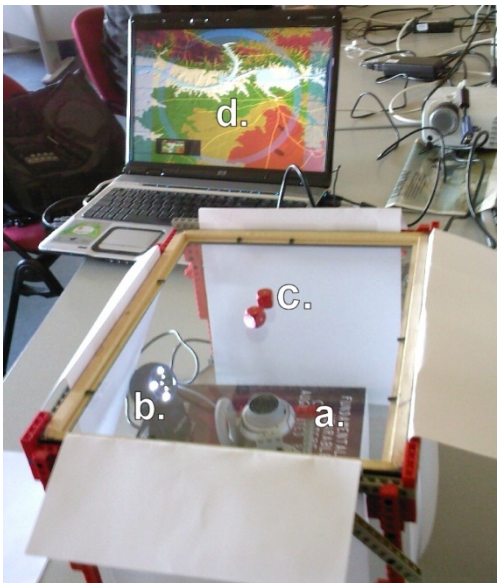

Fig. 5.2– Multipoint research device: a) USB web camera, b) illumination, c) two markers that sit on top of the glass, d) computer where the image is processed and the position of the marks is detected.

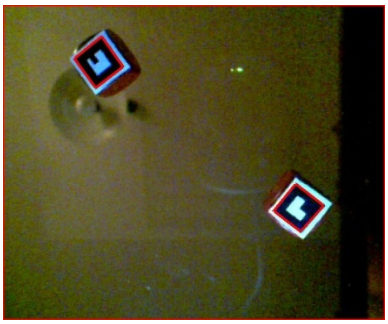

Fig. 5.3– Camera view showing the AR toolkit markers. The algorithm detects the position of the black squares.

The first approach to a multipoint device was a table with a glass top (fig.5.2). Beneath the glass there is a common USB camera aimed right up. The glass is covered with a white polyethylene plastic that makes objects far away diffuse and is see-through with near objects. On the top of the glass are placed multiple objects that have special markers on the bottom. The goal is to capture the markers with the camera and use detection algorithms to determine their positions on the table.

The users interact with the computer by moving the different marked objects (fig.5.3). Each object is represented on the screen with a different colored cursor. The markers and detection algorithm used are based on the Augmented Reality Toolkit (AR toolkit) [5].

Besides the marker position it is also possible to detect if the marker is touching the glass and the rotation of the marker. This allows creating “click” events and rotation events that work like knobs.

A prototype application was created to test the device. It had two colored cursors and each was controlled by an independent marker. The experiment worked, it was possible to move the two cursors independently, proving that a multi-touch interface was feasible although there were some problems. The main problems of this device result essentially from lightning conditions and the low frequency of the camera (15 fps). Other problem was the slow marker detection by the AR toolkit algorithm. This caused jumps in the detection of the marker position resulting in a deficient interaction experience. Other problems were detected related with the poor construction of the table, like vibration and the possibility of markers being out of sight of the camera. This research is at prototype level and was not ready for real user deployment but showed promising results.

5.2.2 TouchLib approach

Following the footsteps of the FTIR (Frustrated Total Internal Reflection) [15] technique there were several developments in finger and object detection while doing this project. A library called Touchlib[38] was released from the creators of the first FTIR tables. It included camera driver detection, image processing, protocols support, blob detection and application multi-touch events.

After the construction of the glass table for the AR toolkit approach it was decided to build a diffused illumination [9] table. This table detects the shadow created by the fingers when they touch the screen. The construction is the same as the one described in the last section: a table with a glass on top and a polyethylene sheet above to diffuse light (fig.5.4).

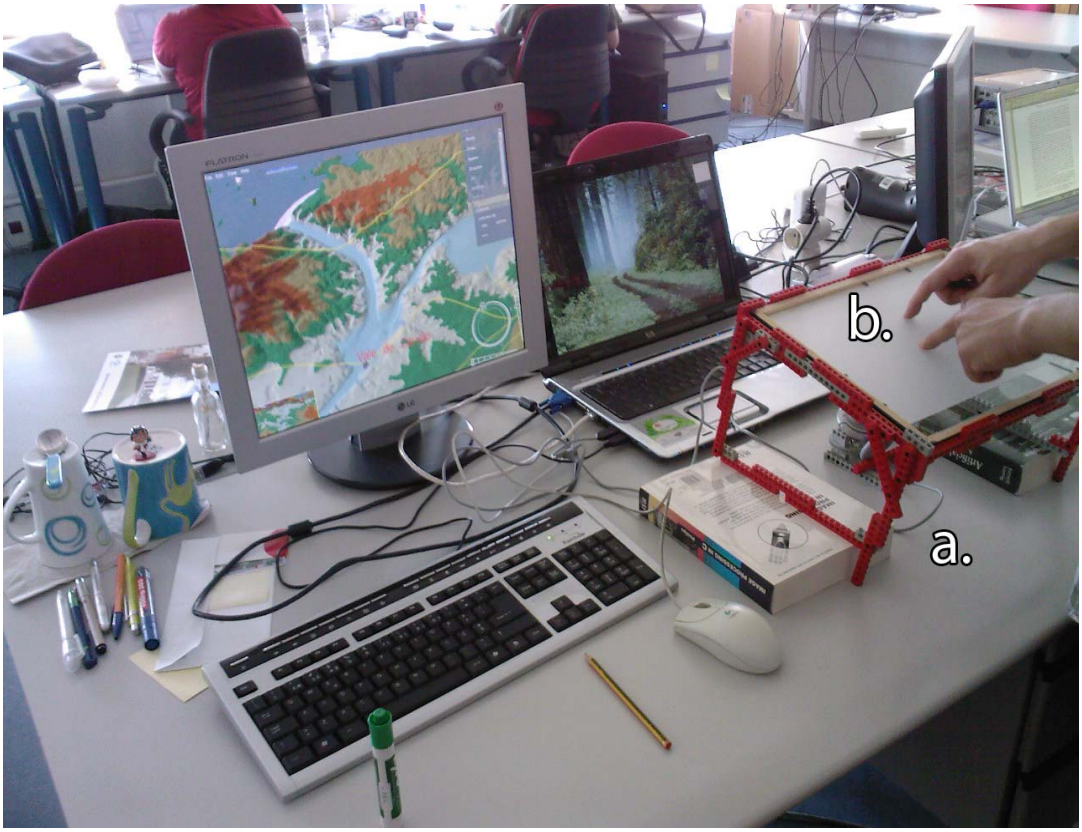

Fig. 5.4– Multi-touch input device. a) Camera, b) Touchable area.

The camera was upgraded to improve detection. The current camera operates at a resolution of 640 by 480 at 30 frames per second and has a wider field of view of 71 degrees. A wider field of view means that the table does not need to be so high and that the useful area of the table-top is larger than before.

Another change was the inclination of the table. Instead of being horizontal the table has a 20º inclination. This allowed a more ergonomic and natural interaction with the hands. It also resulted better in terms of illumination, mainly because the inclination reduces the chances of direct light from ceiling lamps.

To detect the position of the fingers MTmini [27] was used a slightly modified version of TouchLib. This toolkit has a configuration tool where several filters can be added to the image. From figure 5.5 it is possible to see the original camera image on the left top corner. The filters that were applied to the image were in this order:

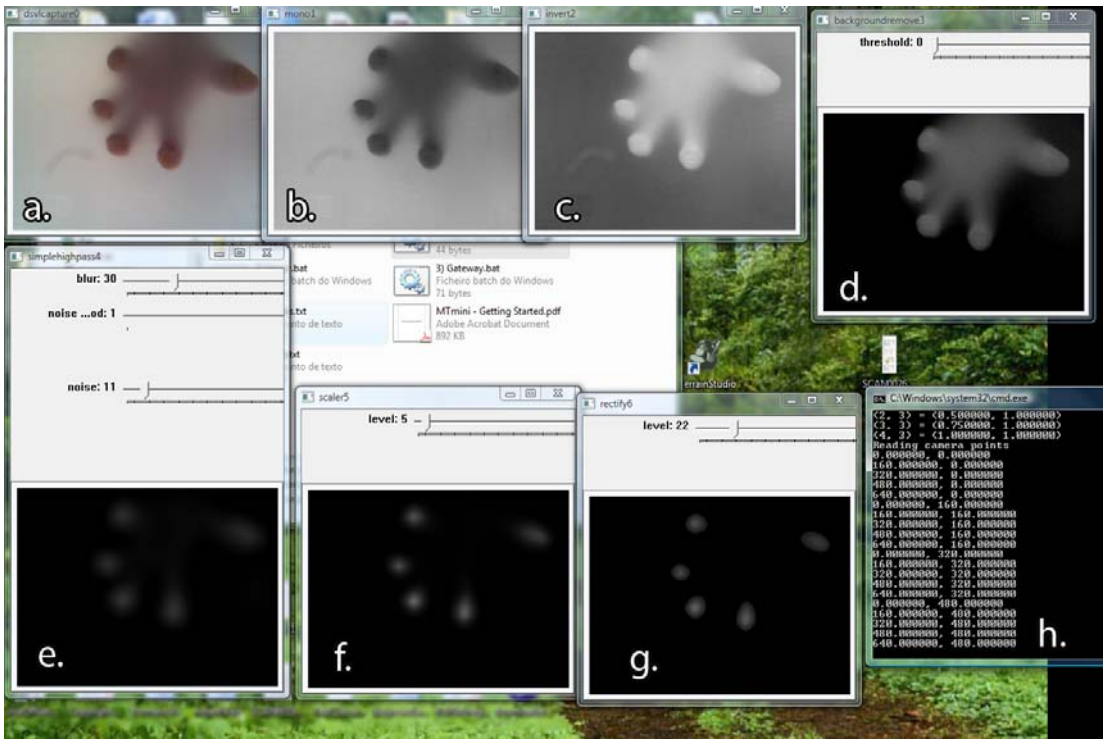

Fig. 5.5– a) Camera capture, b) monochromatic, c) invert, d) background removal, e) blur, f)scaler, g) rectify and final image and h) coordinates output.

- Monochromatic filter to transform the image to grayscale
- Inverse filter to invert the colors
- Background removal filter. Removes the background taking into account the first frames captured.
- Blur filter to remove noise and excess of information.
- Scaler filter to increase the intensity of the image.
- Rectify filter that works like a cutoff that increases the definition of the signal.

The final parameters were defined in a XML configuration; this configuration can be used with any TouchLib application. Here is the relevant part of the file:

```
<filtergraph>
  <dsvlcapture label="dsvlcapture0" />
  <mono label="mono1" />
  <invert label="invert2" />
  <backgroundremove label="backgroundremove3">
    <threshold value="0" />
  </backgroundremove>
  <simplehighpass label="simplehighpass4">
    <blur value="54" />
    <noise value="11" />
    <noiseMethod value="1" />
  </simplehighpass>
  <scaler label="scaler5">
    <level value="26" />
  </scaler>
  <rectify label="rectify6">
    <level value="250" />
  </rectify>
</filtergraph>
```

After all filters are applied the configuration is complete. The TouchLib has a Server that processes the frames and runs a blob detection algorithm on them. There are two types of detection: (1) simple blob detection and (2) fiducial object detection. The resulting output is the position of each blob or object. The server then communicates with the client applications through TCP/IP sockets.

The communication between the server and the clients is done using the TUIO protocol [23]. This is a protocol created to unify the behavior of the multi-touch applications. It was based on the behavior of the Reactable [24] and the tDesk [36]. Before TUIO the Reactable project used the TUI protocol made by the Tangible MIT media group [35]. This new protocol extends it by combining multiple cursors and multiple objects. The main points of the protocol are:

- All finger gestures are considered cursors and have a cursor id.
- All objects have a cursor id.
- For each change in a cursor or object there is a *Set* message with an update on the position, acceleration, rotation and movement vector.

For further details, the protocol is well detailed in [23]. To test the multi-touch device it was necessary to create a client application that implemented the TUIO protocol. The Visualization application built for the Life-Saver project was then altered to use this device.

The client for the Visualization component was based on the ReactVision[31] example client. It implements the TUIO protocol and generates the following events inside the application:

```
void addTuioObj(unsigned int s_id, unsigned int f_id)=0;
void updateTuioObj(unsigned int s_id, unsigned int f_id,
                   float xpos, float ypos, float angle, float x_speed,
                   float y_speed, float r_speed, float m_accel,
                   float r_accel)=0;
void removeTuioObj(unsigned int s_id, unsigned int f_id)=0;

void addTuioCur(unsigned int s_id)=0;
void updateTuioCur(unsigned int s_id, float xpos, float ypos,
                         float x_speed, float y_speed, float m_accel)=0;
void removeTuioCur(unsigned int s_id)=0;

void refresh()=0;
```

These events are managed by the InputManager class just like the keyboard and mouse events. With them it is possible to create several new gesture combinations in order to control the interface.

5.2.3 Software behaviour

For the demonstration of the multi-touch capabilities the Visualization prototype supports three basic touch gestures. These operations used for navigation purposes are movement, rotation and scaling.

For the move operation it is necessary to calculate the point of application and the direction of the movement vector. The user just touches the screen with one or more fingers and moves the terrain. For that it is necessary to analyze two frames. For each frame it is necessary to calculate the geometric center of all the points detected. For each point *i* the final center coordinates C are:

$$C_f = (X, Y) \in \mathbb{R}^2 \qquad \textbf{(5.1)}$$

$$X = \frac{\sum_{i=0}^{i<n} x_i}{n} \quad \textbf{(5.2)}$$

$$Y = \frac{\sum_{i=0}^{i<n} y_i}{n} \quad \textbf{(5.3)}$$

The movement will be applied to point C_f with a direction vector of:

$$\overrightarrow{C_{f+1} - C_f} \quad \textbf{(5.4)}$$

The scale movement is created by touching with two or more fingers on the screen. The result is a real number which represents the percentage of scale that was changed. This is calculated using the average sum of the distances between the points *i* and the center *C* of that frame:

$$D_f \in \mathbb{R} \quad \textbf{(5.5)}$$

$$D_f = \frac{\sum_{i=0}^{i<n} \sqrt{(x_i - X)^2 + (y_i - Y)^2}}{n} \quad \textbf{(5.6)}$$

The percentage of scale that will be applied to the Visualization scenario in frame *f* will be P_f, defined as:

$$P_0 = 1 \quad \textbf{(5.7)}$$

$$P_{f+1} = \frac{D_{f+1}}{D_f} \quad \textbf{(5.8)}$$

The last operation is the rotate operation. To perform this operation the user must use two fingers and rotate one of them around the other. The Center of the rotation will be C_f and α is the angle of the rotation. Let V be the vector created by the two points:

$$V_f = (x_1 - x_0, y_1 - y_0) \quad \textbf{(5.9)}$$

$$V_{f+1} \cdot V_f = |V_{f+1}||V_f| \cos\theta \Leftrightarrow \theta = arccos\left(\frac{V_{f+1} \cdot V_f}{|V_{f+1}||V_f|}\right) \quad \textbf{(5.10)}$$

The only parameter that has to be decided after calculating the angle θ is the direction of the rotation. To do that the cross product *K* is used:

$$k = V_{f+1} \times V_f \quad \textbf{(5.11)}$$

$$\alpha = \begin{cases} 2\pi - \theta, & k < 0 \\ \theta, & k \geq 0 \end{cases} \quad \textbf{(5.12)}$$

In the end, the terrain scenario is rotated α radians. All these features were implemented in the Visualization prototype and are currently working with the emergency scenario.

5.2.4 Other device approachs

Other experiments were made at the interaction device level. Several tests were made with infrared light that is invisible to the human eye. Fortunately cameras, even cheap ones can detect visible and infrared light. For example, to check if a TV remote LED is working, it is only necessary to point it to a webcam and see if a white light appears on the screen.

The multi-touch table described in the last chapter uses the visible spectrum of light to work. This presents some difficulties because it is not possible to project a screen above the table. That would interfere with the process of finger detection. Another problem is the excess of visible light during the day.

The solution to this problem would be as stated in [15] to use infrared light for finger detection and visible light for display. This concept was prototyped with a circuit made of four infra-red LED that were connected to a switcher and a power source (fig.5.6). The LED were then positioned in several points in the table. The infrared filter of the camera used for movement detection was removed, and a visible light filter was added to it.

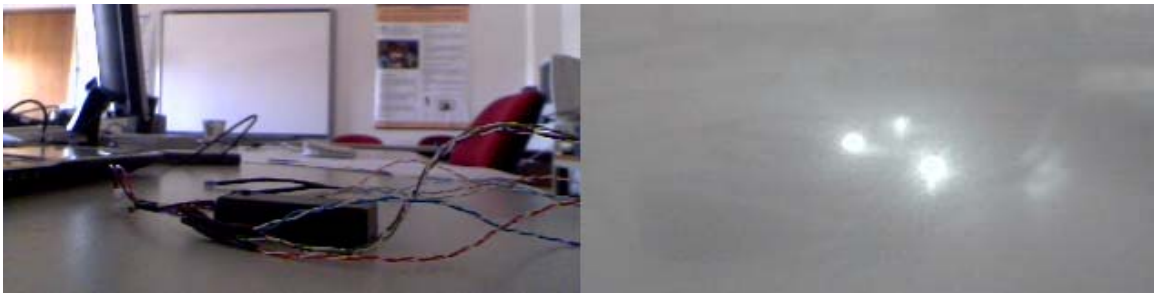

Fig. 5.6– LED circuit on daylight and the same image captured with an infra-red camera

Several experiments were conducted with the LED in different positions, but the results were not very satisfactory. The main causes were the small amount of light produced by the four LED, the fact that the light source was too concentrated and the glass reflected the infrared light.

One possible future application for the LED is to use them as light pens. Instead of detecting the reflection of the infra-red light in the fingers, it would be detected the light emitted by an object.

5.3 Comparing interaction devices

The following table compares, in several situations, the tested input devices:

	Keyboard/ Mouse	IWB/Tablets	AR toolkit table	Multi-Touch table
Supports movement, left and right clicking	All	No movement only direct clicking	No movement only direct clicking	No movement only direct clicking
Multi-point	No	No	Yes	Yes
Navigation with gestures	No	No	Yes	Yes
Finger interaction	No	Yes	No, interaction with objects	Yes
Precision	Pixel level	2 cm	Very imprecise*	Very imprecise*
Availability	Commercial, cheap	Commercial, expensive	Lab-made, cheap	Lab-made, cheap
Touch/Display relation	Indirect	Direct	Indirect	Indirect
Visualization Prototype support	Yes	Yes	No	Yes
Shortcut keys	Yes	No	No	No

* This is related to the fact that the user has no idea where it is pressing. This can be resolved with a screen projection on the table where the user is touching.

The comparison table above shows that the mouse and keyboard setup continues to be the most precise combination of devices. It also supports all the interface functions available. The IWB is also very complete allowing a full interactive experience. The lack of movement (as in cursor movement) is compensated with the direct relation between touch and display, although the developer has to take this into account on the software implementation. The AR toolkit and Multi-touch tables are still in prototype level. They were successful as a prove of concept but are not ready for real user deployment. The main advantage of the multi-point approaches is that the interface becomes invisible and the interaction is made through combinations of gestures.

6. Conclusions

This dissertation presents a system for emergency simulation, focusing on the user interface and design decisions for the Visualization component. Using a game engine to support the interface, enables a rich set of interaction possibilities, which can be augmented as more actions are needed. Supporting multiple input devices enables several metaphors that were experimented for navigating in the emergency area.

The main objectives were achieved. There is now a prototype application that illustrates all research that was carried out. A multi-touch device was created and connected to the prototype. Two generic libraries were created to support the system. Finally two papers (section 1.2.2) about this work were submitted and accepted and the work was shown to Life-Saver partners LNEC and EDIA.

6.1 Future work

The current work is integrated in project Life-Saver and the future work can advance it in several ways. The integration between the several components of the project can be enhanced. The simulation should be fully controllable through the Visualization component. Additional functions can be added, such as visual emergency plan elaboration or real-time simulation manipulation. From an interaction point of view, there are new commercial devices that

support multi-touch. It would be interesting to expand the touch gestures to create an interaction experience where the interface is almost invisible.

There are possibilities where further research would be necessary, although the main goal in the future should be to use the prototype in real settings to help save lives by making the emergency plans more accurate and more realistic.

Bibliography

1. A. Betâmio de Almeida, C. Matias Ramos, Maria A. Santos, Teresa Viseu, **Dam Break Flood Risk Management in Portugal**, Laboratório Nacional de Engenharia Civil, Lisbon, 2003

2. Alan Dix, Janet Finlay, Gregory Abowd, Russel Beale, **Human-Computer Interaction**, 2nd Edition, Prentice Hall, 1998

3. André Sabino, Rui Nóbrega, Nuno Correia, Armanda Rodrigues, **Life-Saver: Flood Emergency Simulator**, *Proceedings of ISCRAM 2008*, Washington D.C., 2008

4. ArcGis - http://www.esri.com/, 2007

5. ARToolkit projects - http://www.hitl.washington.edu/artoolkit/projects/, 2007

6. Bill Baxton's site - http://www.billbuxton.com/multitouchOverview.html, 2008

7. Bill Moggridge, **Designing Interactions**, The MIT Press, 2007

8. Blazej Kot , Burkhard Wuensche, John Grundy, John Hosking, **Information visualisation utilising 3D computer game engines case study: a source code comprehension tool**, *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*, 2005

9. Diffused-Illumination, http://www.whitenoiseaudio.com/blog/2007/06/diffused-illumination-vs-ftir.html, 2008

10. FMOD - http://www.fmod.org/, 2007

11. Google Earth - http://earth.google.com/, 2007

12. Google KML API - http://code.google.com/apis/kml/, 2007

13. Gregory Junker, **Pro OGRE3D Programming**, APress, 2006

14. Gsoap - http://www.cs.fsu.edu/~engelen/soap.html, 2008

15. Han, J. Y. **Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection**. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 2005

16. HazMat, http://www.etc.cmu.edu/projects/hazmat_2005/, 2007

17. Ingmar Rauschert, Pyush Agrawal, Rajeev Sharma, Sven Fuhrmann, Isaac Brewer, Alan MacEachren, **Designing a human-centered, multimodal GIS interface to support emergency management**, In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, (119-124), 2002

18. James Patten, Hiroshi Ishii, Jim Hines, Gian Pangaro, **Sensetable: a wireless object tracking platform for tangible user interfaces**, In *Proceedings of the SIGCHI conference on Human factors in computing systems*, (253-260), 2001

19. Jeff Han Ted Talks, http://www.ted.com/index.php/talks/view/id/65, 2008

20. Jojn F. Koegel Buford, **Multimedia Systems**, ACM Press, Addison-Wesley Publishing Company

21. Jordà, S., Geiger, G., Alonso, M., Kaltenbrunner, M. , **The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces** *Proceedings of First International Conference on Tangible and Embedded Interaction*, 2007

22. José H. Canós, Gustavo Alonso, Javier Jaén, **A Multimedia Approach to the Efficient Implementation and Use of Emergency Plans**, IEEE Computer Society, 2004

23. Kaltenbrunner, M., Bovermann, T., Bencina, R., Constanza, E., **TUIO: A protocol for table-top tangible user interfaces**, *Proc. of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005

24. Kaltenbrunner, M., Jordà, S., Geiger, G., Alonso, M., **The reacTable*: A Collaborative Musical Instrument**, *Proceedings of the Workshop on "Tangible Interaction in Collaborative Environments" (TICE), at the 15th International IEEE Workshops on Enabling Technologies (WETICE 2006).* Manchester, U.K.

25. Kim Pallister, **Game Programming Gems 5**, Charles River Media, 2005

26. Matthew Plumlee, Colin Ware, **An evaluation of methods for linking 3D views**, *Proceedings of the 2003 symposium on Interactive 3D graphics*, (193-201), 2003

27. MTmini, http://ssandler.wordpress.com/MTmini/, 2008

28. Ogre3D - http://www.ogre3d.org/, 2007

29. OpenCV - http://www.intel.com/technology/computing/opencv/, 2008

30. PLSM2 Ogre3D Plug-in, http://tuan.kuranes.free.fr/Ogre.html, 2007

31. ReactVision, http://reactable.iua.upf.edu/?software, 2008

32. Sanjay Jain, Charles McLean, **A Framework for Modeling and Simulation for Emergency Response**, *Proceedings of the 2003 Winter Simulation Conference* (1068 – 1076)

33. ShapeLib - http://shapelib.maptools.org/, 2007

34. Stop Disaster - http://www.stopdisastersgame.org/, 2007

35. Tangible Media Group - http://tangible.media.mit.edu/, 2008

36. TDesk, http://www.multigesture.net, 2008

37. Teresa Romão, Nuno Correia et al, **ANTS-Augmented Environments**, Computers & Graphics 28(5): 625-633 (2004)

38. TouchLib, http://www.whitenoiseaudio.com/touchlib/, http://www.nuigroup.com/touchlib/, 2008

39. Ullmer, B., Ishii, H. **Emerging Frameworks for Tangible User Interfaces**, In *J. M. Carroll, editor, Human-Computer Interaction in the New Millennium*, pp. 579-601, 2001

40. Virtools - http://www.virtools.com/, 2007

41. Woodrow Barfield, Thomas Caudell, **Fundamentals of Wearable Computers and Augmented Reality**, Lawrence Erlbaum Associates, Publishers, 2001