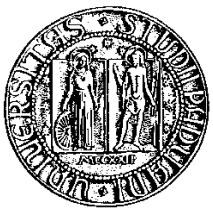

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingeneria Informatica

tesi di laurea

3D modeling and visualization of utility networks

Relatore: Ch.mo Prof. Massimo Rumor

Laureando: Francesco Dolcetto

24 ottobre 2011

Abstract

Most of the current 2D GIS representations are tied to the limits of the GIS software tools used and are, more properly, simplifications of the real aspects of the territory. The development of 3D GIS tools has brought to life by the belief that the description of the reality and the analysis of the phenomena that take place in it must be done in the proper dimensions. In this way, it is possible to solve spatial problems non addressable in 2D. Moreover 3D representation can be used to communicate territorial information to non experts in a immediate realistic way.

An important field of application of GIS software tools is the utility networks management. Utility networks, such as water distribution networks or sewage networks, can take advantage from GIS representation for data visualisation, query and management.

Existing software tools that support management and 3D visualisation of utility networks are powerful but very expansive.

The aim of the project is to design a general network model that will be used to model an utility network and to visualise it in 3D.

The general network model has been designed to be data source and platform independent. It can take data from OGC standard compliant data source like GeoDBMS and WFS Servers and can be enriched by a user defined set of attributes, making it suitable for every network analysis and management need.

On the top of the general network model has been designed a geometrical model that transforms the general network model in a 3D model which can be displayed using a WebGL graphic engine.

Using the general network model and the geometrical model has been designed and developed a WebGL viewer that visualises the network model over a DTM with a complete navigation system that allows a complete tour of the scene and data query and editing.

Contents

Chapter 1

Introduction to 3D GIS

1.1 Geographic Information Systems

A Geographic Information System (GIS) is an organised collection of computer hardware, software and geographic data designed to efficiently capture, store, update, manipulate analyse and display all forms of geographically referenced information.

In a more generic sense, a GIS is a tool that allows users to create interactive queries, analyse the spatial information, edit data, maps and present the result of all these operations.

Spatial features are stored in a coordinate system (latitude/longitude, state plane, UTM, etc.), which references a particular place on Earth. Descriptive attributes in a tabular form are associated with spatial features. For example, in a GIS oriented to water utility networks, features could be pipes and manholes. The fundamental components of spatial data in a GIS are points, lines (arcs), and polygons. Spatial data and associated attributes in the same coordinate system can then be layered together for mapping and analysis. GIS can be used for scientific investigations, resource management, and development planning.

GIS differs from CAD and other graphical computer applications in that all spatial data are geographically referenced to a map projection in a earth coordinate system. For the most part, spatial data can be re-projected from one coordinate system into another, thus data from various sources can be brought together into a common database and integrated using GIS software. Boundaries of spatial features should register or align properly when re-projected into the same coordinate system.

Another property of a GIS database is that it has topology, which defines the spatial relationship between features. When such relationship

exists, it is possible to perform analysis on spatial data, such as modelling the flow through connecting lines in a network, combining adjacent polygons that have similar characteristics, and overlaying geographic features.

1.2 GIS for utility networks

The main features of a typical GIS software tool oriented to utility networks are

- Visualization, query and search (navigate on the map, spatial search for elements).
- Editing and update of data (query and edit alphanumeric attributes of network elements).
- Creation of new network elements.

A desirable feature of such a software is the ability to run on multiple platforms and environments, in particular inside an Internet browser.

Web-based GIS (WebGIS) are, in fact, one of the most deployed approach. This is due to their easy upgrade/update mechanism and the absence of traditional software on the client, that reduce investments in hardware. Currently are available many OGC standards compliant geospatial servers (like GeoServer or MapServer) and GeoDBMS (like PostgreSQL-PostGIS) that allows the creation of scalable, reliable and, more important, standard compliant platforms on the top of which it is possible to design a WebGIS.

An utility network can be easily imagined as a graph. In a water distribution network, for example, the vertices could be water storage facilities, water pressurising components and water usage points and the edges could be the pipes connecting network components. In a 2D GIS this way of thinking about an utility network directly leads to a representation in which network components are represented by punctual features and network connections are represented by linear features.

This representation could be sufficient to display the general aspects of the network, such as the topology, but for more sophisticated analysis, such as the spatial relationship between the network and other networks or the terrain this 2D approach reveals its limits. With a 3D representation, on the other hand, it is possible to represent the network as it appears in the real world. This improves dramatically the potentiality of a GIS system.

With a 3D representation of the network and the surrounding environment it is possible to navigate the network and see were all its components are located on the territory and in relation with buildings, terrain and other networks. This allows the design of many features that can be adapted to the needs of any utility company. It is possible, for example, in case of planning maintenance intervention or creation of new network elements, to visualize the situation of the network before and after the intervention. In the long run this translates into minor costs and better service to customers thanks to a better planning phase.

Chapter 2

Analysis of existing 3D GIS for utility networks

In this chapter will be analysed existing 3D GIS software tools that can be used by an utility company to visualize and manage an utility network.

Searching the Web has been found two market-leading solutions that will be analysed in the following sections:

- Autodesk AutoCAD Map 3D.
- Bentley MicroStation

2.1 AutoCAD Map 3D

AutoCAD Map 3D mapping is a model-based infrastructure planning and management application that provides broad access to CAD and GIS data. It provides intelligent industry data models and tools that made possible to users to apply regional and discipline-specific standards. With AutoCAD Map 3D Enterprise, users have the ability to integrate spatial information into an Oracle database.

AutoCAD Map 3D is a software tool designed to be adapted to the needs of companies and governments in more fields:

- Electric & gas
- Water and waste water
- Telecommunications
- Utility and public works

- Transportation

AutoCAD Map 3D GIS mapping software offers easier access to design, GIS, imagery, point cloud, and business information from a broad range of sources, including Esri, Bentley, Oracle, GE, and other software providers. By using comprehensive data models for the gas, water, waste water, and electric industries, users can organise disparate asset information and apply industry standards and business requirements to their data. AutoCAD Map 3D also makes possible to use AutoCAD drafting and editing commands directly with a variety of geospatial formats.

Modules oriented to utility networks are three:

- AutoCAD Map 3D for Electric & Gas.
- AutoCAD Map 3D for Water, Waste water & Storm water.
- AutoCAD Map 3D for Telecommunications

This modules are also grouped into AutoCAD Map 3D for Government that integrates also the road network management.

For Electric & Gas AutoCAD Map 3D offers specific and extensible data models that aim to improve data quality standards, minimise as-built backlogs, and respond to information requests, including environmental or regulatory reporting.

For water, waste water and storm water AutoCAD Map 3D offers specific data models that aim to improve data quality standards, identifying network issues such as leaks and ageing pipes, and providing support to future decision making for operations, maintenance, and capital planning. It can also run hydrology and hydraulic analysis with Autodesk Storm and Sanitary Analysis, included with AutoCAD Map 3D for planning of urban drainage systems, storm sewers, and sanitary sewers.

For telecommunications AutoCAD Map 3D offers specific data models oriented to manage cable and fibre network.

AutoCAD Map 3D offers many other features including survey functionality to organise field measurements acquired from GPS and terrestrial sources and analysis tools.

Another important feature of AutoCAD Map 3D is the ability to integrate with other AutoDesk software solution for utility firms like, for example, AutoDesk Infrastructure Design Suite.

2.2 Bentley MicroStation

Bentley MicroStation is, , currently at version V8*i*, the Bentley's CAD solution. It is one of the market-leading CAD solutions and it is oriented to infrastructure design. On the top of MicroStation has been developed a set of 3D GIS tools, the ones oriented to utility networks are:

- Bentley Electric.
- Bentley Fiber.
- Bentley Gas.
- Bentley Water.

2.2.1 Bentley Electric & Gas

Bentley Electric and Bentley Gas address the facility management, modelling, and maintenance issues that electric or gas utilities encounter in day-to-day operations.

They integrates with Bentley Expert Designer through their configurability and customization capabilities. Pre-configured data models provide immediate productivity, yet Bentley Electric and Bentley Gas's data model independence allows for reconfiguration and customisation to meet specific needs.

2.2.2 Bentley Fiber

Bentley Fiber is a comprehensive product for designing, documenting, and maintaining outside plant fibre networks. It accommodates all the requisite fibre architectures in a geospatial environment that provides for detailed engineering calculations performed interactively during the design process. Bentley Fiber includes capabilities for GIS land base development, strand mapping, and duct management to complete the geospatial engineering environment.

2.2.3 Bentley Water

Bentley Water is a comprehensive geospatial engineering solution for municipal water networks design and management. An integrated GIS and design environment coupled with an intelligent network model customisable by the users allow municipalities and utilities to address all operations of a typical water supply network.

Bentley Water transparently integrates with the industry-leading Bentley's Haestad Methods hydraulic modelling and analysis software in order to share network connectivity, maintenance records, and operational data to run hydraulic simulations of their potable water distribution systems.

2.3 Advantages and dis-advantages of existing solutions

The software tools described before are market-leading 3D GIS solutions that allow an utility company to manage and visualise its networks' topology and data. The main advantage in the use of one of these tools are that they integrate many functions encompassing operations made on network from planning to maintenance.

Nevertheless, since GIS are moving towards a WebGIS approach a dis-advantage in the adoption of these products is that they are classical desktop products which need to be installed on every workstation that will use them, while a WebGIS solution has only to be deployed on a server in order to be accessed by clients. Another dis-advantage in the adoption of these products is that they require big investments both in hardware and in software technology. Per-license costs of the products, DBMS costs and hardware costs make these products often unaffordable by small utilities. Moreover, these products are often over-sized for small utilities, since they are designed to be used by large utilities.

With a WebGIS approach, on the other hand, it is possible to design a software tool that meets all the requirements listed before without the need of installing specific software on client workstations. A scalable, standard compliant (OGC Standards, regional and discipline-specific standards) WebGIS software tool is a better solution than traditional desktop-based ones, and can be tailored on the needs of any utility company. Since there are plenty of OpenSource standard compliant geospatial tools, it is possible to develop this solution on the basis of widely community used and supported tools that can also be adapted to the specific needs of users improving the quality of the software and its maintainability.

For the reasons just explained the aim of the project is to develop a software component that can be integrated with a WebGIS for modelling and visualising an utility network. The modelling part will integrate with OGC standard data sources (typically GeoDBMS and WFS Servers) to retrieve network data and build a 3D model of the network. For 3D visualization of the network model, the software will use WebGL technology that permits

to display hardware accelerated 3D graphics directly in a browser without the need of install specific components on client workstations.

Chapter 3

Requirement Analysis

Requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users.

Requirements analysis is critical to the success of a development project. Requirements must be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

In requirement analysis is often useful to use some categorisation scheme as a check-list for requirement coverage, to reduce the risk of not considering some important detail of the system.

In this analysis requirements will be categorised according to the FURPS+ model, a mnemonic with the following meaning:

- Functional (features, capabilities, security etc.).
- Usability (human factors, help, documentation etc.).
- Reliability (frequency of failure, recoverability, predictability etc.).
- Performance (response times, throughput, accuracy, availability, resource usage etc.).
- Supportability (adaptability, maintainability, internationalisation, reconfigurability etc.).

The "+" in the acronym indicates auxiliary factors such as: implementation, interface, operations, packaging, legal and so on.

Some of these requirements are collective called the **quality requirements** (or quality attributes) of a system. These include usability, reliability, performance and supportability. In common usage, requirements are

categorised as **functional** (behavioural) or **non-functional** (everything else).

3.1 User requirements

The user requirements phase is the problem definition phase of a software project and it is mainly focused on functional requirements.

The main activity of this phase is to *capture the user requirements and document them*. The scope of the software has to be established and the interfaces with external systems identified.

While user requirements originate in the spontaneous perception of needs, user requirements should be clarified through the criticism and experience of existing software and prototypes. The widest possible agreement about the user requirements should be established through interviews and surveys. User requirements definition is an iterative process, and requirements capture activities may have to be repeated a number of times before the documentation is ready.

Determining the operational environment should be the first step in defining the user requirements. When it has been established, specific user requirements are extracted and organised. Implementation considerations are omitted, unless they are the essence of the requirement.

Writing use cases is an excellent technique to understand and describe requirements. A **use case** defines the interactions between external actors and the system under consideration to accomplish a goal. An **actor** specifies a role played by a person or thing when interacting with the system. The same person using the system may be represented as two different actors because they are playing different roles.

Use cases treat the system as a black box, and the interactions with the system, including system responses, are perceived as from outside the system. This is a deliberate policy, because it forces the author to focus on what the system must do, not how it is to be done, and avoids the trap of making assumptions about how the functionality will be accomplished.

A use case should:

- Describe what the system shall do for the actor to achieve a particular goal.
- Include no implementation-specific language.
- Be at the appropriate level of detail.
- Not include detail regarding user interfaces and screens.

3.1.1 User requirements use cases

Use cases are a mature model to capture user (person or system) proffered interaction requirements and begin to establish some of the functional requirements before construction of a new system begins.

Proponents prefer them to large, monolithic documents which they believe cannot be simultaneously complete and meaningful, and regard them as an excellent technique for capturing the functional requirements of a system. In tables 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6 are indicated the user requirements for a 3D GIS software tool oriented to utility network modelled as use case templates. Use case diagram is reported in figure 3.1.

USE CASE: run on multiple platforms	
Summary	The software should be able to run on multiple platform and environment
Priority	Desired
User frequency	Always
Direct actor	User
Stakeholders	
Prerequisites	Different platforms available
Main scenario	Run on multiple platforms
Scenario extensions	
Notes	

Table 3.1: USE CASE: run on multiple platforms

USE CASE: display the network model	
Summary	The software has to display the network model
Priority	Essential
User frequency	Always
Direct actor	User
Stakeholders	Model, Model Generator
Prerequisites	Model has been created
Main scenario	Display model in 3D view inside a window
Scenario extensions	
Notes	

Table 3.2: USE CASE: display the network model

USE CASE: pick network element on the scene	
Summary	The software has to allow mouse picking of a network element on the scene
Priority	Essential
User frequency	Very Often
Direct actor	User
Stakeholders	Model
Prerequisites	Scene is displayed
Main scenario	Get the element at mouse position
Scenario extensions	Pick from search
Notes	

Table 3.3: USE CASE: pick network element on the scene

USE CASE: read network element information	
Summary	The software has to show information about a network element on the scene
Priority	Essential
User frequency	Very Often
Direct actor	User
Stakeholders	Model, WFS Server, GeoDBMS
Prerequisites	Scene is displayed and network element is picked
Main scenario	Display the network element information
Scenario extensions	
Notes	

Table 3.4: USE CASE: read network element information

USE CASE: edit network element information	
Summary	The software has to give to the user the capability to edit information about a network element on the scene
Priority	Essential
User frequency	Often
Direct actor	User
Stakeholders	Model, WFS Server, GeoDBMS
Prerequisites	Scene is displayed and network element is picked
Main scenario	Edit the network element information
Scenario extensions	
Notes	

Table 3.5: USE CASE: edit network element information

USE CASE: navigate through model	
Summary	The software has to be able to navigate through the model
Priority	Essential
User frequency	Always
Direct actor	User
Stakeholders	Model
Prerequisites	Scene is displayed
Main scenario	User should be able to zoom, pan and rotate inside the model
Scenario extensions	
Notes	

Table 3.6: USE CASE: navigate through model

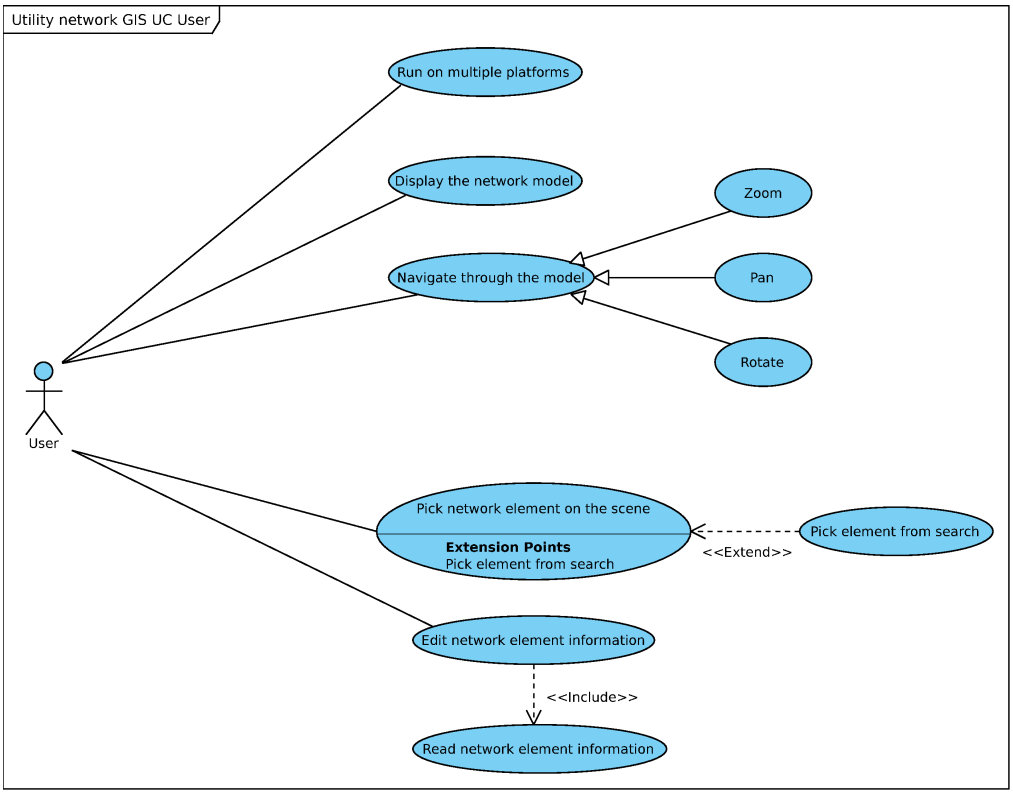

Figure 3.1: User requirements use case diagram

3.2 Software requirement specification

A Software Requirements Specification (SRS) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe all of the interactions that the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

3.2.1 Software requirements use cases

The use cases reported in tables 3.7, 3.8, 3.9 and in figure 3.2 are extensions to the user requirements analysed in the previous section from a software design point of view.

USE CASE: run inside a web browser	
Summary	The software should be able to run inside a web browser
Priority	Desired
User frequency	Always
Direct actor	User
Stakeholders	
Prerequisites	Supported web browser
Main scenario	The software runs inside a web browser
Scenario extensions	
Notes	

Table 3.7: USE CASE: run inside a web browser

USE CASE: create model from WFS Service	
Summary	The software should be able to create a model from network data stored on a WFS Server
Priority	Desired
User frequency	Often
Direct actor	User
Stakeholders	Model, WFS Server
Prerequisites	Network data are available on a WFS Server
Main scenario	Create model from WFS Server
Scenario extensions	
Notes	

Table 3.8: USE CASE: create model from WFS Service

USE CASE: create model from GeoDBMS	
Summary	The software should be able to create a model from network data stored on a GeoDBMS
Priority	Essential
User frequency	Very Often
Direct actor	User
Stakeholders	Model, GeoDBMS
Prerequisites	Network data are available on a GeoDBMS
Main scenario	Create model from GeoDBMS
Scenario extensions	
Notes	

Table 3.9: USE CASE: create model from GeoDBMS

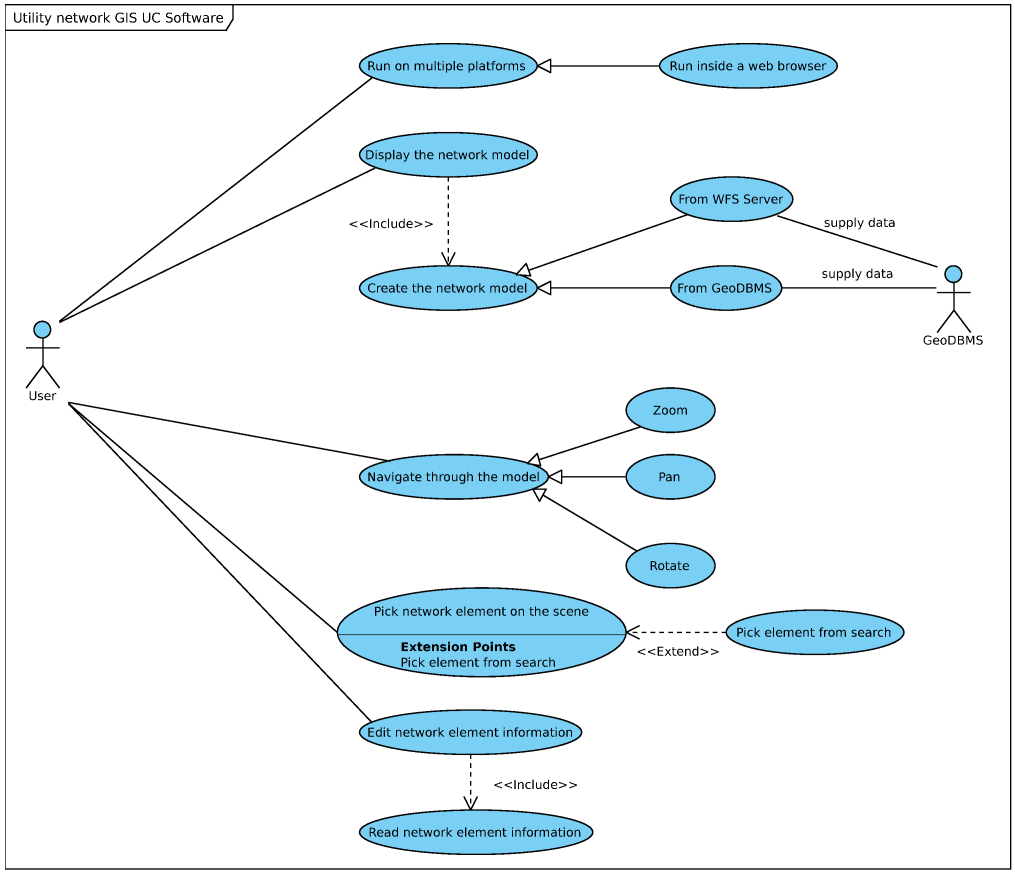

Figure 3.2: Software requirements use case diagram

Chapter 4

Development platform

The requirements defined in chapter 3 are essential to choose the right development platform to accomplish the project without big troubles and design issues.

The first topic to face with is the choice of the development language. The development language must be object oriented to permit the use of software engineering tools like UML and the use of some design patterns defined by the namesake book. Every OOP language permits to write software libraries that can be used from other software in the same, or in a different language, so this is a weak restriction, if an object oriented language is chosen.

One of the main requirements of the Network Model is the capability to be used from web applications on multiple platforms. To ensure that this requirement and the need of scalability and reliability proper of any well engineered software, Java has been chosen as the programming language. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere.". Java is currently one of the most popular programming languages in use, particularly for client-server web applications, so it is a good choice for a project that aims to be used from a web application.

With the introduction of the HTML5 standard and of the related technologies like WebGL, web applications are dramatically incrementing their potential. So, operations before unimaginable like running 3D graphics directly in a browser, are now possible. Since GIS are in continuous evolution and, as said before, are moving towards a WebGIS approach, WebGL technology potential should be investigated with the aim to design a 3D WebGIS.

Has already been noticed that two crucial requirements of the Network

Viewer are that it has to be platform-independent and embeddable in most common Internet browsers. These requirements can be met using the WebGL technology.

4.1 Graph Libraries available for Java

Once Java has been chosen as the development platform for the project, it is necessary to analyse the available graph libraries that are compliant with the requirements. A graph library is needed to abstract the network features and think to the network in terms of vertexes and edges of a graph. In this way it is possible to create a model that is independent from underlying data and also it is possible to model a network different from an utility network.

The main project requirement for the graph library is the type-safeness. In this way it is possible to build the vertex and edge models and create a graph using these models without knowing directly how the graph library works and adapt the graph representation to any need.

Searching on the web and looking for the requirements has been found some libraries that will be evaluated in the following pages. These libraries are:

- JGraphT.
- JUNG.

Another aim of the search is to select a library that permits, in a future stage of the project, also the analysis of the network graph, for example to make network flow measurements.

4.1.1 JGraphT

JGraphT is a FOSS (released under the terms of the GNU LGPL) Java library that provides mathematical graph-theory objects and algorithms.

The project is part of The Eigenbase Project which aims to provide an extensible open-source platform for building specialised data management systems in a wide variety of application spaces.

The current distribution (0.8.2) of JGraphT supports many types of graphs such as:

- Directed graphs and undirected graphs.
- Graphs with weighted, unweighted, labelled or user-defined edges.

- Various edge multiplicity options, including: simple-graphs, multi-graphs, pseudo-graphs.
- Unmodifiable graphs: allow modules to provide "read-only" access to internal graphs.
- Listenable graphs: allow external listeners to track modification events.
- Sub-graph: graphs that are auto-updating sub-graph views on other graphs.
- All composition of above graphs.

JGraphT is designed to be simple and type-safe (via Java generics). For example, graph vertexes can be of any objects.

4.1.2 JUNG

JUNG the Java Universal Network/Graph Framework is a FOSS (released under the terms of BSD License) Java library designed to support the modelling, analysis, and visualization of data that can be represented as graphs.

It was created by three Information and Computer Science PhD students at the University of California, Irvine: Joshua O'Madadhain, Danyel Fisher, and Scott White.

Its focus is on mathematical and algorithmic graph applications pertaining to the fields of social network analysis, information visualization, knowledge discovery and data mining. However, it is not specific to these fields and can be used for many other applications pertaining to graphs and networks.

The JUNG architecture is designed to support a variety of representations of entities and their relations, such as:

- Directed and undirected graphs.
- Multi-modal graphs.
- Graphs with parallel edges.
- Hypergraphs.

The current distribution (2.0.1) of JUNG includes implementations of a number of algorithms from graph theory, data mining, and social network

analysis, such as routines for clustering, decomposition, optimisation, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures.

JUNG also provides a mechanism for annotating graphs, entities, and relations with metadata. This facilitates the creation of analytic tools for complex data sets that can examine the relations between entities as well as the metadata attached to each entity and relation.

4.2 Topology library for Java

Since the Network Model has to deal with geospatial data, there is the need of a Java library to deal with geometry. Recalling that one of the main requirements is the ability to take data from multiple source and typically it isn't known a priori how the spatial data are made, it is better to develop the software thinking at geometries as they are defined by the OGC standard specifications.

The JTS Topology Suite is an open source Java software library that provides an object model for Euclidean planar linear geometry together with a set of fundamental geometric functions. JTS is primarily intended to be used as a core component of vector-based geomatics software such as GIS. It can also be used as a general-purpose library providing algorithms in computational geometry.

JTS implements the geometry model and API defined in the OGC Simple Features Specification for SQL. The software is published under the GNU LGPL and it is maintained as an independent software project by Martin Davis.

The JTS Geometry model support modelling points, linestrings, polygons, and collections. Geometries are linear, in the sense that boundaries are implicitly defined by linear interpolation between vertexes. Geometries are embedded in the 2-dimensional Euclidean plane. Geometry vertexes may also carry a z value.

User-defined precision models are supported for geometry coordinates. Computation is performed using algorithms which provide robust geometric computation under all precision models.

JTS support many geometric functions including:

- Topological validity checking.
- Overlay functions (including intersection, difference, union, symmetric difference).

- Buffer computation (including different cap and join types).
- Convex hull.
- Geometric simplification and densification.
- Precision reduction.
- Delaunay triangulation and constrained Delaunay triangulation.
- Voronoi diagram generation.
- Smallest enclosing rectangle.

Another important feature of JTS that will be useful in the project is the ability to process geometries in WKB, WKT and GML formats. In this way it is possible to have a solid basis on top of which can be designed a module that reads geometries from multiple sources.

4.3 WebGL

It has already been said that the Viewer will use the WebGL technology, so, for the sake of completeness, in the next section will be introduced this technology.

4.3.1 Introduction to WebGL

WebGL is a cross-platform API used to create 3D graphics in a Web browser. It is based on OpenGL ES 2.0 and uses the OpenGL shading language GLSL. Since it runs in the HTML5 Canvas element, WebGL has full integration with all Document Object Model (DOM) interfaces and it can be used from any DOM-compatible language (for example JavaScript).

Main advantages of WebGL technology are:

- An API that is based on a familiar and widely accepted 3D graphics standard.
- Cross-browser and cross-platform compatibility.
- Tight integration with HTML content, including layered compositing, interaction with other HTML elements, and use of the standard HTML event handling mechanisms.
- Hardware-accelerated 3D graphics for the browser environment.

- A scripting environment that makes it easy to prototype 3D graphics. There is no need to compile and link before of viewing and debugging the rendered graphics.

4.3.2 OpenGL ES 2.0 Pipeline Structure

With reference to the figure 4.1 main components of OpenGL ES 2.0 Pipeline Structure are:

- Primitive Processing.
- Vertex Shader.
- Primitive Assembly.
- Rasterizer.
- Fragment Shader.
- Fragment Operations (Depth/Stencil buffer testing, Colour Buffer Blending, Dithering etc.)

Primitive processing

In this step the application sets up an ordered list of vertices to send to the pipeline. These vertices define the boundaries of a scene geometry. Vertices are grouped to form primitives.

Primitives are basic drawing shapes, like triangles, lines, and points.

This part of the pipeline deals with a number of objects like **Vertex Array Objects** and **Vertex Buffer Objects**. Vertex Array Objects define what data each vertex has, while Vertex Buffer Objects store the actual vertex data itself.

A vertex's data is a series of attributes. Each attribute is a small set of data that the following stages will do computations on.

Vertex Shader

The Vertex Shader is called once for each input vertex. The main task of the Vertex Shader is to provide vertex positions for the following stages of the pipeline. Additionally, it can calculate further attributes that can be used as input for the Fragment Shader later. The most basic shader just takes vertex positions as input and directly assigns the input data to the `gl_Position` Varying.

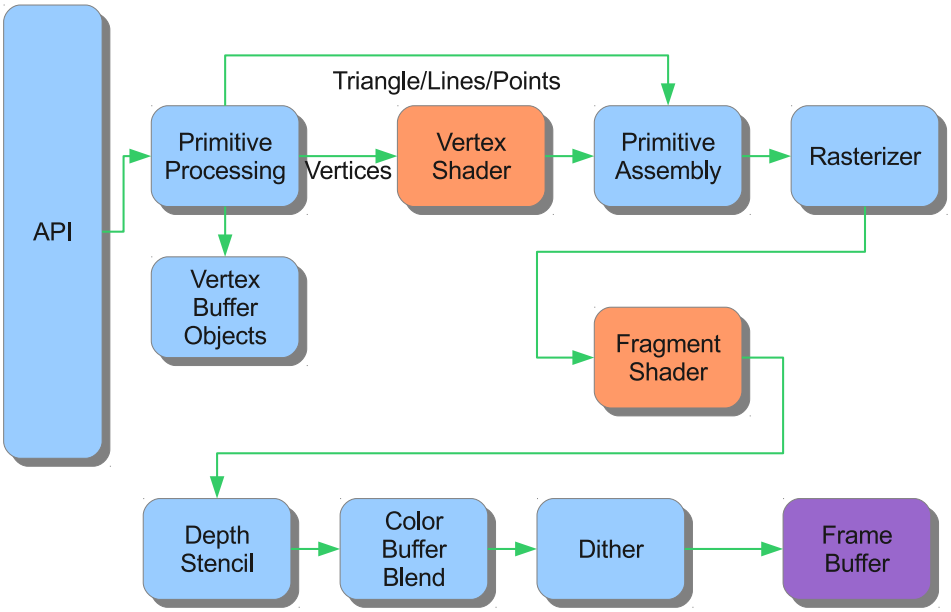

Figure 4.1: ES2.0 Programmable Pipeline

Typically, the shader does a multiplication with the model-view projection matrix (passed as a Uniform constant) to allow translation and rotation of input geometry as well as perspective projection, possibly passes texture coordinates, and calculates lighting parameters. Additional user outputs typically include:

- **Texture coordinates**. These may be just passed through from input attributes for simple texturing but also might get generated or processed for implementing reflective surfaces and environment mapping or other effects such as dynamic texturing.

- **Fog factor**. For a fog effect, the distance of the primitive from the eye can be calculated in the vertex shader. The fragment shader can later fade out the fragment based on this value.

- **Lighting parameters**. Based on light source positions (passed as Uniform constants) and vertex normals (needed as additional per-vertex input), lighting parameters can be generated for the fragment shader.

Primitive Assembly

In the Primitive Assembly stage, several coordinate transformations are done:

- **Clipping**. Primitives lying outside the viewing volume are discarded, and primitives lying partially outside the view will be clipped. Varying outputs of the vertex shader get clipped, too.

- **Perspective Division**. The three main elements of `gl_Position` (x, y, z) are normalised to $[-1.0...1.0]$ by division by the fourth vertex element w. The result is normalised device coordinates.

- **Viewport Transformation**. Coordinates are transformed to window coordinates by means of a linear transformation.

Rasterization

Rasterization is the process of creating a two-dimensional rasterized image from a scene geometry. In other words it is the process of calculating the set of fragments (pixels) for each primitive that compose the geometry. For polygon rasterization, this includes the following steps:

- **Culling**. Polygons viewed from the back can be discarded if the culling is enabled.
- **Depth Offset**. A depth offset can be applied to polygon coordinates using. This can prevent Z-fighting[1] for polygons that lie in the same plane.
- **Varying interpolation**. Vertex shader Varying outputs and depth are interpolated when prepared as input for the Fragment Shader.

Fragment Shader

The Fragment Shader is called once for each geometry fragment (pixel). The main task of the Fragment Shader is to provide colour values for each output fragment. The most basic Fragment Shader just assigns a constant value to its `gl_FragColor` output. Typically, the Fragment Shader does a texture look-up and implements lighting based on the lighting parameters the Vertex Shader computed previously.

Fragment Operations

Possible operations on fragments include:

- **Scissor testing**. If enabled, only pixels in a specified rectangular region are drawn.
- **Stencil buffer testing**. If enabled, pixels may be updated only when passing a test against the stencil buffer.
- **Depth buffer testing**. If enabled using, pixels are only drawn if passing the depth buffer test, implementing hidden-surface removal.
- **Blending**. If enabled, pixels output by the Fragment Shader may be blended with pixel values already present in the output buffer.
- **Dithering**. If enabled using, dithering may be used to increase the perceived colour depth.
- **Antialiasing**. it is possible to configure simple antialiasing.

[1] Z-fighting is a phenomenon that occurs when two or more primitives have similar values in the z-buffer. Affected pixels are rendered with fragments from one polygon or the other arbitrarily, in a manner determined by the precision of the z-buffer.

4.3.3 Getting a WebGL Implementation

The WebGL 1.0 specification has recently been released, and the latest builds of several browsers have reached or are close to reach full conformance. In the following list is shown the status of WebGL implementation in most common desktop Internet browsers.

- **Mozilla Firefox** - WebGL has been enabled on all platforms that have a capable graphics card with updated drivers since version 4.0.
- **Google Chrome** - WebGL has been enabled by default since version 9.
- **Safari** - Safari 5.1 installed on Apple Mac OS X Lion has support for WebGL, but is disabled by default.
- **Opera** - WebGL is not implemented in the latest Opera 11.51 release. However, it is partially supported by development versions for Microsoft Windows. There are plans to support it from release 12.
- **Internet Explorer** - Microsoft has not announced any plans to officially support WebGL. The Chrome Frame and IEWebGL plugins provide options to add support for WebGL to Internet Explorer.

4.3.4 Security

Since some security issues concerning WebGL technology has been discovered, it is necessary to investigate about how it is possible to mitigate these risks without having to abandon this promising technology.

Resource restrictions

Traditional browser content would not normally have direct access to the hardware in any form, if one drew a bitmap it would be handled by some code in the browser with responsibility for drawing bitmaps. This would then be likely to delegate that responsibility to an OS component, which would perform the drawing itself. While this distinction is blurring somewhat with the introduction of 2D graphics acceleration in all the popular browsers it is still the case that the actual functionality of the GPU is not directly exposed to a web page. The salient facts are that the content is pretty easy to verify, has a measurable render time relative to the content, and generally contains little programmable functionality (at least which would be exposed to the graphics hardware).

WebGL on the other hand provides, by virtue of its functional requirements, access to the graphics hardware. Shader code, while not written in the native language of the GPU, are compiled, uploaded then executed on the graphics hardware. Render times for medium to complex geometry can be difficult to determine ahead of time from the raw data as it is hard to generate an accurate value without first rendering it. Also some data can be hard to verify and security restrictions can be difficult to enforce once out of the control of the WebGL implementation.

This might not be such an issue, except for the fact that the current hardware and graphics pipeline implementations are not designed to be pre-emptable or maintain security boundaries. Once a display list has been placed on the GPU by the scheduler it can be difficult to stop it, at least without causing obvious, system-wide visual corruption and instabilities. By carefully crafting content it is possible to seriously impact the OS's ability to draw the user interface, or worse. The difficultly in verifying all content and maintain security boundaries also have potential impact on the integrity of the system and user data.

To mitigate this risk, WebGL resources such as textures and vertex buffer objects (VBOs) must always contain initialised data, even if they were created without initial user data values. Creating a resource without initial values is commonly used to reserve space for a texture or VBO. If initial data are not provided to these calls, the WebGL implementation must initialise their contents to 0; depth render buffers must be cleared to the default 1.0 clear depth. For example, this may require creating a temporary buffer of 0 values the size of a requested VBO, so that it can be initialised correctly. All other forms of loading data into a texture or VBO involve either ArrayBuffers or DOM objects such as images, and are therefore already required to be initialised.

When WebGL resources are accessed by shaders the WebGL implementation must ensure that the shader cannot access either out of bounds or uninitialised data.

A WebGL implementation must only accept shaders which conform to The OpenGL ES Shading Language, Version 1.00. In particular, a shader referencing state variables or functions that are available in other versions of GLSL (such as that found in versions of OpenGL for the desktop), must not be allowed to load.

Origin restriction

One of the fundamental security boundaries in the specification of the DOM and browser handling of JavaScript is the domain boundary. This

is to prevent content served from a domain being able to access authenticated/trusted resources on another domain. Whether content is permitted to be accessed across this boundary very much depends on the type of resource being accessed. This is sometimes referred to as "Right to Embed" vs. "Right to Read". For example it is perfectly acceptable to embed an image from outside of a domain because the underlying APIs never gave a mechanism to read the actual content (outside of image dimensions, and an indication of success or failure to load). On the other hand trying to use the `XMLHttpRequest` object to pull content from outside a domain (and therefore giving access to the raw data) is generally not permitted.

In order to prevent information leakage, the HTML5 canvas element has a `origin-clean` flag. For a WebGL context, the `origin-clean` flag must be set to false if any of the following actions occur:

- The `texImage2D` method is called with an `HTMLImageElement` or `HTMLVideoElement` whose origin is not the same as that of the Document object that owns the canvas element.
- The `texImage2D` method is called with an `HTMLCanvasElement` whose origin-clean flag is set to false.

Whenever the `readPixels` method of the 2D context of a canvas element whose origin-clean flag is set to false is called with otherwise correct arguments, the method must raise a `SECURITY_ERR` exception.

Defence against DoS

It is possible to create, either intentionally or unintentionally, combinations of shaders and geometry that take an undesirably long time to render. This issue is analogous to that of long-running scripts, for which user agents already have safeguards. However, long-running draw calls can cause loss of interactivity for the entire window system, not just the user agent.

In the general case it is not possible to impose limits on the structure of incoming shaders to guard against this problem. Experimentation has shown that even very strict structural limits are insufficient to prevent long rendering times, and such limits would prevent shader authors from implementing common algorithms.

User agents should implement safeguards to prevent excessively long rendering times and associated loss of interactivity. Suggested safeguards include:

- Splitting up draw calls with large numbers of elements into smaller draw calls.

- Timing individual draw calls and forbidding further rendering from a page if a certain time-out is exceeded.
- Using any watchdog facilities available at the user level, graphics API level, or operating system level to limit the duration of draw calls.
- Separating the graphics rendering of the user agent into a distinct operating system process which can be terminated and restarted without losing application state.

The supporting infrastructure at the OS and graphics API layer is expected to improve over time, which is why the exact nature of these safeguards is not specified.

4.3.5 WebGL Frameworks

Since WebGL is the new frontier of the web development, especially in the entertainment and scientific fields, there are plenty of frameworks that aim to ease the development of web applications that use WebGL for rendering 3D graphics.

Only some of them, nevertheless, are sufficiently supported and have sufficient functionality to be used in the project. The framework that are going to be analysed in the following paragraphs are:

- C3DL.
- SpiderGL.
- Three.js.

C3DL

The Canvas 3D JS Library (C3DL) is an open source (released under the terms of MIT License) JavaScript library that will make it easier to write 3D applications using WebGL. It was first developed as part of the CATGames Research network at Seneca College (Toronto, Canada) working on providing a middle layer API for Canvas 3D (the precursor of WebGL). It provides a set of math, scene, and 3D object classes that makes WebGL more accessible for developers that want to develop 3D content in browser but do not want to have to deal in depth with the 3D math needed to make it work.

C3DL supports loading of Collada models.

SpiderGL

SpiderGL is a a JavaScript library for developing 3D graphics web applications designed by Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli and Roberto Scopigno from the Visual Computing Lab of ISTI-CNR (Istituto di Scienza e Tecnologie dell'informazione - Consiglio Nazionale delle Ricerche)[2]. SpiderGL provides data structures and algorithms to ease the use of WebGL, to define and manipulate shapes, to import 3D models in various formats, to handle asynchronous data loading it extends JavaScript by including geometric data structures and algorithms and wraps their implementation towards WebGL. SpiderGL was designed keeping in mind three fundamental qualities:

- Efficiency: with JavaScript and WebGL, efficiency is not only a matter of asymptotic bounds on the algorithms, but the ability to find the most efficient mechanism to implement, for example, asynchronous loading or parameters passing to the shader programs, without burdening the CPU with respect to a bare bone implementation;

- Simplicity and Short Learning Time: users should be able to reuse as much as possible of their former knowledge on the subject and take advantage of the library quickly. For this reason SpiderGL carefully avoids over-abstraction: almost all of the function names in SpiderGL have a one to one correspondence with either OpenGL or GLU commands, or with geometric/mathematics entities.

- Flexibility: SpiderGL does not try to hide native WebGL functions, instead it provides higher level functionality that fulfil the most common needs of the CG developer, who can use SpiderGL and WebGL calls almost seamlessly.

SpiderGL is composed of five modules:

- GL: access to WebGL functionality.

- MESH: 3D model definition and rendering.

- ASYNC: asynchronous content loading.

- UI: User Interface.

- Space: math and geometry utilities.

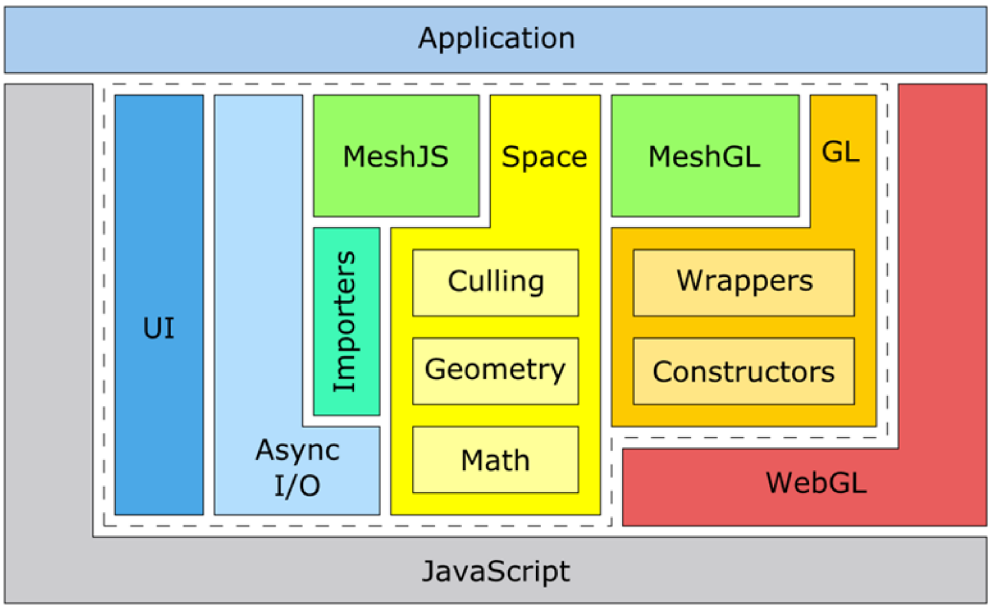

Figure 4.2: SpiderGL structure

In figure 4.2 can be seen the structure of SpiderGL.

Currently SpiderGL supports loading of models from it's own JSON format and OBJ.

Three.js

Three.js aims to create a lightweight 3D engine with a very low level of complexity. The engine can render graphics using HTML5 canvas, svg and WebGL. It is currently under heavy development and it is supported by an active community of developers.

Three.js is a scene-graph based engine, so users have to create:

1. **A scene** that contains a set of user-created Object3D objects that are elements of the scene.
2. **A camera** that will be used by the renderer to set the viewable area.
3. **A renderer** that, receiving in input a camera and a scene renders the scene using canvas, svg or WebGL.

In figure 4.3 can be seen a simplified object diagram of Three.js.

Three.js provides an interface that completely abstracts the underlining WebGL, or other renderer, internal mechanisms, so it is possible to port the same project on multiple renderers. It supports direct loading of Collada models and loading of OBJ models via its JSON Model format. Also, Three.js supports the use of shaders and permits the creation of user controls to navigate and interact with the scene.

Three.js is an OpenSource project (released only MIT License) supported by a community of about forty-five developers, and the code is hosted on GitHub.

4.4 Chosen development platform

While programming language and topology library has already been chosen, graph library and WebGL engine are still to be choose. The main aim is to get performance and scalability, while portability is ensured by a using a completely web based approach, so there is no need to install additional software on the client.

Although JGrapht and JUNG are quite similar in functionality, has been chosen JGraphT because it offers a better support to complex graph

[2]Institute of Information Science and Technology - Italian National Council of Research

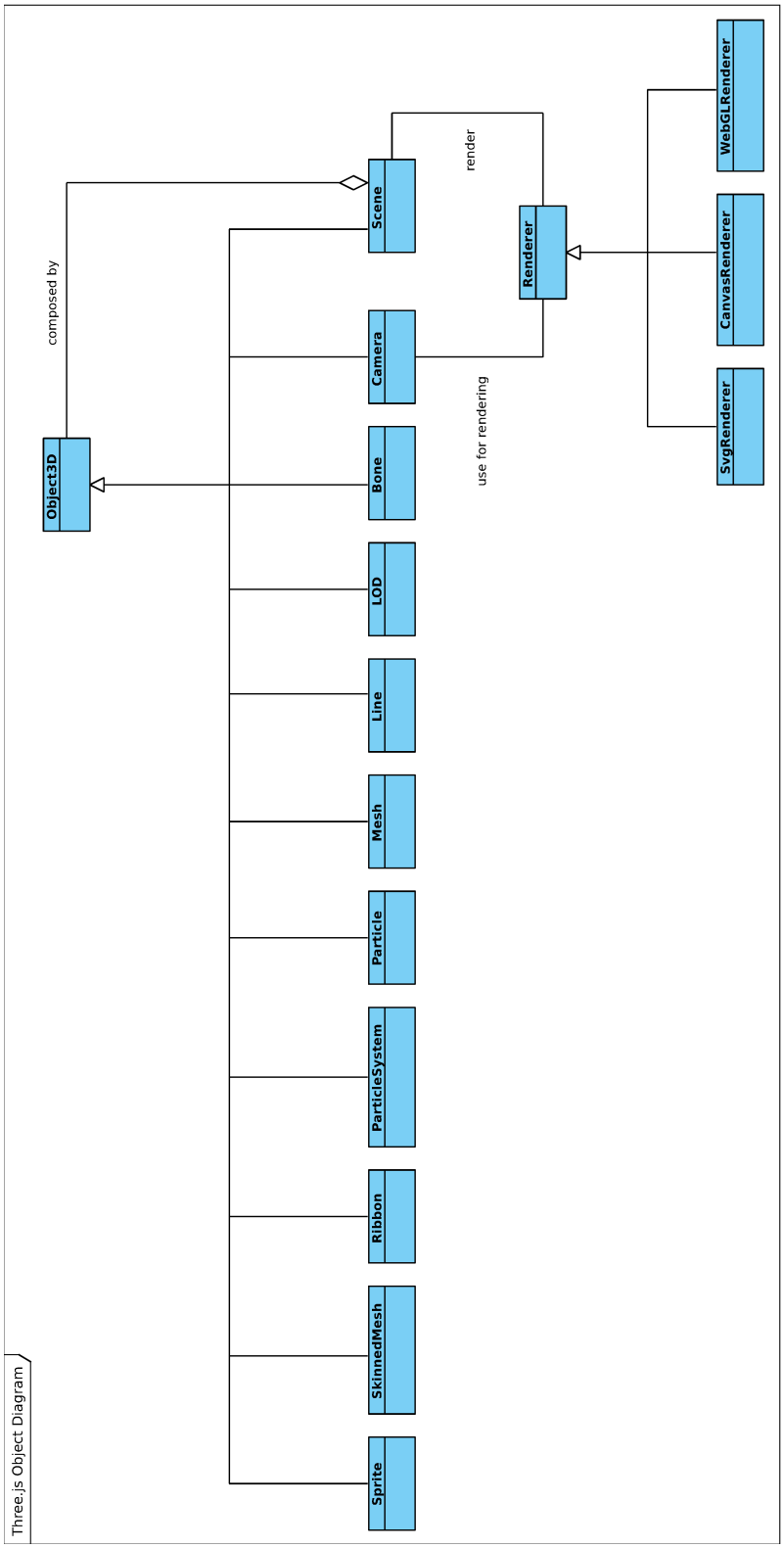

Figure 4.3: Three.js object diagram

like hypergraphs. Since the topology of the network graph isn't known a priori it could be necessary to generate almost every type of graph, and, in the future, there could be the need to apply graph-theory algorithms to these graphs. Another important aspect to take in account is that JGraphT appears to be better supported than JUNG.

Since WebGL has been chosen as the technology for 3D rendering, one of the three JavaScript libraries that have been analysed before has to be chosen. The library that will be chosen has to permit the model created by the model creator, so it should have an internal model that could be used directly or adapted to the model created by the model creator.

Although C3DL is good library for rendering and supports loading of Collada models, it lacks of some navigation controls that are required to implement navigation functions required in the project and it it too simplified to deal with the needs of the viewer that is going to be developed in the project.

SpiderGL is efficient and has a JSON model that could be extended to load a customised model but it doesn't have a scene-graph approach, so it is difficult to deal with complex scene graphs like the ones that are going to be displayed in the viewer that is going to be developed in the project. Also, it lacks a good material system and a user interaction system. Nevertheless, the most important problem with SpiderGL is the lack of support. There is no documentation apart the source code, source repository is not updated and there are no forums or mailing-list to support developers or to submit bugs and bug-fixes, in other words, it appears to be, although a very good project, a dead project so it is not a good choice.

Three.js, on the other hand, is in continuous development (currently is at revision r45) and it is supported by an active community of developers who write in the wiki, submit and fix bugs. There are many examples and the code is auto-explicative.

Three.js provides a scene-graph approach that makes easier to deal with complex scenes and a JSON model format that can be easily extended to support the needs of the project. Also Three.js has extensible camera and control models to implement the navigation and user interface and provides a good material model with the possibility of use shaders. For all these reasons Three.js has been chosen as WebGL engine for the project.

Design and development of the project will be done using Java, JTS Topology Suite, JGraphT and Three.js as operative environment.

Chapter 5

Software design, development and test

Software design is a process of problem-solving and planning for a software solution. After the purpose and specifications of software is determined, software developers will design or employ designers to develop a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view.

The software requirements analysis (SRA) step of a software development process yields specifications that are used in software engineering. If the software is "semi-automated" or user-centred, software design may involve user experience design yielding a story board to help determine those specifications. If the software is completely automated (meaning no user or user interface), a software design may be as simple as a flow chart or text describing a planned sequence of events. There are also semi-standard methods like Unified Modelling Language and Fundamental modelling concepts. In either case some documentation of the plan is usually the product of the design. A software design may be platform-independent or platform-specific, depending on the availability of the technology called for by the design.

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

- **Extensibility** - New capabilities can be added to the software without major changes to the underlying architecture.
- **Robustness** - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with a resilience to low memory conditions.

- **Reliability** - The software is able to perform a required function under stated conditions for a specified period of time.
- **Fault-tolerance** - The software is resistant to and able to recover from component failure.
- **Security** - The software is able to withstand hostile acts and influences.
- **Maintainability** - The software can be restored to a specified condition within a specified period of time. For example, anti-virus software may include the ability to periodically receive virus definition updates in order to maintain the software's effectiveness.
- **Compatibility** - The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
- **Modularity** - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
- **Re-usability** - the modular components designed should capture the essence of the functionality expected out of them and no more or less. This single-minded purpose renders the components reusable wherever there are similar needs in other designs.

A software designer or architect may identify a design problem which has been solved by others before. A template or pattern describing a solution to a common problem is known as a design pattern. The reuse of such patterns can speed up the software development process, having been tested and proved in the past.

5.1 Software life cycle

The software life cycle is the sequence of different activities that take place during software development.

There are also different deliverables produced. Although deliverables can be agreements or evaluations, normally deliverables are objects, such

as source code or user manuals. Usually, the activities and deliverables are closely related.

Milestones are events that can be used for telling the status of the project. For example, the event of completing the user manual could be a milestone. For management purposes, milestones are essential because completion of milestones allow the manager to assess the progress of the software development.

The software life cycle model adopted for the development of the project is the **Linear sequential model**, also called the *waterfall model*, since the typical diagram (see figure 5.1) looks like a series of cascades. First described by Royce in 1970, it was the first realization of a standard sequence of tasks.

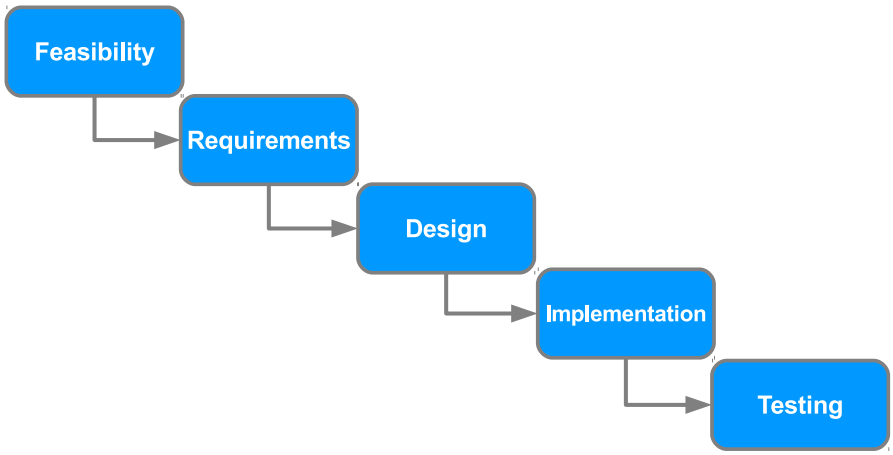

Figure 5.1: The waterfall model

There are many versions of the waterfall model. Although the specific development tasks will occur in almost every development, there are many ways to divide them into phases. Note that in this version of the waterfall, the project planning activities are included in the requirements phase. Similarly, the delivery and maintenance phases have been left off. Feasi-

bility of the project has already been considered in the previous chapters, so it will not be analysed in the following design phases.

5.2 Requirements analysis

The user requirement and software requirement analysis has already been done in chapter 3. In the following section the requirement analysis will be extended with object diagram, state diagram and system diagram.

5.2.1 Object diagram

The basic approach in an object-oriented (OO) methodology is to develop an object model that describes that subset of the real world that is the problem domain. The purpose is modelling the problem domain and not designing an implementation. Thus, entities that are essential to understanding the problem will be included even if they are not going to be included in the solution. The attributes and methods included in the object model will also be those needed for understanding the problem and not those that will just be important for the solution.

An object diagram is an UML diagram that shows a complete or partial view of the structure of a modelled system at a specific time. This snapshot focuses on some particular set of object instances and attributes, and the links between the instances. A correlated set of object diagrams provides insight into how an arbitrary view of a system is expected to evolve over time. Object diagrams are more concrete than class diagrams, and are often used to provide examples, or act as test cases for the class diagrams. Only those aspects of a model that are of current interest need be shown on an object diagram.

In figure 5.2 is shown the object diagram of the project. In this model is shown the working environment of the software. The working environment can be described in the following way. A **user** can navigate a 3D **model** created by a **model generator** from data stored on a GeoDBMS or on a WFS Server. The model contains a **scene** that is made by **scene elements**. Scene elements can be **network nodes** (for example manholes), **network edges** (pipes connecting network nodes) or a **terrain** model.

5.2.2 State diagram

The state of a machine or program is the collection of all the values of all the variables, registers, and so on. A state diagram (or state machine

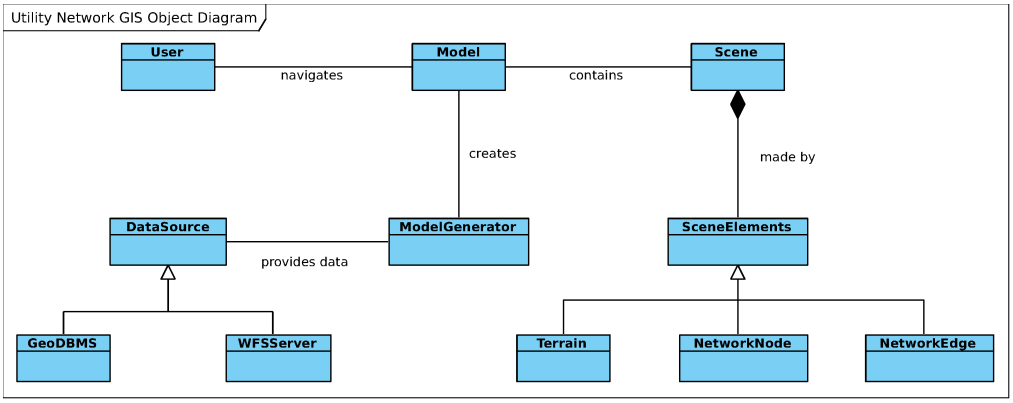

Figure 5.2: Object diagram

diagram) shows the states of the system ad the possible transitions between these states. A state diagram shows the states of the system and the possible transitions between these states. A program or machine will have an extremely large number of different states. However, many states will be similar in how the machine will behave on the next input, and so forth. A set of states with similar behaviours can be grouped together into a state. These states can be reported in a diagram to show the transition between them.

When being used as a part of the requirements specification, it is important that the states reflect domain conditions that are understandable to the users. States that are only significant to the implementation should be coalesced into domain significant states.

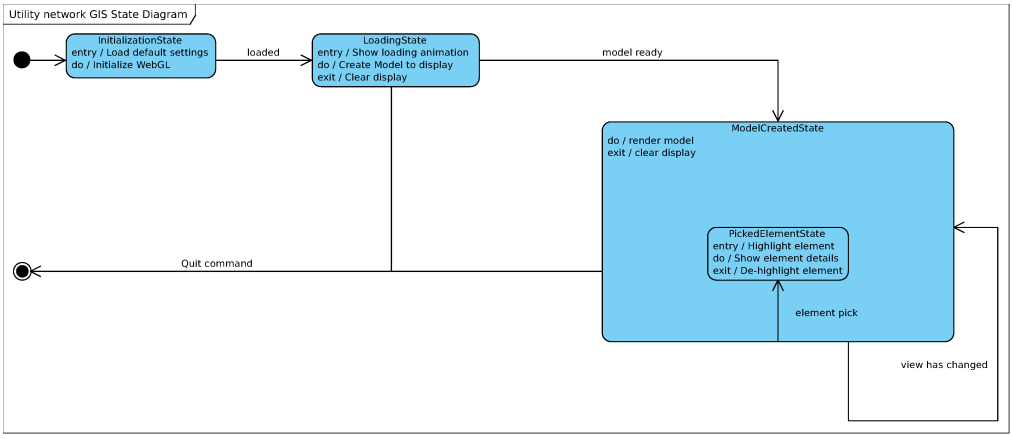

Figure 5.3: State diagram

In figure 5.3 is shown the state diagram of the project. At page loading WebGL context and user interface are initialized, then the Model Creator creates the model to display from the data-source. When the model is ready, it is rendered by the WebGL renderer and the user can navigate into it. Once the model has been created and rendered, if the user picks an element on the scene, this is highlighted and a detail window with the details of the picked element is shown. The rendering is repeated when the view changes. User can exit the viewer both when this is loading the model and when this is display the model.

5.2.3 System diagram

A system diagram is a non-formally defined diagram used to give an overview of a proposed system. It is often used when the more formally

defined diagrams are too limited to express the necessary overview.

System diagrams usually incorporate aspects of data flow and use case diagrams. They usually have ovals representing processing parts of the system, data objects representing files and/or databases, boxes representing data, and stick figures representing persons. Arcs are used to show the ow into and out of functions.

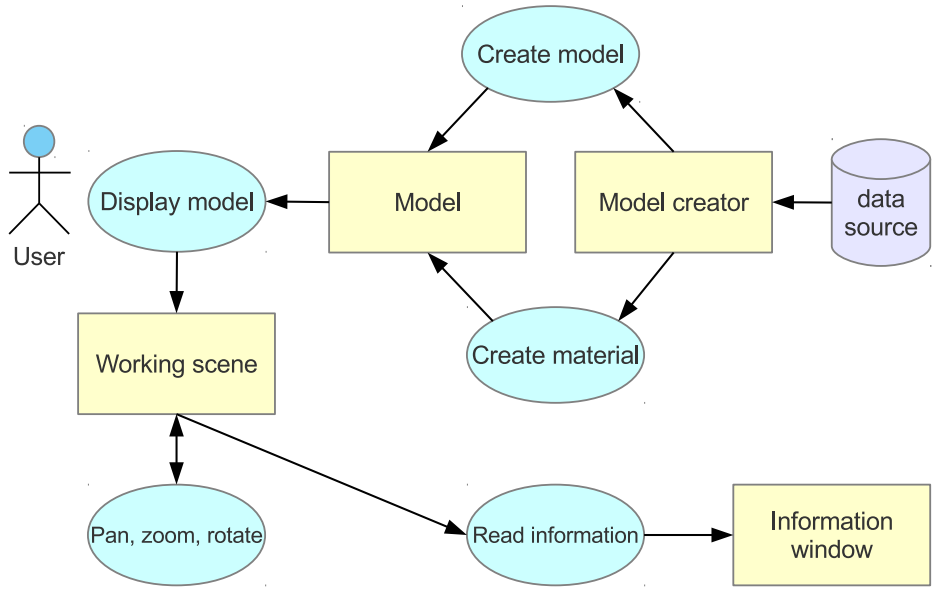

Figure 5.4: System diagram

In figure 5.4 is shown the system diagram for the project.

5.3 Architectural Design

The purpose of the Architectural Design phase is to define the structure of the software. The model constructed in the Software requirements phase is the starting point.

This model is transformed into the architectural design by allocating functions to software components and defining the control and data ow between them. This phase may involve several iterations of the design.

Technically difficult or critical parts of the design have to be identified. Prototyping of these parts of the software may be necessary to confirm the basic design assumptions.

5.3.1 Software architecture

The project will be developed using the **Model-View-Controller** (MVC) architecture. This architectural pattern isolates "domain logic" (the application logic for the user) from the user interface (input and presentation), permitting independent development, testing and maintenance of each (separation of concerns).

Model

The **model** manages the behaviour and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). In event-driven systems, the model notifies observers (usually views) when the information changes so that they can react.

In the project the model is a collection of Java classes that reflect the tables of the GeoDBMS on which network data are stored. **Hibernate** will be used to manage database access in order to provide an abstraction level between the application and data and ensure portability. In this way, for example, it is possible to reuse the same Java classes to retrieve data from a WFS Server instead of GeoDBMS maintaining the upper levels untouched.

Hibernate Hibernate is an object-relational mapping (ORM) library for the Java language, that provides a framework for mapping an object-oriented domain model to a traditional relational database.

Hibernate is free software that is distributed under the GNU Lesser General Public License. The current version is 3.6.7.

Hibernate primary feature is mapping from Java classes to database tables (and from Java data types to SQL data types). Hibernate also provides data query and retrieval facilities. Hibernate generates the SQL calls and attempts to relieve the developer from manual result set handling and object conversion and keep the application portable to all supported SQL databases with little performance overhead.

View

The **view** renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. A viewport typically has a one to one correspondence with a display surface and knows how to render to it.

In the project the view is represented by JavaServer Pages (JSPs) that communicate with the server using Apache Struts and display both the model using WebGL and the picked element data.

Apache Struts Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture. It is opensourc software that is released under Apache Software License. The current version is 2.2.3.

The goal of Struts is to separate the model from the view and the controller. Struts provides the controller (a servlet known as ActionServlet) and facilitates the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT and Velocity are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml that binds together model, view and controller.

Requests from the client are sent to the controller in the form of "Actions" defined in the configuration file; if the controller receives such a request it calls the corresponding Action class that interacts with the application-specific model code. The model code returns an "ActionForward", a string telling the controller what output page to send to the client. Information is passed between model and view in the form of special JavaBeans. A powerful custom tag library allows it to read and write the content of these beans from the presentation layer without the need for any embedded Java code.

Controller

The **controller** receives user input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and a viewport to perform actions based on that input.

In the project, to improve scalability and reuse of code, Struts Actions will not make direct calls to the model objects, so the controller is represented by a collection of Java classes (called **services**) that communicate both with the view and the model.

All Java classes in model, view and controller are bounded together using Spring Framework.

Spring Framework The Spring Framework is an open source application framework for the Java platform. It is opensource sofware released under the Apache Software License. The current version is 3.0.6.

The core features of the Spring Framework can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the Spring Framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, replacement for, or even addition to the Enterprise JavaBean (EJB) model.

The Spring Framework comprises several modules that provide a range of services. The ones used in the project are:

- **Inversion of Control container**: provides a consistent means of configuring and managing Java objects using reflection. The container is responsible for managing object life-cycles: creating objects, calling initialization methods, and configuring objects by wiring them together.
- **Data access**: working with relational database management systems on the Java platform using JDBC and object-relational mapping tools and with NoSQL databases.
- **Transaction management**: unifies several transaction management APIs and coordinates transactions for Java objects.
- **Testing**: support classes for writing unit tests and integration tests.

5.3.2 Activity diagram

An activity diagram represents the business and operational step-by-step work-flows of components in a system. An activity diagram shows the overall flow of control, describing the sequencing of activities, with support for both conditional and parallel behaviour. The activity diagram in figure 5.5 represents the main flow of an instance of the project.

5.3.3 Class Diagrams

In the Unified Modelling Language, a class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes and the relationship between the classes.

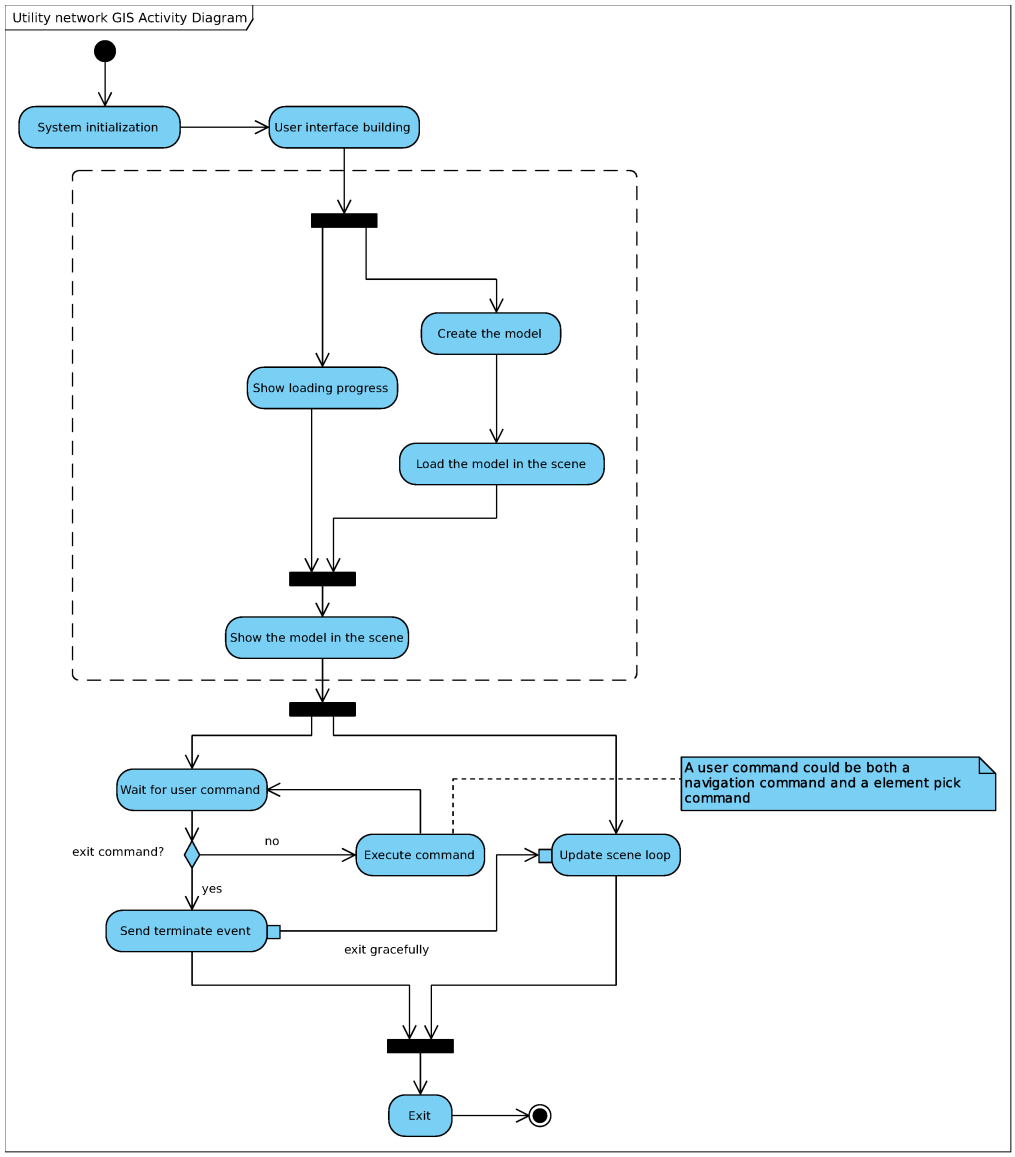

Figure 5.5: Activity diagram

In the following are reported the class diagram of most important packages in the project focusing the attention on generating network 3D model from network data.

graph3d.model.data package

This package contains the abstract data model that describes any feature (node or edge) of the network. The classes contained in this package are used by the viewer to describe the network, so the model generator doesn't need to know how data are stored. In this way it is possible to model not only an utility network, but also other types of networks (for example road network). The class diagram is shown in figure 5.6.

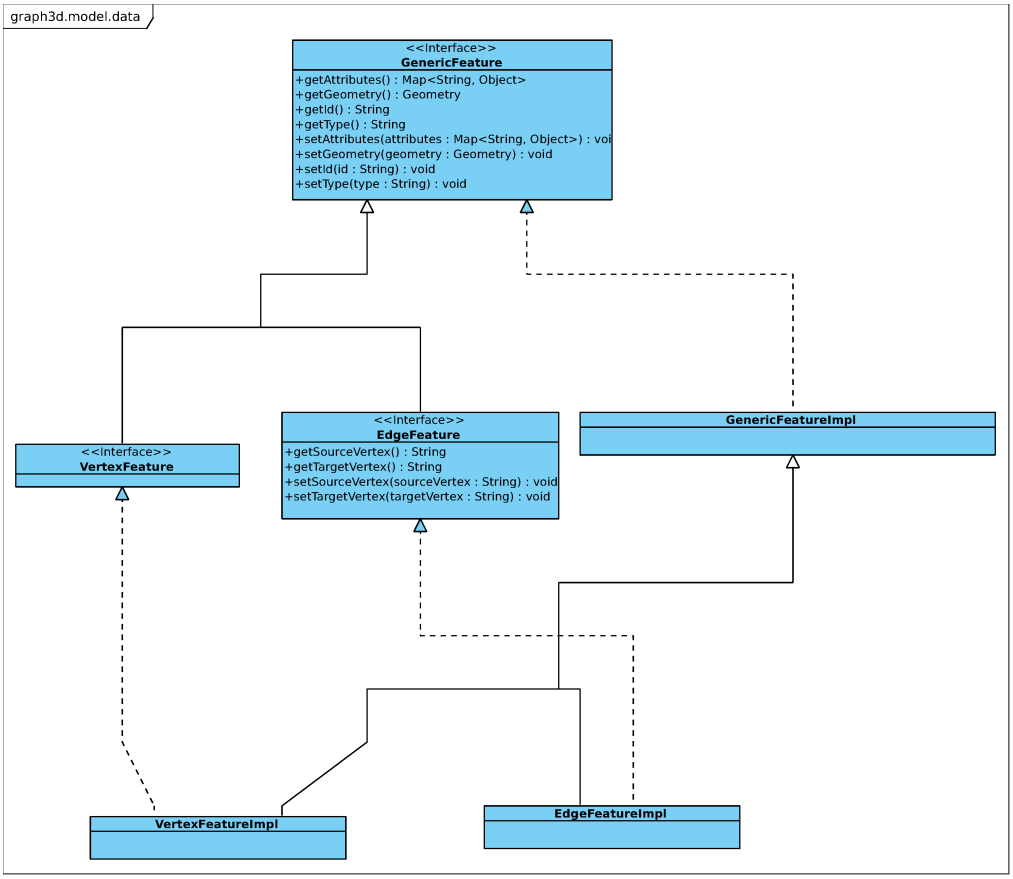

Figure 5.6: graph3d.model.data package class diagram

graph3d.model.graph package

This package contains the implementation of the graph model of the network. The class diagram is shown in figure 5.7.

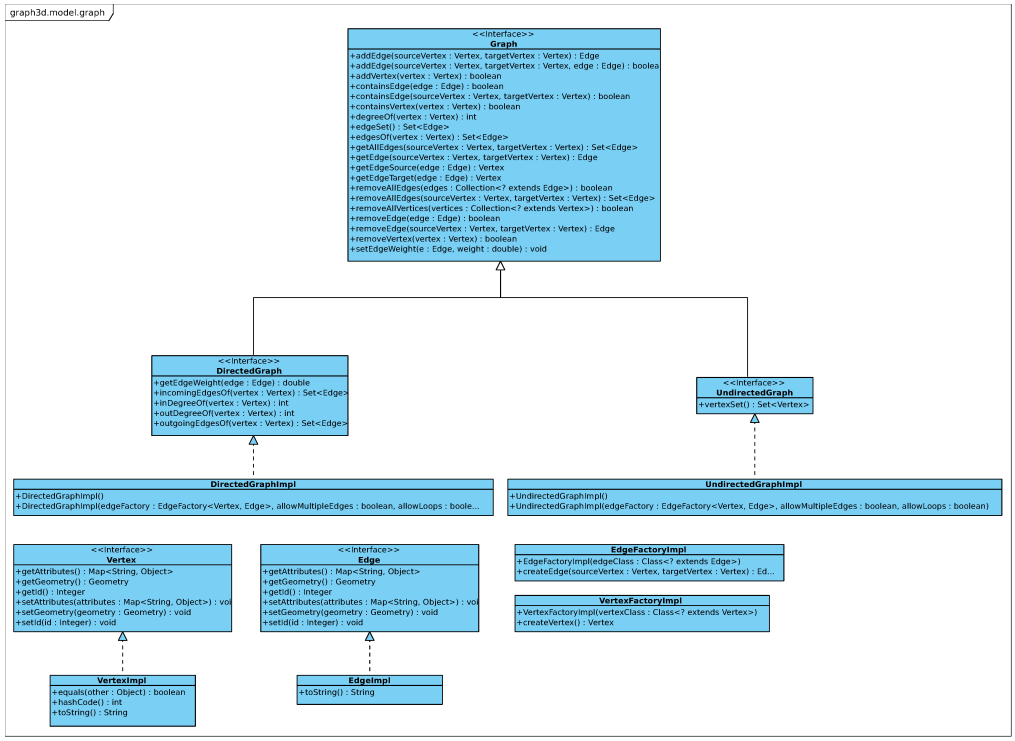

Figure 5.7: graph3d.model.graph package class diagram

graph3d.converter.graph package

This package contains classes that apply the adapter pattern to the classes in graph3d.model.data package transforming their instance into vertex or edge of the graph network model. The class diagram is shown in figure 5.8.

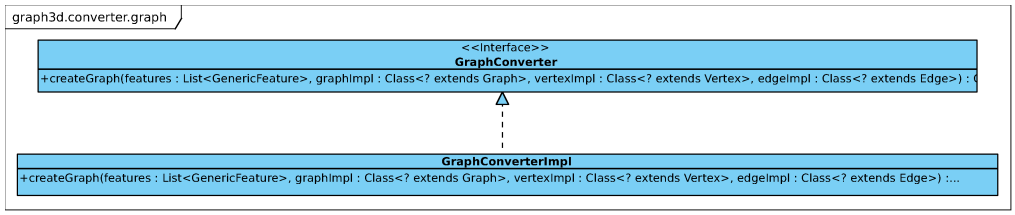

Figure 5.8: graph3d.converter.graph package class diagram

graph3d.scene package

This package contains the classes that describe the scene and are used to send data to the viewer.

graph3d.converter.model package

This package contains classes that convert a network graph representation into a scene-graph. It also translate geographic coordinates in screen coordinates and creates materials. This package can be extended by the viewer to adapt the scene representation to the needs of the specific viewer application. For example in the viewer application this package has been extended in order to add the DTM to the scene.

5.3.4 Package diagram

A package diagram depicts how a system is split up into logical groupings by showing the dependencies among these groupings. As a package is typically thought of as a directory, package diagrams provide a logical hierarchical decomposition of a system.

Package are usually organized to maximize internal coherence within each package and to minimize external coupling among packages. With these guidelines in place, the packages are good management elements. Each package can be assigned to an individual or team, and the dependencies among them indicate the required development order. In figure 5.11 is reported the package diagram of package graph3d.

5.3.5 Sequence diagrams

A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them in the order in which they occur. This allows the specification of simple run time scenarios in a graphical manner.

In figure 5.12 is reported the sequence diagram for the process of creation and rendering of a network model.

5.4 Development work-flow

The development of the project has been engineered to follow a specific work-flow to keep track of code changes, bugs and tickets. This allow better control of the project life cycle and permit easy backup and recovery in

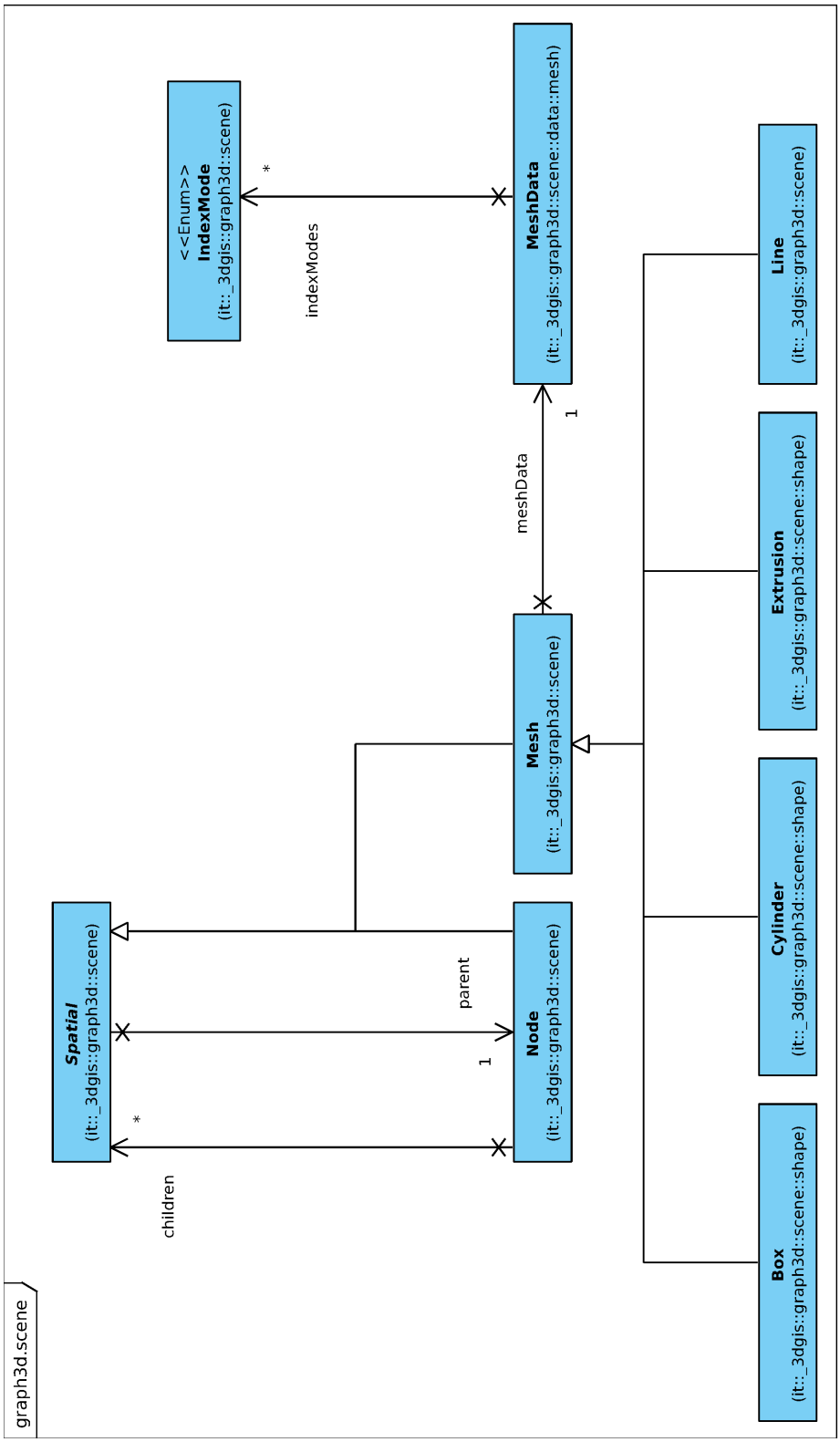

Figure 5.9: graph3d.scene package class diagram

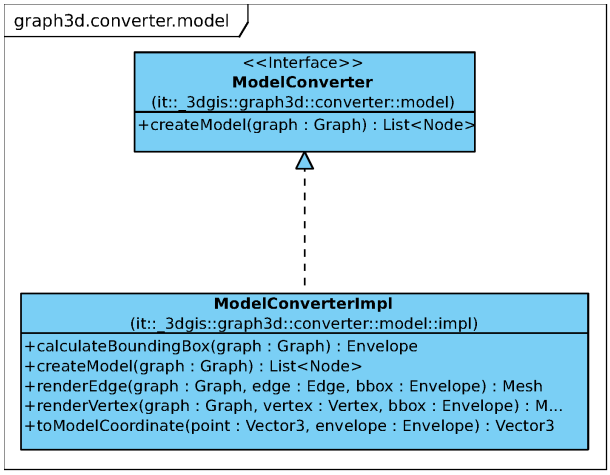

Figure 5.10: graph3d.converter.model package class diagram

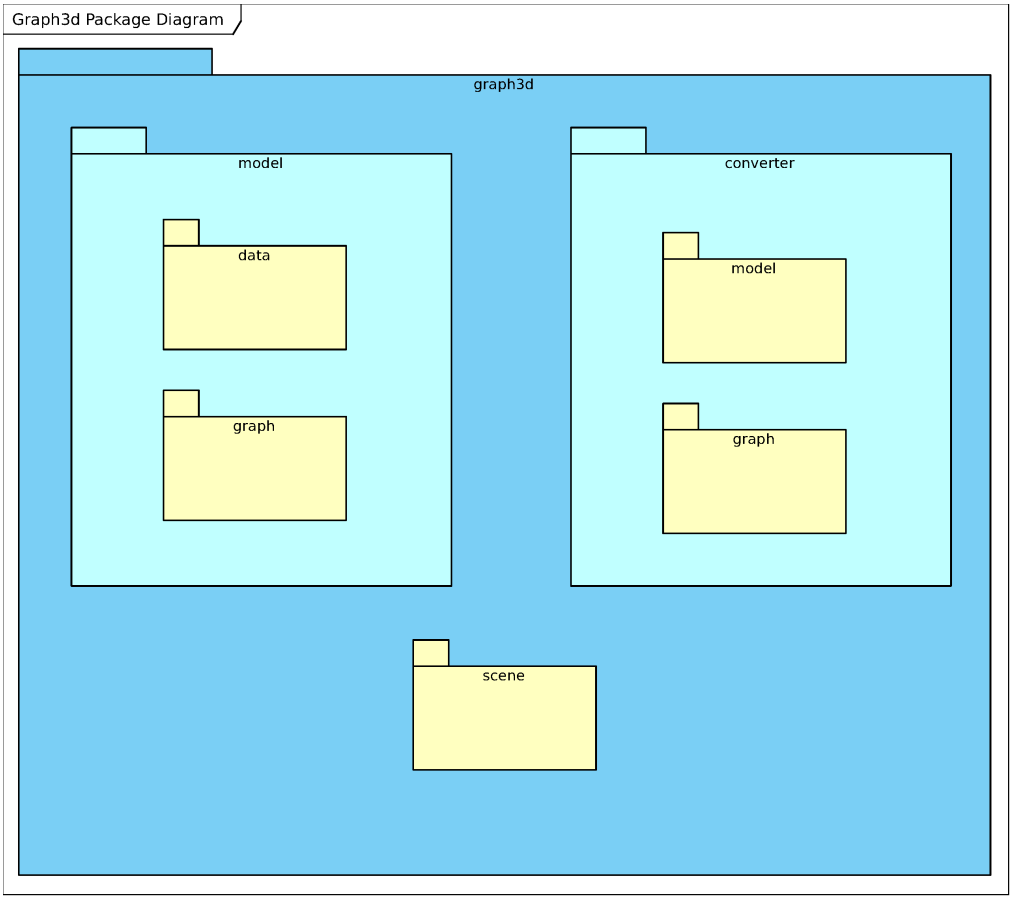

Figure 5.11: graph3d package diagram

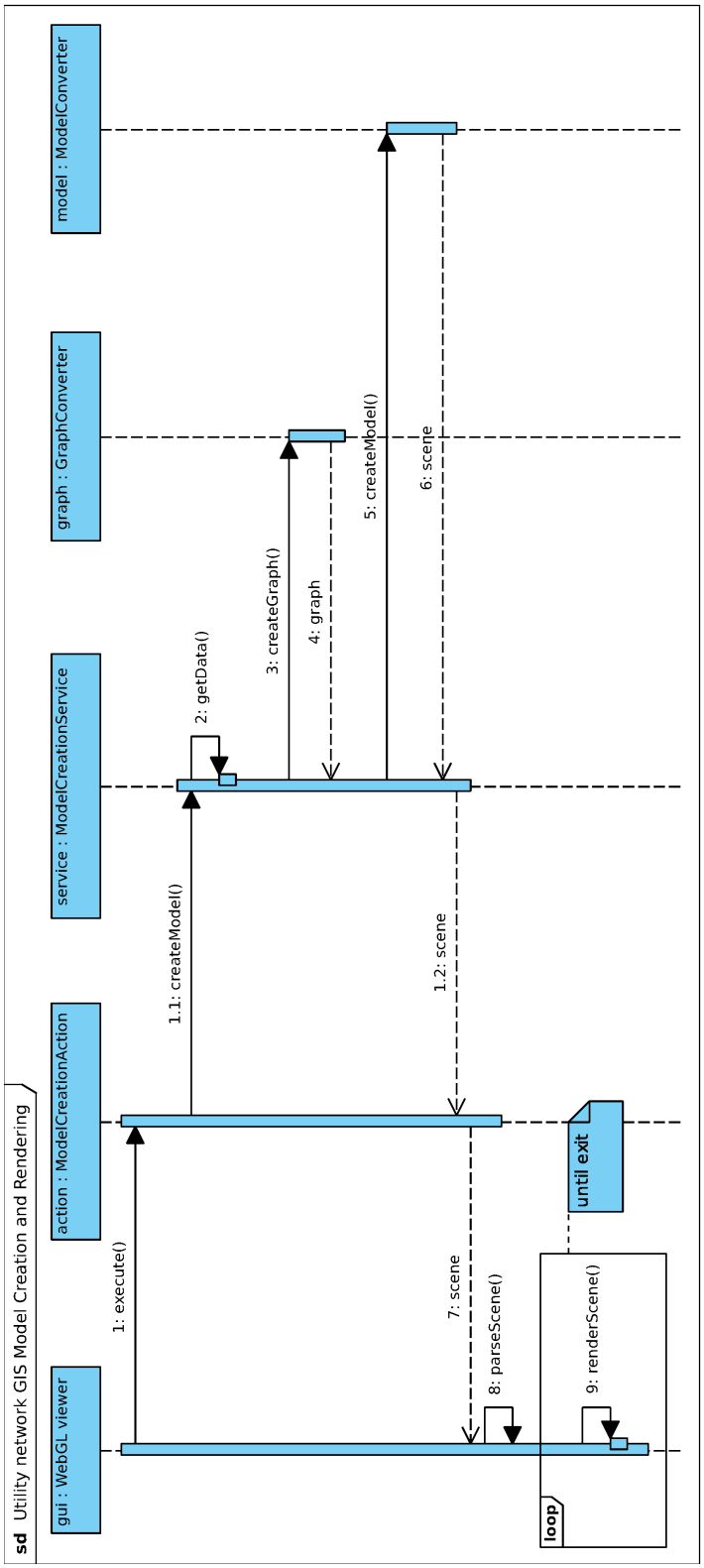

Figure 5.12: graph3d sequence diagram

case of data loss or not working code. The aim of this strategy is to be proactive and not reactive while concentrating on code quality to minimise rework.

Development tools are an essential part of this process and the ones used for the development of the project are:

- **Integrated Development Environment**: Eclipse. Eclipse is an integrated development environment (IDE) written primarily in Java. In its default form it is meant for Java developers, consisting of the Java Development Tools (JDT). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development tool-kits for other programming languages, and can write and contribute their own plug-in modules.

- **Version Control System**: Git. Git is a distributed revision control system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Git is free software distributed under the terms of the GNU General Public License version 2.

- **Issue and bug tracking**: Mantis Bug Tracker. Mantis Bug Tracker is a free and open source web-based bug tracking system released under the terms of the GNU General Public License version 2. The most common use of MantisBT is to track software defects. However, MantisBT is often configured by users to serve as a more generic issue tracking system and project management tool.

- **Dependency management and building:** Apache Maven. Maven is a build automation and software comprehension tool. It is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project. Maven uses a construct known as a Project Object Model (POM) to describe the software project being built, its dependencies on other external modules and components, and the build order. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories. Maven provides built-in support for retrieving files from the Maven Central Repository and other Maven repositories, and can upload artifacts to specific repositories after a successful build. A local cache of downloaded artifacts acts as the

primary means of synchronising the output of projects on a local system. Maven is built using a plug-in based architecture that allows it to make use of any application controllable through standard input.

5.4.1 Project work-flow

The main objectives of the work-flow used in the development of the project are the following: to maintain the local copy of the project up to date with the latest version in the repository and to keep the repository up to date with changes made by developers. If working copy is updated regularly, the probability that the changes made conflict with other changes published to the repository is reduced.

These tasks have been followed on regular basis:

- Update local copy with the latest version in the repository.
- Make changes to local copy of the project.
- Resolve conflicts.
- Publish changes.

In addition to these tasks, bug notices and enhancement tickets have been tracked using Mantis Bug Tracker to keep development under control.

5.4.2 Project JavaDoc

To support further development and code analysis, the project's documentation has been completed with JavaDoc source code documentation. JavaDoc is a documentation generator from Sun Microsystems for generating API documentation in HTML format from Java source code. JavaDoc is the industry standard for documenting Java classes.

Project's JavaDoc can be found attached to the source code distribution and it is not reported here due to its length.

5.5 Software Testing

Software testing is the process used to assess the quality of computer software. Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a

program or application with the intent of finding software bugs. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behaviour of the product against a specification.

5.5.1 Software metrics

A software metric is a measure of some property of a piece of software or its specifications. The purpose is to quantify some attribute of the objects, for example, to measure the size of software projects. Additionally, a purpose may be to predict some other attribute that is not currently measurable, such as effort needed to develop a software project.

JDepend

JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows to automatically measure the quality of a design in terms of its extensibility, re-usability, and maintainability to manage package dependencies effectively. There is a Maven plugin based on JDepend that produces an HTML metrics report and has been used for metrical analysis of the project. JDepend is distributed under the terms of BSD license.

JDepend features the following metrics:

- **Number of Classes** - The number of concrete and abstract classes (and interfaces) in the package is an indicator of the extensibility of the package.
- **Afferent Couplings** - The number of other packages that depend upon classes within the package is an indicator of the package's responsibility.
- **Efferent Couplings** - The number of other packages that the classes in the package depend upon is an indicator of the package's independence.
- **Abstractness** - The ratio of the number of abstract classes (and interfaces) in the analysed package to the total number of classes in the analysed package. The range for this metric is 0 to 1, with $A = 0$ indicating a completely concrete package and $A = 1$ indicating a completely abstract package.

- **Instability** - The ratio of efferent coupling (Ce) to total coupling ($Ce/(Ce+Ca)$). This metric is an indicator of the package's resilience to change. The range for this metric is 0 to 1, with $I = 0$ indicating a completely stable package and $I = 1$ indicating a completely unstable package.

- **Distance** - The perpendicular distance of a package from the idealized line $A + I = 1$. This metric is an indicator of the package's balance between abstractness and stability. A package squarely on the main sequence is optimally balanced with respect to its abstractness and stability. Ideal packages are either completely abstract and stable ($x = 0$, $y = 1$) or completely concrete and unstable ($x = 1$, $y = 0$). The range for this metric is 0 to 1, with $D = 0$ indicating a package that is coincident with the main sequence and $D = 1$ indicating a package that is as far from the main sequence as possible.

- **Cycles** - Packages participating in a package dependency cycle are in a deadly embrace with respect to re-usability and their release cycle. Package dependency cycles can be easily identified by reviewing the textual reports of dependency cycles. Once these dependency cycles have been identified with JDepend, they can be broken by employing various object-oriented techniques.

5.5.2 Static code analysis

Static code analysis is the analysis of computer software that is performed without actually executing programs built from that software (analysis performed on executing programs is known as dynamic analysis). In most cases the analysis is performed on some version of the source code and in the other cases some form of the object code. The term is usually applied to the analysis performed by an automated tool, with human analysis being called program understanding or program comprehension.

PMD

PMD is a static rule-set based Java source code analyser used for code analysis in the project. It is available a Maven Plugin to generate an HTML report of code analysis based on PMD.

PMD scans Java source code and looks for potential problems like:

- **Possible bugs** - empty try/catch/finally/switch statements.

- **Dead code** - unused local variables, parameters and private methods.
- **Suboptimal code** - wasteful String/StringBuffer usage.
- **Overcomplicated expressions** - unnecessary if statements, for loops that could be while loops.
- **Duplicate code** - copied/pasted code means copied/pasted bugs.

5.5.3 Code coverage

Code coverage describes the degree to which the source code of a program has been tested. It is a form of testing that inspects the code directly and is therefore a form of white box testing.

To measure how well the program is exercised by a test suite, one or more coverage criteria are used. There are a number of coverage criteria, the main ones being:

- **Function coverage** - Has each function in the program been executed?
- **Statement coverage** - Has each line of the source code been executed?
- **Condition coverage** - Has each evaluation point (such as a true/false decision) been executed?
- **Path coverage** - Has every possible route through a given part of the code been executed?
- **Entry/exit coverage** - Has every possible call and return of the function been executed?

The target software is built with special options or libraries and/or run under a special environment such that every function that is executed in the program is mapped back to the function points in the source code. This process allows developers and quality assurance personnel to look for parts of a system that are rarely or never accessed under normal conditions and helps reassure test engineers that the most important conditions have been tested.

Cobertura

The code coverage analysis tool used for the project testing is Cobertura a free Java tool based on jcoverage. It is avaliable a Maven Plugin to generate an HTML report of code coverage analysis made by Cobertura.

Main features of Cobertura are:

- Can be executed from ant or from the command line.
- Instruments Java bytecode after it has been compiled.
- Can generate reports in HTML or XML.
- Shows the percentage of lines and branches covered for each class, each package, and for the overall project.
- Shows the McCabe cyclomatic code complexity of each class, and the average cyclomatic code complexity for each package, and for the overall product.
- Can sort HTML results by class name, percent of lines covered, percent of branches covered, etc. And can sort in ascending or descending order.

5.5.4 Unit testing

Unit testing is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

JUnit

JUnit is a simple, open source framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. JUnit features include assertions for testing expected results, test fixtures for sharing common test data, test runners for running tests.

In the project JUnit has been used for unity testing because it can be tightly integrated with Eclipse IDE, used for the project development and because JUnit Tests can be run in Maven build in order to ensure that the build is not broken.

Specific test cases have been developed to test every aspect of the project. They have been used to look for bugs and to test source quality.

Chapter 6

Results

The 3D GIS software tool that has been developed in this project is able to create and visualise in a browser a network model from data stored on a GeoDBMS or on a WFS Server. In the test application a Digital Terrain Model has also been loaded into the scene using the same scene model used for the network.

The software meets all the user and functional requirements requirements listed in chapter 3. The requirements and how the developed software meets them are recapped in the following list:

- **Run on multiple platforms and inside a web browser**. The software runs in a browser, so it runs in every operating system where a WebGL enabled Internet browser runs. Since there are WebGL enabled Internet browsers running on most diffused operating system, the software meets this essential requirement.

- **Create model from GeoDBMS or WFS Server**. In the test software network data used for creating the model has been retrieved from a GeoDBMS, but the structure of the software permits also to retrieve data from a WFS Server.

- **Display the network model**. This is the fundamental requirement of the software and it is its main function.

- **Pick network element on the scene and edit element information**. Clicking on a network element on the scene, it is possible to visualise and then edit its alphanumeric information.

- **Navigate through the model**. Users, using mouse and keyboard controls, can move into the displayed scene.

6.1 System recommended requirements

The software has been tested both on Mozilla Firefox (version 4 or greater) and in Google Chrome (version 9 or greater) that support natively WebGL techology. It is developed to run in all WebGL enabled Internet browsers, but Chrome and Firefox better support WebGL, so they are recommended.

It is necessary a video card that supports WebGL technology. In the following will be reported Chrome and Firefox's requirements about video card on main operating systems.

6.1.1 Mozilla Firefox

Firefox's graphics driver blacklist is fully documented on Mozilla's wiki (`https://wiki.mozilla.org/Blocklisting/Blocked_Graphics_Drivers`). Below is a summary of the main rules affecting WebGL.

Firefox on Windows

For WebGL in Firefox on Windows, Windows XP or newer is required. The following minimal driver versions are required: either NVIDIA >= 257.21, or ATI/AMD >= 10.6, or Intel driver versions from September 2010 or newer.

Firefox on Mac OS X

For WebGL in Firefox on Mac, Mac OS 10.6 or newer is required.

Firefox on X11

In Firefox 4 and 5 on X11, only the NVIDIA driver is whitelisted. In Firefox 6 and newer on X11, the following minimal versions are required: either Mesa >= 7.10, or NVIDIA >= 257.21, or any version of FGLRX implementing OpenGL 3.0 or newer.

6.1.2 Chrome

On all operating systems, WebGL is disabled on: the Intel Mobile 945 Express family of chipsets and on NVIDIA GeForce FX Go5200

Chrome on Windows

On all versions of Windows, WebGL is disabled on all graphics drivers older than January 1, 2009. Additionally, on Windows XP, WebGL is disabled on ATI/AMD drivers older than version 10.6, on NVIDIA drivers older than version 257.21, and on Intel drivers older than version 14.42.7.5294.

Chrome on Mac OS X

On Mac OS X, WebGL is disabled on: ATI Radeon X1900 and NVIDIA GeForce 7300 GT. Additionally, WebGL's antialiasing support is disabled on most ATI/AMD cards on Mac OS X, although WebGL is otherwise supported on these cards. The exceptions, where all features are supported, are: ATI Radeon HD 4670, ATI Mobility Radeon HD 4670

Chrome on Linux

WebGL is disabled on all Intel and ATI/AMD cards.

6.2 User control

In table 6.1 are listed the navigation controls. Mouse double-click on a network element opens the detail window of the slected element.

R	fly up
F	fly down
A D	pan left and right
W S	move forward and backward
Mouse	Camera looking point movement
Directional Arrows	Camera looking point movement

Table 6.1: User controls

6.3 Screenshots

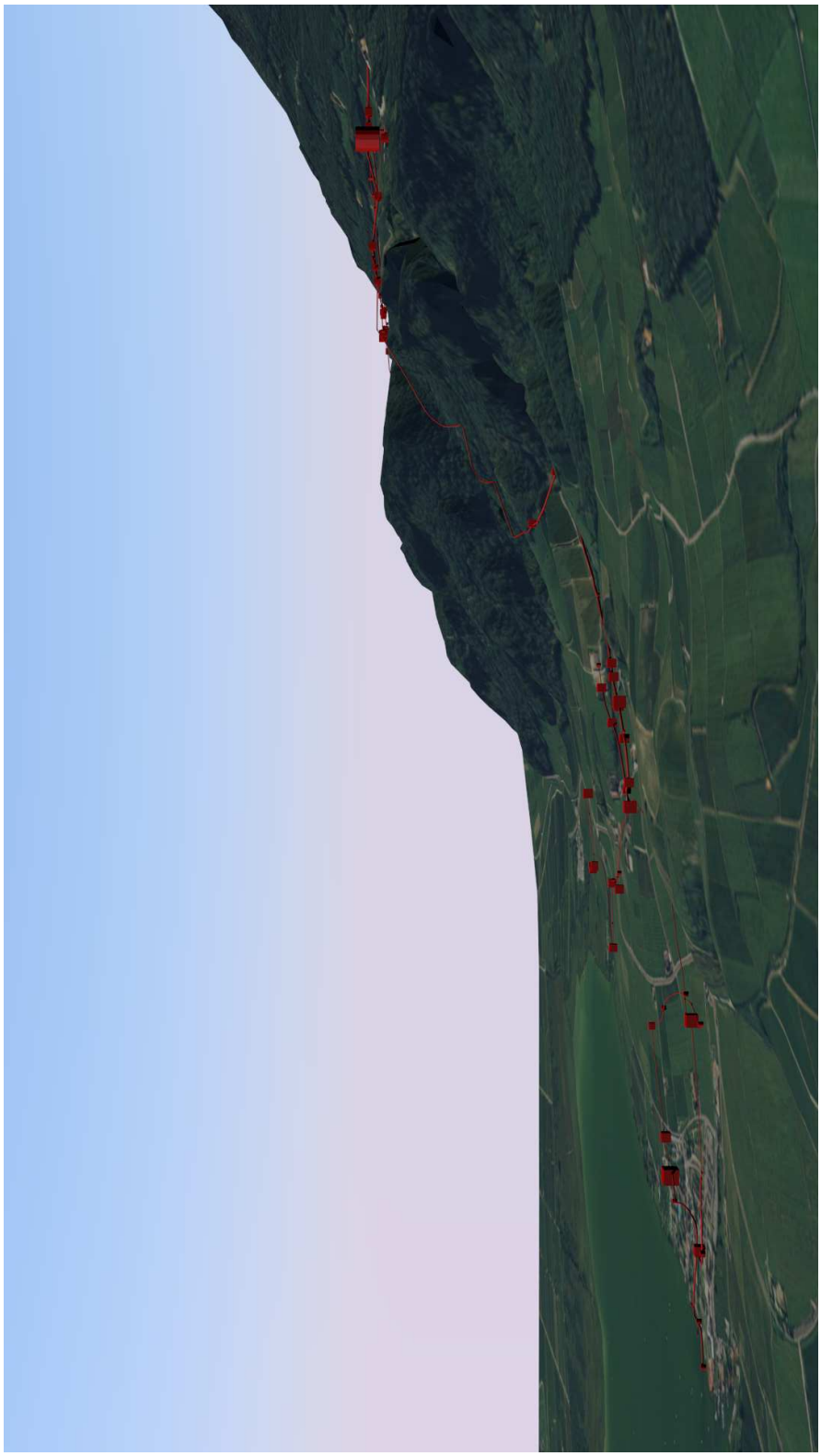

Figure 6.1: Panoramic view of the network and the terrain

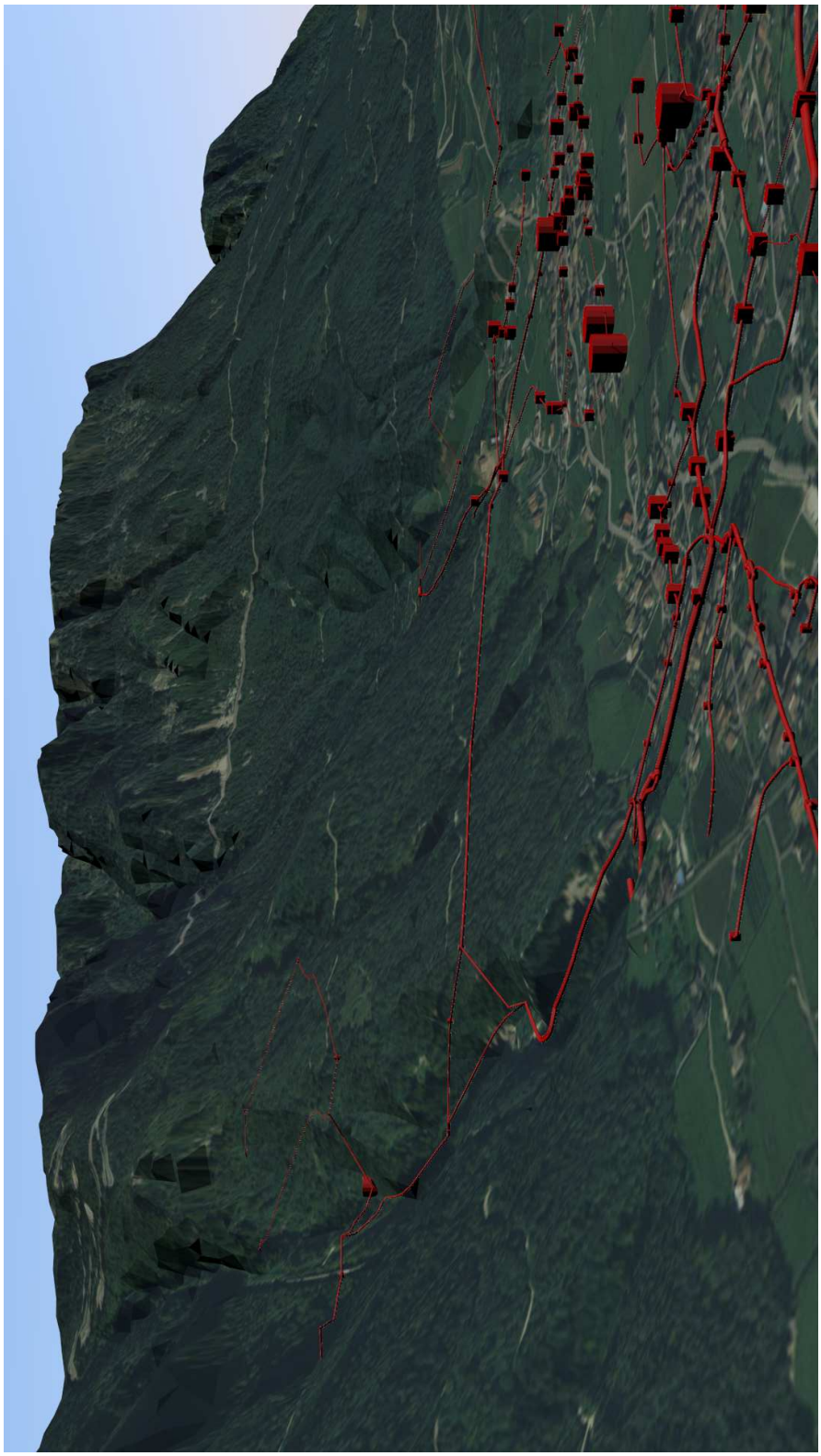

Figure 6.2: Close view of the network and the terrain

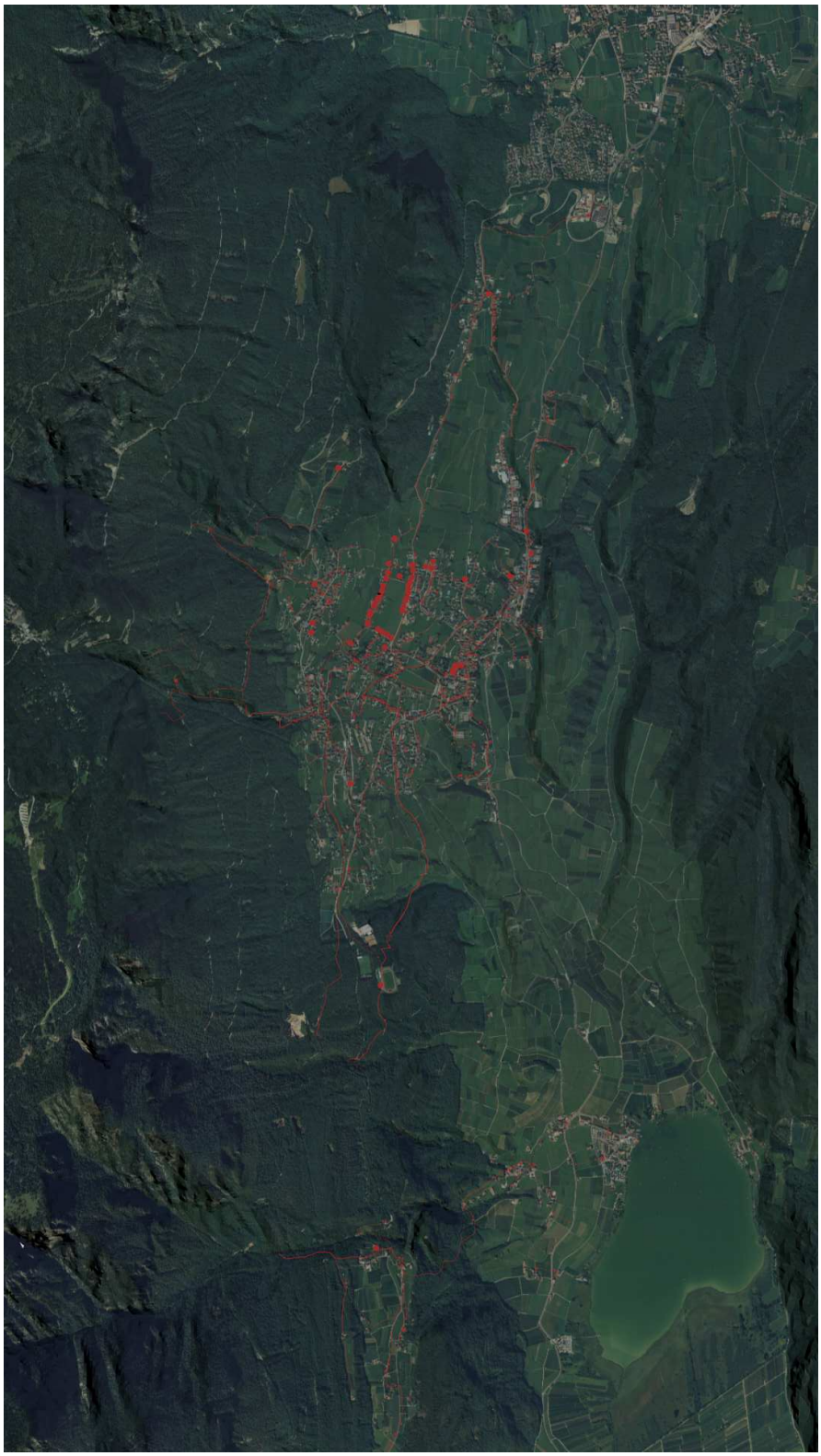

Figure 6.3: Bird's-eye view of the network and the terrain

Figure 6.4: Close view with different hillshade texture

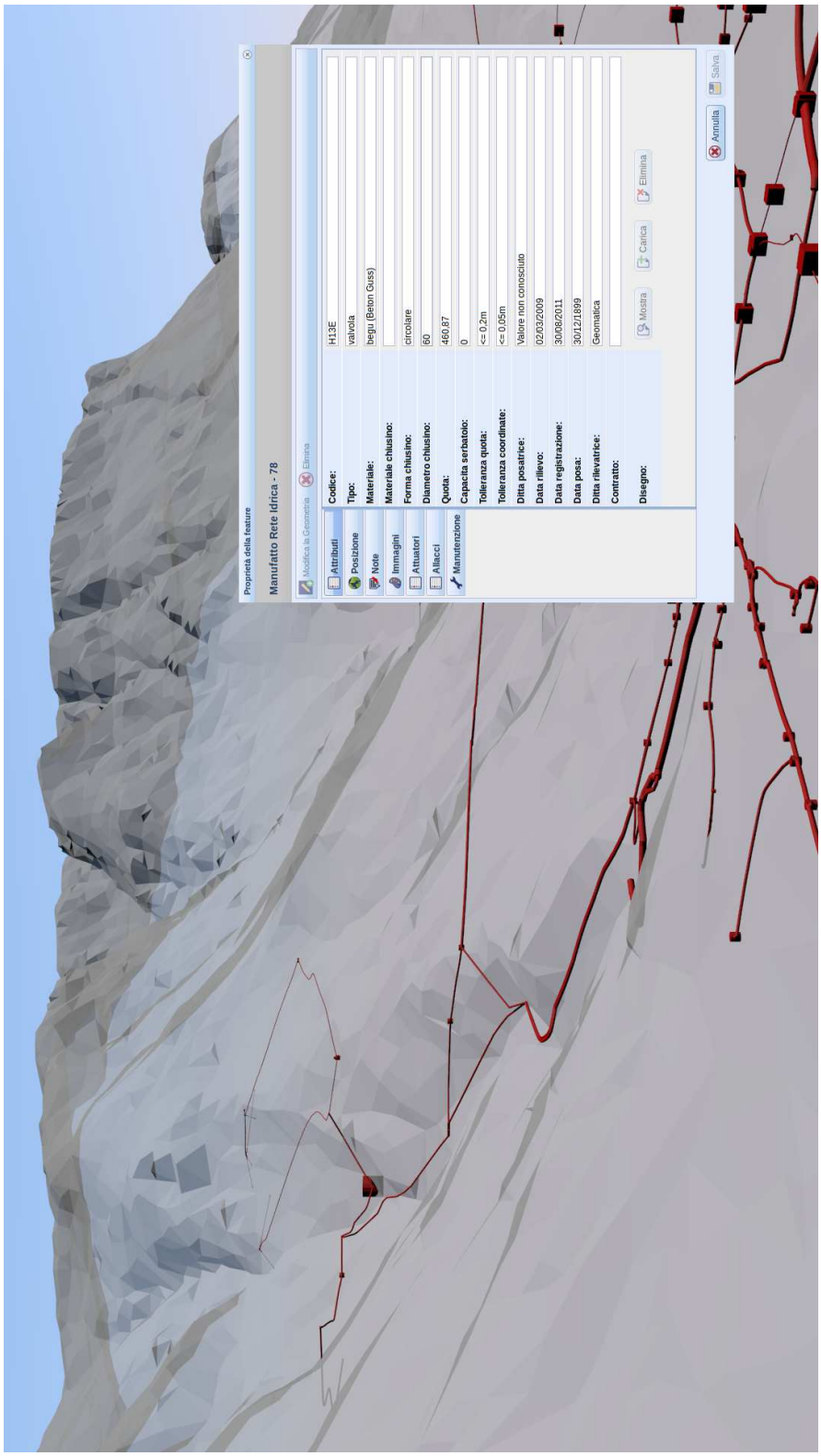

Figure 6.5: Element detail

Chapter 7

Conclusions and future development

The developed software is one of the first example of 3D WebGIS applied to utility networks. HTML5 and WebGL technology had a fundamental role in the project and it has been demonstrated that they can be used to develop full-featured WebGIS application with 3D hardware accelerated rendering of models.

The model developed in this project can be applied to other networks, so it is possible to create a WebGIS that support management and visualisation of various networks showing the spatial relations between them as they appears in the real world. This translates in better network management and better service to network users, because utility companies can have a precise idea of the impact of their interventions since the planning phase.

Integrating the network models with city models and terrain models, it will be possible to create 3D WebGIS that show a whole city, with streets, buildings, utility networks, transportation networks and so on. Such a WebGIS running in a browser and accessible from almost anywhere could have may uses, for example security and risk prevention, architectural planning, public transportation route planning or, this project aim, utility network planning.

Future roadmap of the project development includes the following features:

- **Streaming and LOD of terrain and network**. To improve rendering time and overall performance of the software will be introduced a system of LOD and streaming of terrain and network.
- **Capability to make section views of the network**. It will be

possible to make sections view on the network to have a more precise idea of the real world situation.

- **Support of CityGML standard**. The model of the network will be also exported in OGC CityGML standard. In this way it will be possible to integrate accurate utility networks models into city models.
- **Drawing of new network elements**. It will be possible to draw new network elements directly on the scene.

Appendix A

Achronyms

CAD	Computer Aided Design
DTM	Digital Terrain Model
DoS	Denial of Service
GIS	Geographic Information System
GML	Geographic Markup Language
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
VBO	Vertex Buffer Object
WKB	Well Known Binary
WKT	Well Known Text

Bibliography

[1] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*. Berlin: Springer, 1985.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Jan. 2000.

[3] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Upper Saddle River, NJ: Prentice Hall PTR, third edition ed., 2002.

[4] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer graphics: Principles and practice in C*. Addison-Wesley, 2nd ed., 1996.

[5] E. Gamma, R.Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series, Addison-Wesley, 1995.

[6] D. A. Gustafson, *Theory and problems of Software Engineering*. McGraw Hill, 2002.

[7] M. Rumor and E. Roccatello, "Design and development of a visualization tool for 3D geospatial data in CityGML format," in *Urban and regional data management: UDMS annual 2009: proceedings of the Urban Data Management Society symposium 2009, Ljubljana, Slovenia, 24-26 June 2009*, p. 31, CRC Press, 2009.

[8] "Webgl specification." `https://www.khronos.org/registry/webgl/specs/1.0/`, Feb. 2011.

[9] "Webgl public wiki." `http://www.khronos.org/webgl/wiki/Main_Page`.

[10] M. Di Benedetto, F. Ponchio, F. Ganovelli, and R. Scopigno, "SpiderGL: a JavaScript 3D graphics library for next-generation WWW,"

in *Proceedings of the 15th International Conference on Web 3D Technology*, pp. 165–174, ACM, 2010.

[11] "SpiderGL website." http://spidergl.org/.

[12] "Three.js project page." https://github.com/mrdoob/three.js/.

[13] "Canvas 3D JS Library (C3DL)." http://www.c3dl.org/.

[14] "JGraphT website." http://www.jgrapht.org/.

[15] "Java Universal Network/Graph Framework (JUNG)." http://jung.sourceforge.net/.

[16] "JTS Topology Suite." http://tsusiatsoftware.net/jts/main.html.

[17] "Spring Framework website." http://www.springsource.org/.

[18] "Hibernate website." http://www.hibernate.org/.

[19] "Struts website." http://struts.apache.org/2.x/.

[20] "Autodesk AutoCAD Map 3D website." http://usa.autodesk.com/autocad-map-3d/.

[21] "Bentley Water website." http://www.bentley.com/en-US/Products/Bentley+Water/.

[22] "ArcGIS for Water Utilities website." http://resources.arcgis.com/content/water-utilities.

List of Tables

List of Figures