



POLO TERRITORIALE DI COMO
Master of Science in Computer Engineering

NASA Web WorldWind: Visualization tool for multidimensional environmental variables

Supervisor: Prof.ssa Maria Antonia Brovelli
Co-Supervisor: Patrick Hogan

Master Graduation Thesis by:
Gabriele Prestifilippo id. 835735

Academic Year 2015/16



POLO TERRITORIALE DI COMO
Corso di Laurea Specialistica in Ingegneria Informatica

NASA Web WorldWind: Visualization tool for multidimensional environmental variables

Relatore: Prof.ssa Maria Antonia Brovelli
Correlatore: Patrick Hogan

Tesi di laurea di:
Gabriele Prestifilippo
matr. 835735

Anno Accademico 2015/16

Contents

List of Figures	4
List of Snippets.....	6
List of Tables	7
Abstract.....	8
Sommario	9
Introductive Steps	11
1.1 Virtual Globes	11
1.2 Big Geo Data	12
1.2.1 Data Collection: Crowdsourcing	13
1.2.2 Data Storing	14
1.2.3 Data Visualization	15
1.2.4 Data Analysis.....	18
1.3 Geographic data over the web.....	18
1.4 Our Solution for Big Geo Data	19
1.5 Related Work	20
Virtual Globes.....	21
2.1 Available Virtual Globes	21
2.1.1 AGI Cesium	21
2.1.2 ESRI ArcGIS Explorer Desktop.....	22
2.1.3 Google Earth.....	22
2.1.4 NASA World Wind	23
2.1.5 NASA Web WorldWind.....	23
2.1.6 SOS Explorer	24
2.2 Modern Virtual Globes.....	24
2.3 Our choice: NASA Web WorldWind	26
2.4 Voxel Model within Virtual Globes.....	26
Storing Solution for Big Geo Data	28
3.1 RDBMS: An Old Inefficient Solution.....	28
3.2 NoSQL DBMS	29
3.2.1 Key-Value Stores	30
3.2.2 Document Stores	30
3.2.3 Wide Columns Store.....	30
3.2.4 Graph Databases	31
3.3 OLAP: Multidimensional Analytic System	32
3.4 Rasdaman	34
Application Development	36

4.1	Requirement Analysis	36
4.2	Application Audience and Potential Uses.....	36
4.3	Use Cases	37
4.4	Application Introduction	37
4.5	Application Architecture: Client Side	38
4.5.1	NASA Web WorldWind Framework	39
4.5.2	jQuery	40
4.5.3	Bootstrap	41
4.5.4	jQuery UI.....	42
4.5.5	Google Charts	42
4.6	Application Architecture: Server Side	43
4.6.1	Rasdaman.....	44
4.6.2	MongoDB	44
4.6.3	Document Model.....	45
4.6.4	Node.js.....	45
4.6.5	Express	47
4.6.6	REST Interface	47
4.7	Application Components and Flow.....	48
4.7.1	Terminology and Code Convention.....	49
4.7.2	Data Importing from CSV	50
4.7.3	Doxels creation.....	51
4.7.4	Data Importing from Database	52
4.7.5	Retrieving Doxels data	52
4.7.6	Big Doxels Creation	53
4.7.7	Big Doxels Showing & Hiding	55
4.7.8	Filtering	55
4.7.9	Latitude & Longitude Filter.....	55
4.7.10	Altitude Filter	57
4.7.11	Values Filter	57
4.7.12	Time Browsing	57
4.7.13	Updating Options	58
4.7.14	Automatic Big Doxels	58
4.7.15	Starting Height.....	58
4.7.16	Statistical Index.....	58
4.7.17	Dataset Comparison	59
4.7.18	Doxel Extrusion.....	60
4.8	User Interface Demonstration	60
4.8.1	Data Importing from CSV	60
4.8.2	Personalization Options	61
4.8.3	Interface Options.....	62
4.8.4	Dataset Comparison	65
4.8.5	Voxel Extrusion	66
4.9	Multiresolution Grid Creator	67

4.9.1	Grid Typologies	67
4.9.2	Voronoi Diagram.....	68
4.9.3	Quadtree.....	69
4.9.4	Quadtree Grid Generation.....	70
4.10	Development Environment - WebStorm.....	72
4.11	Unit Testing	74
4.11.1	When to test	74
4.11.2	Why testing.....	74
4.11.3	What to test	74
4.11.4	How to test	75
4.11.5	Testing Environment.....	75
4.11.6	Tests	76
4.12	Versioning System	77
4.12.1	Local Version Control System.....	77
4.12.2	Centralized Version Control System	77
4.12.3	Distributed Version Control System	78
4.12.4	GitHub.....	78
4.12.5	SourceTree.....	78
Case Studies.....		79
5.1	Milan: Telecommunication Data	79
5.2	Turin: Point Features Data	85
5.3	West Turrock: Geological Data	88
5.4	World Coverage: Average Land Temperature	92
Conclusions		95
6.1	Evaluation of Experiences	95
6.2	Future Developments.....	95
Bibliography.....		96

List of Figures

Figure 1 – Cloud data Storage [8].....	15
Figure 2 – time slices of a 3D representation in archaeology [10].....	17
Figure 3 – Cone Tree representation of data [11].....	17
Figure 4 - Big Geo Data Architecture.....	19
Figure 5 – Color thematization of different buildings in Olbia [21].....	26
Figure 6 – Voxel representation of water temperature using EST-WA [21].....	27
Figure 7 - Big Data dimension and variety [23].....	28
Figure 8 - Graph Databases Representation [26].....	31
Figure 9 - Apache Kylin Architecture [30].....	33
Figure 10 - Rasdaman Architecture [32].....	34
Figure 11 - Possible use cases of our application.....	37
Figure 12 – Simple and complex representation of a Voxel.....	38
Figure 13 – Web WorldWind structure [33].....	39
Figure 14 – Application Architecture.....	43
Figure 15 - Sample MongoDB document.....	44
Figure 16 - MongoDB document model.....	45
Figure 17 – Node.js Server with non-blocking IO [35].....	46
Figure 18 – Multi-Threaded Server with blocking IO [35].....	46
Figure 19 – Class Diagram of ESTWA Application.....	48
Figure 20 – ESTWA Class dependencies.....	49
Figure 21 – Layer made of doxels.....	49
Figure 22 – CSV importing configurator.....	50
Figure 23 – Data Importing Flow.....	51
Figure 24 – Subdivision of a rectangle in 3x3*1 dimensions.....	53
Figure 25 – Subdivision of a rectangle in 3*3*3 dimensions.....	54
Figure 26 –Big Voxels creation flow.....	55
Figure 27 – Latitude and Longitude filter flow.....	56
Figure 28 – Importing data interface.....	61
Figure 29 – Color Range interface selector.....	62
Figure 30 – Big Doxels partially hidden.....	63
Figure 31 – Filtered Doxels on value.....	64
Figure 32 – Interface to control the filters.....	64
Figure 33 – Doxel comparison over two variables.....	65
Figure 34 – Doxel extrusion using two variables.....	66
Figure 35 – Doxel Extrusion and Comparison with three variables.....	67
Figure 36 – Quadtree structure explanation [39].....	68
Figure 37 – Voronoi Diagram structure.....	68
Figure 38 – Voronoi Sample with four nearby points.....	69
Figure 39 – QuadTree sample with four nearby points.....	70
Figure 40 – Fully populated QuadTree.....	71
Figure 41 – Real example of QuadTree from points.....	72
Figure 42 - Testing running successfully.....	77
Figure 43 - Milano Grid, BigData Challenge - Telecom Italia.....	80
Figure 44 – Settings to import data in our study case.....	81

Figure 45 - Doxels representing outgoing calls over Milano grid.....	81
Figure 46 – Filtering out low values in our study case	82
Figure 47 – Values filtered out in our study case	82
Figure 48 - Group representation of average outgoing calls	83
Figure 49 – Information about a single doxel in our study case	84
Figure 50 – Comparison of two variables in our study case.....	84
Figure 51 – Doxel extrusion in our study case.....	85
Figure 52 - 2D representation of point features	86
Figure 53 - QuadTree representation of Turin Dataset	86
Figure 54 - Doxel Representation of Turin Dataset.....	87
Figure 55 - Time representation of Turin Dataset.....	88
Figure 56 - Extrusion representation of Turin Dataset	88
Figure 57 - WestThurrock: Chalk Deposit in 4 Quadrant subdivision	89
Figure 58 - WestThurrock: Chalk Deposit in 6 Quadrant subdivision.....	90
Figure 59 - West Thurrock chalk data	91
Figure 60 - Average Land Temperature, Europe.....	92
Figure 61 - Average Land Temperature, Western Africa	93
Figure 62 - Average Land Temperature, North Italy.....	94
Figure 63 - Average Land Temperature, North Italy, 12 Months.....	94

List of Snippets

Code Snippet 1 – Structure of the parsed data	50
Code Snippet 2 – Color Retrieval through getColor method.....	52
Code Snippet 3 – Process to filter the longitude from UserInterface	56
Code Snippet 4 – Changing time from the UserInterface	57
Code Snippet 5 – Switch for the statistic in GlobeHelper	59
Code Snippet 7 – sample test structure in Jasmine	75
Code Snippet 6 – Testing two properties of an object.....	76

List of Tables

Table 1 – Technical characteristics of main Virtual Globes.....	21
Table 2 – Features of main Virtual Globes	24
Table 3 - Main NoSQL DBMS.....	31
Table 4 - OLAP Cube Representation [29]	32
Table 5 – Example of CamelCase notation	50

Abstract

In this work of thesis, we presented a Web application created using the NASA Web WorldWind framework. NASA Web WorldWind is a Web Virtual Globe, which runs in any modern browser thanks to JavaScript and HTML5. It born to visualize geospatial data, such as satellite images and environmental data. Although it comes with a well comprehensive set of APIs (Application Programming Interface) to customize the globe, and thus, it offers several ways to enrich the globe and extend its functionalities. Nowadays as an individual, we generate a large volume of data, specifically geographic ones. Several ways of collecting, visualizing and analyzing data arose in the last few years.

To face with the necessity of visualizing, storing and analyzing these data, we developed a whole system permitting to handle in a proper way various datasets.

We, thus, tested the application with some sample datasets, coming from different areas. We created a use case for telecommunication data, using as sample data from “Telecom Italia Big Data Challenge” 2015. In this case, we demonstrated how call records, SMSs, and internet usage can be shown on a Virtual Globe and, we also presented, possible analysis that could be done.

Another case study we created, refers to the city of Turin, showing data gathered from weather stations. Here we created a customized grid, generated from a QuadTree algorithm to implement a suitable model.

A different use case shows some geological data, representing a three-dimensional model of Chalk deposits in West Thurrock.

And finally we illustrate how we can connect to an external database in real-time to retrieve any kind of data, and as use case we retrieved the average land temperature, available for the whole globe, during the year 2014.

Moreover, all available standard WMSs (Web Map Services) and WCSs (Web Coverage Services) can be connected to enrich the context of data. Data visualization is provided together with summary statistics, such as minimum, maximum, average, standard deviation, range and correlation between two variables in the dataset. This approach allows an easy and interactive way to browse the data, keeping them in their real geospatial context and providing at the same time basic statistical analysis, useful for their exploration. The developed tool is multipurpose and can show several kinds of data. For this reason, in this work of thesis we presented several use cases to show how the application can be applied to different datasets.

Sommario

In questo lavoro di tesi, presentiamo un'applicazione Web creata usando il framework NASA Web WorldWind. Esso è un globo virtuale, che può essere visualizzato su qualsiasi browser moderno grazie alle tecnologie JavaScript e HTML5. Web WorldWind è nato per visualizzare dati geospaziali, quali immagini satellitari e variabili ambientali. Nonostante ciò, viene fornito con un vasto set di API (Application Programming Interface) per personalizzare le funzionalità, e quindi offre diversi modi per arricchire il globo virtuale ed estendere le sue caratteristiche di base.

Al giorno d'oggi, come individui, generiamo un grande volume di dati, specialmente geografici. La necessità di trovare modi diversi di collezionare, visualizzare ed analizzare dati, è emersa principalmente negli ultimi anni. Per far fronte a questa necessità, abbiamo sviluppato un intero sistema che permette di gestire in modo opportuno diversi set di dati.

Abbiamo quindi testato l'applicazione con diverse tipologie di datasets, provenienti da diverse aree. Abbiamo creato un caso d'uso per dati di telecomunicazione, usando come esempio i dati provenienti dal "Telecom Italia Big Data Challenge" del 2015. In questo caso, abbiamo dimostrato come record di chiamate, SMS ed uso di internet, possano essere visualizzati su un globo virtuale, ed abbiamo anche presentato possibili analisi che possono essere effettuate.

Un altro caso di studio che abbiamo creato si riferisce alla città di Torino, dove mostriamo dei dati raccolti da diverse stazioni meteorologiche. In questo esempio, abbiamo creato una griglia personalizzata, rappresentata da una struttura QuadTree, generata da un algoritmo apposito che implementa un modello appropriato.

Un differente caso d'uso mostra alcuni dati geologici, mostrando un modello tri-dimensionale di depositi di gesso a West Thurrock.

Infine, illustriamo come abbiamo sviluppato un sistema per connetterci ad un database esterno in tempo reale per ottenere svariati tipi di dati. Come caso di esempio abbiamo usato dei dati contenenti informazioni sulla temperatura del terreno, disponibile per l'anno 2014, con una copertura globale.

Inoltre, tutti gli standard WMSs (Web Map Services) e WCSs (Web Coverage Services) possono essere aggiunti alla nostra applicazione per arricchire il contesto in cui i dati vengono visualizzati.

La visualizzazione dei dati viene fornita insieme a statistiche, quali minimo, massimo, media, deviazione standard, intervalli dei valori e correlazione tra due variabili in un set di dati. Questo approccio permette un metodo facile ed interattivo per navigare attraverso dei dati, mantenendoli nel loro contesto geospaziale e fornendo allo stesso tempo statistiche di base, utili per la loro esplorazione. Lo strumento sviluppato ha diverse funzionalità e può mostrare diverse tipologie di dati. Per questo motivo, in questo lavoro di tesi, abbiamo presentato diversi casi d'uso per mostrare come l'applicazione può funzionare con svariate tipologie di dati.

Introduction

In the last decade, the increase of data collected increased exponentially, in the same way, methodologies to store, visualize and process a huge quantity of data has become fundamental. Big data comprehends many sub categories and an important one is represented by the big geo data. The big geo data consists of sets of data in which there is information about the location. Therefore, in this work of thesis we wanted to design and implement a web application to handle and visualize multidimensional geo-data. To achieve so we had to fulfill several secondary goals.

- Understanding the geo-data and the multidimensional data.
- Understanding the development process of a modern Web application according to a principled methodology.
- Developing a complete rich Internet application featuring data interaction and visualization.
- Learning the design of the architecture, data structures, and interactions of a modern Web application.
- Applying client-side scripting in the JavaScript language, by designing the dashboard front-end using state-of-the-art client-side frameworks.
- Understanding the usage of frameworks and how to extend their functionalities.

This paper is organized as follows:

The next chapter explains what is the current state of the art regarding Virtual Globes and shows some related works. Moreover, it will explain the available solutions for handling big data and further describe the one we adopt.

The third chapter shows the available solutions to store big data and some analysis performed using these technologies.

The fourth chapter presents the development of a Web application to visualize data and illustrates the models we adopted. This chapter is divided into other parts: a first part to show the internal architecture and the tools used. A second part, explaining the user interface, and its functionalities. The third part explains what happen internally in the application when we use it, showing some algorithm and the code used for it.

The fifth chapter presents our case studies. These show examples of where our application can be used and what are the advantages. Within this chapter, all the techniques we used are explained, and some sample images are provided.

The last chapter explains what we achieved and ways that our application might be further optimized and extended.

Chapter 1

Introductory Steps

1.1 Virtual Globes

"Life happens in three dimensions, so why doesn't science?" [1]

In recent decades, Virtual Globes have changed the way we can interact with visualization of data, improving the user experience for understating different phenomena. Their proliferation followed the development of VRML - Virtual Reality Modeling Language - in 1994 providing the opportunity to share a Virtual Globe on the web. This permitted the web community to see and interact with a three-dimensional representation of Earth.

Although, developing with VRML was an activity reserved for a select few since it needed sophisticated programming skills to create even very simple applications.

A few years later, with the start of the new century and the increasing evolution of the Web, new technologies arose, along with improved tools. Together these made the 3D visualization more available to a much larger community.

Earth Viewer, created by Keyhole Inc., was presented in 2001 and afterward acquired by Google in 2004. Earth Viewer renamed in 2005 into Google Earth became a point of reference for many scientists and a broader public. One unique characteristic of the virtual globe, besides the possibility of exploring the globe in three dimensions, is the DEM – Digital Elevation Model – which shows the surface elevation, putting the user into a very real view of the world.

Meanwhile, another important point of reference for the GIS community was the release of NASA WorldWind in 2003 as open source. With NASA WorldWind, the approach to virtual globes changed, greatly expanding the user community who could utilize Virtual Globes to interact with geo-data. The essential difference WorldWind provided versus Google Earth was the opportunity to customize a virtual globe and create any application one might imagine.

Even though this was just the beginning of the Virtual Globe era, these new possibilities discoveries into broader areas of study. The use of Virtual Globes began to escalate in sectors such as: Education, Research for GIS, Disaster Response, Data Analysis and verification, Geo-Collaboration and much more. Michael F. Goodchild Professor of Geography at the University of California said:

"It's like the effect of the personal computer in the 1970s, where previously there was quite an élite population of computer users" [1]

The community of GIS users is still rapidly increasing, thanks to the advantage of Virtual Globes now running on the Web, without the need for any external components. And what is also creating a favorable environment for development and use of Virtual Globes is the increase of spatial data resources available. The vastly increased capability for understanding of the data made possible by this spatial data being presented in its native context of a virtual globe greatly enhances research and the decision-making process.

1.2 Big Geo Data

Different kinds of data have become available in the last years, thanks to the advances in various technology fields. For instance, a lot of satellites orbiting above the Earth provide enormous amounts of data daily. Among all the information acquired from satellites, some are available to everyone without costs – such as Sentinel’s data - while other may be available at a price. For this reason, we can talk about Big Data.

Big data is a term for massive data sets having large, more varied and complex structure with the difficulties of storing, analyzing and visualizing for further processes or results. The process of research into massive amounts of data to reveal hidden patterns and secret correlations named as big data analytics. [2]

Big Data refers to a large amount of different data, but what is incumbent to all spatial data is the necessity to access, store, analyze and visualize. Geo-Data is data that contains information specific to a geographic location. Geo-Data can be visualized using its geographic location on a map, or have a more flexible representation in a virtual globe, so it can be experienced in its native context.

When we talk about Big Data, there are several reasons why we use the word “Big”. A large volume of data is being collected in a short amount of time. The data for each ‘snapshot’ in time might be large and the frequency (Hz) of data sampling might also be large. This volume easily reaches terabytes and even petabytes of data in a day. Using Walmart¹ as an example, it collects around 2.5 petabytes of unstructured data from 1 million customers every hour [3]. Having such quantity of data brings many advantages because they can be analyzed and much information can be extracted from them. This information can be further used in a variety of ways; it can be sold to third parties, used for decision-making and applied to many other fields.

A second fundamental point when talking about Big Data is the variety of collected data. Among the data collected, we can find photo, video, audio, demographic, social data, and more. It is important to note that much, if not most of these data will be unstructured.

The last BigData point to be defined is the frequency in which data are collected. To understand better ‘data velocity’ we can analyze how the data are collected, but in general, many terabytes of data can be generated in real time via mobile applications, computer usage, and monitoring sensors, both mobile and fixed.

¹ <http://www.walmart.com>

The combination of all these factors generate a gigantic volume of different and unstructured data, giving rise to multiple challenges especially when there is a need to manage this data efficiently.

1.2.1 Data Collection: Crowdsourcing

Every day, large amounts of data are collected. However, they do not come from a single source, many sources provide those data. Many companies handle everyday data from their clients, and they collect them, often anonymizing the source. As in the example of Walmart, there are many more companies having this volume of data every day. Many of these companies incorporate on-line websites, and thus, as we visit these website, information is being collected about our on-line behavior. Other sensor data is also being collected, not only via our smartphones, but also dedicated sensors established in many cities for different purposes, primarily for monitoring of the environment, i.e., air and water quality, etc.

The collection of data from the larger community, whether from websites or sensors, can be defined Crowd-Sourcing. Crowdsourcing is the participatory data collection for some purpose, involving geodata being provided by a group of people, a *crowd*. The term has become popular recently and found important application in diverse fields.

When crowdsourcing is associated with the geography, it is referred to as *neogeography*. M. Goodchild gives us a good definition of it:

In other words, the old geography involves a prescribed role/interaction between the four main components, namely the audience, the information, the presenter and the subject, which are common to most standard practises of learning. In NeoGeography, there are, however, no such boundaries on roles, ownership and interactions of these four components [4]

Neogeography and crowdsourcing are possible thanks to the development of the web and related technologies. Social networking, for example, is a key factor for the collecting large amounts of data from individuals. Every day more information is being transferred from even more places throughout the world and stored in exponentially increasing databases. Then are typically integrated into the cloud, and from there could potentially be accessible to anyone.

Thanks to GPS, this information is typically georeferenced, with latitude and longitude information embedded. Take for example the use of the mobile devices and cameras, they can collect georeferenced pictures and make them available for sharing. Flickr² is a good sample of a website with an extensive database of georeferenced pictures.

We should point out that there are mainly two kinds of crowdsourcing, active and passive. Active crowdsourcing is where the people collect the information by volunteering specific information for the purpose of helping in a project.

² <https://www.flickr.com/>

OpenStreetMap is a sample of volunteers gathering information to improve the quality of that map, for the benefit of the world community. Passive crowdsourcing is mostly done through social networks. This is where people may not know they are contributing information. In this case, the data collection may be anonymizing the source, while sifting as much information as might be useful for a business purpose. One concern might be the quality of this data. The quality could be a problem due to the sensitivity of the sensors used, and large potential for many other types of errors occurring given the indiscriminate data collection over a large quantity of data. Thus, it is necessary in some cases to evaluate the collected information and perform careful quality analyses, before relying on the data.

1.2.2 Data Storing

The necessity of storing data is one of the first three needs when handling a huge amount of data. The solutions that are available mainly depend on the kind of data we want to store. A high-level approach used by Google can be found in “Bigtable: A Distributed Storage System for Structured Data” [5]. In the case of geo-data we know that we have datasets with several dimensions, starting with the geo-location information, often represented by latitude and longitude, giving a set of values that can identify the origin of any data point. In the case of multidimensional data, the solutions adopted for less-dimensional data are not feasible.

The place where data are stored is also a really important to take into consideration. In particular, considering the Cloud as a solution could be an interesting way to manage those data. As regarding the system that can be used, most of them are able to run in the Cloud. Some samples systems are the OLAP - On-Line Analytical

Processing - data cubes, used already in many projects. In “Discovery-driven exploration of OLAP data cubes” [6] we see some examples of the possibilities available with these cubes, in term of storing and furtherly analysis. As we have seen the OLAP systems provides some processing tools, some more are explained in “Range queries in OLAP data cubes” [7] where the authors explain how is possible to query a data cube and what are the advantages in term of performances compared to other systems.

HEAD IN THE CLOUDS

In cloud computing, large data sets are processed on remote Internet servers, rather than on researchers' local computers.

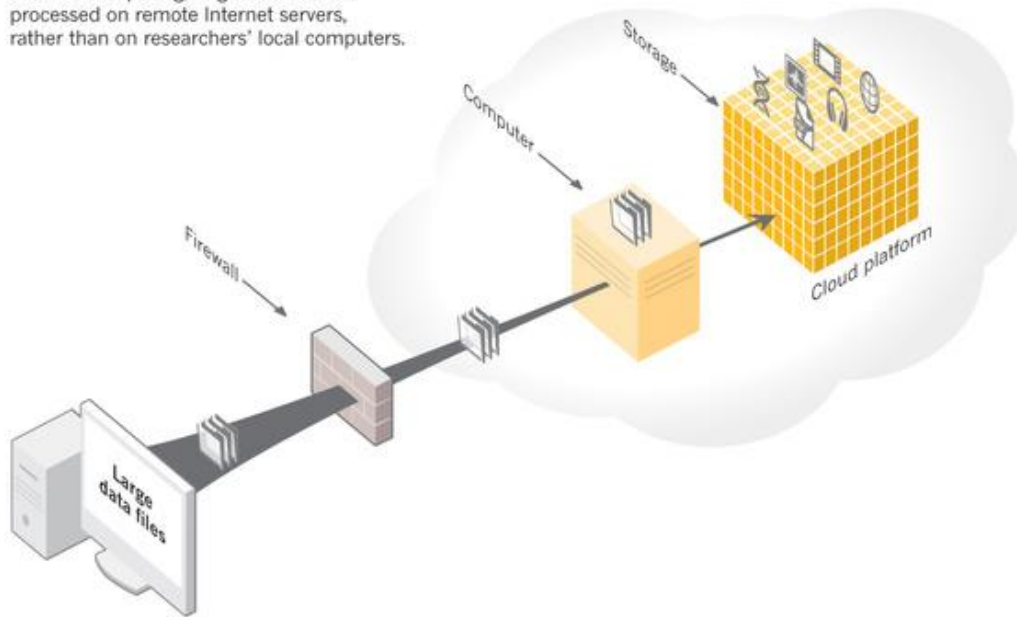


Figure 1 – Cloud data Storage [8]

1.2.3 Data Visualization

Data Visualization is something important in every day's life. The visualization of data gives many advantages. The area in which the visualization is useful goes from 3D graphics to entertainment, and much more. The visualization consists of extracting data, of any type, and picture them in 2D or 3D representation.

The visualization of multidimensional data is a difficult challenge faced by many scholars during the time. To visualize multidimensional data several techniques have been used, some techniques have been invented and still nowadays new approaches are growing and spreading continuously.

To present big amount of two-dimensional information, exist several well-known techniques, used in the past in many studies, and that resulted being productive. Some examples are star plots, scatter plot matrices and star glyphs.

These techniques though are not often used to visualize multidimensional geo-data, because the geo-data have the peculiarity to contain information relatives to geographical coordinates, and thus they belong to the real world.

Although two-dimensional techniques are effective on geo data, the need for a spatial representation is fundamental. By spatial representation, we refer to a visualization into a virtual globe, which would be the most appropriate one.

“Spatial representation allows the use of recognition mechanisms that are built into the visual perception system and allow very rapid recognition. Detection of spatial patterns and groupings is hardwired into the human visual system”. [9]

Referring to spatial representation the introduction of the third dimension allows users to perceive the data, viewing a representation as close as the real one. This system permits to understand the information quickly, especially for the amateur users.

The need to observe and inspect the datasets and their variables is not restricted to a limited category of users; indeed, the occasional users are not experts in Geographic Information Sciences, and a straightforward representation can certainly be more efficient than an obscure one.

The techniques mentioned before can be surely adapted in different ways to have a geographical, three-dimensional, representation of the data but another model has to be used.

Visualizing data in 3 dimensions ease the apprehension of a dataset. To visualize data in more than two dimensions, we need a 3D environment. Several solutions are based on Windows and Mac OS application while less for the web. In general, visualizing objects from the real world gives us a 360° panoramic about an object, plus, might be possible to view the data and have more evidence rather than the ones we can observe from the outside.

The visualization of 3D objects is done from different techniques than collected data, not having three dimensions. Some used techniques are the *splatting* and *textured splats*; that consist on a simulation the surface of an object with small pieces of 2D surfaces, placed in three dimensions. With the past of the years and the increase of the computational power of the computers new techniques based on voxels spread to visualize the objects in an easier way to understand.

Talking about data visualization, thus phenomenal data, fewer techniques are available. It is fundamental to have a 3D representation because it shows to non-expert people an accessible illustration, and simple to perceive. An example, taking into consideration the time, are the time slices of 3D data used in archaeology.

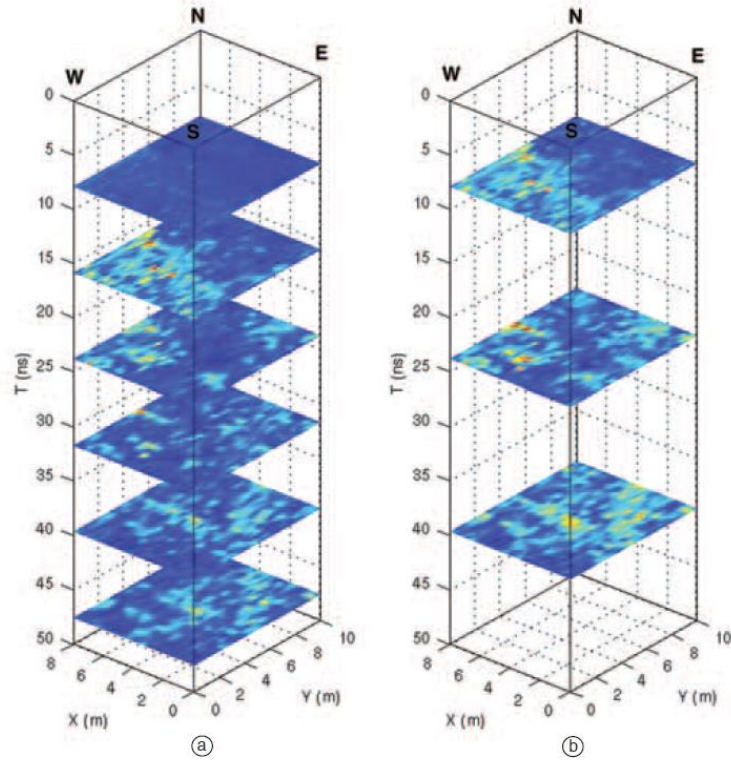


Figure 2 – time slices of a 3D representation in archaeology [10]

It consists of a mixed representation of 2D and 3D data where some slices of a surface are represented and on the z-axis the time is shown. While talking about more complex structures the Cone Tree technique allows the visualization of hierarchical information.

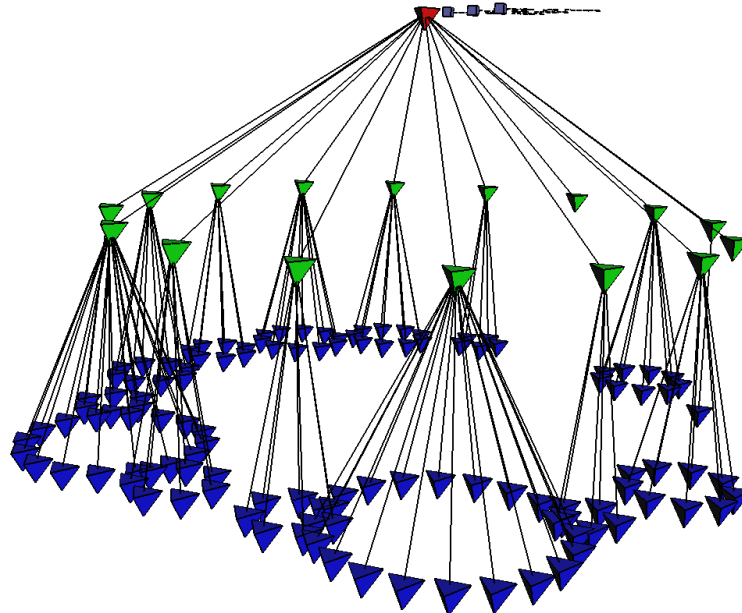


Figure 3 – Cone Tree representation of data [11]

It consists in a top-down visualization, showing a 3D graph with a parent node, and some leaves going down in the z-axis, that might have, in turn, other leaves. As pictured in the image below.

1.2.4 Data Analysis

As regards for the analysis of the data, several techniques existed even before the advantages in computer science and the storing in the databases.

The analysis of multidimensional data takes into consideration many relationships among the dimensions, each of them represented by a variable. Once more a feasible solution consists of tools available in the Cloud, in the case we want to store our data in a Cloud platform. As we mentioned, some analytical tools are provided by OLAP systems; we can see how to use them in “Providing OLAP (on-line Analytical Processing) to User-analysts: An IT Mandate” [12]. Another solution consists of having some of these data in our machines, and with some high-level software analyze them, in this case, many technologies are explained in the book: “Big Data Analysis” [13].

1.3 Geographic data over the web

One approach of using geographic data is the web. Although the possibility of showing and interacting with this information on the internet was not always something available, because the technologies to achieve so were developed recently. When talking about web technologies, we refer mainly to HTML – Hypertext Markup Language – and JavaScript. HTML is a pseudo-language to style a web page, inserting some markups. The pure HTML, at least until HTML5, spread around 2014 [14], did not permit users to interact with the technologies of the geographic world. JavaScript is a web scripting language, oriented to object and events of a web page. JavaScript comes from the ES – ECMAScript – language, standardized from ECMA International. Several versions have been available for the web but only from 2009, with the version 5, we have some features to create web applications. Such as the include of the JSON format – JavaScript Object Notation – which is well spread in the geographic information system world as well.

The first approach to technology to increase the functionalities of the web were the Java Applets [15]. A Java Applet is a program, written in Java language that can be visualized on a web page, after installing the proper Java plugin. It is a mix of a standalone program and a pure web application.

The mentioned technologies all run on the client side, so in the web browser of any users. However, to extend the functionalities of a web application there are also server-side technologies such as *PHP*, *J2EE* and *.Net* that provides several additional features approaching the geographic information to the internet.

Merging client and server side to create a web application we can have some examples of WebGIS, which is the acronym for the geographic information system published on the web.

The firsts WebGIS were although developed using old technologies and not including the latest versions of JavaScript or the new server technologies. Some examples were

static maps with marker and few others information.

New solutions, such as Google Maps³ implement modern technologies and spread all around the web in the last few years. In the same way as Google, other companies developed their software using client and server technologies to have WebGIS. Other examples could be MapQuest⁴ that is a service to provide information within a city using a map. In the application, there are more than a few information available, such as real-time traffic, or shops location and so on. Another interesting solution is OpenStreetMap⁵; it is a WebGIS that uses an openly licensed map, created by volunteers, and updated daily thanks to collected information by people using it.

1.4 Our Solution for Big Geo Data

To create our system, we wanted to resolve all the problems that could arise when creating an application that interacts with Big Geo Data. Thus, our application had to Store, Visualize and Analyze the data, being in a web environment. To achieve this aim we started from the storage of big data. We analyzed different solutions, and we found out that the most efficient for Geographic datasets is to use OLAP data cubes. Thus we used Rasdaman, a raster data manager. After storing the data, we decided to show them using a virtual globe in a client-side application. The application running on the client had to be fully autonomous without the need for a server to work. Although when we wanted to retrieve data from the server it needed a connection to it. The last step, the analysis of the data had to be linked to the visualization. For this reason, we integrated some statistical analysis tool in the virtual globe.

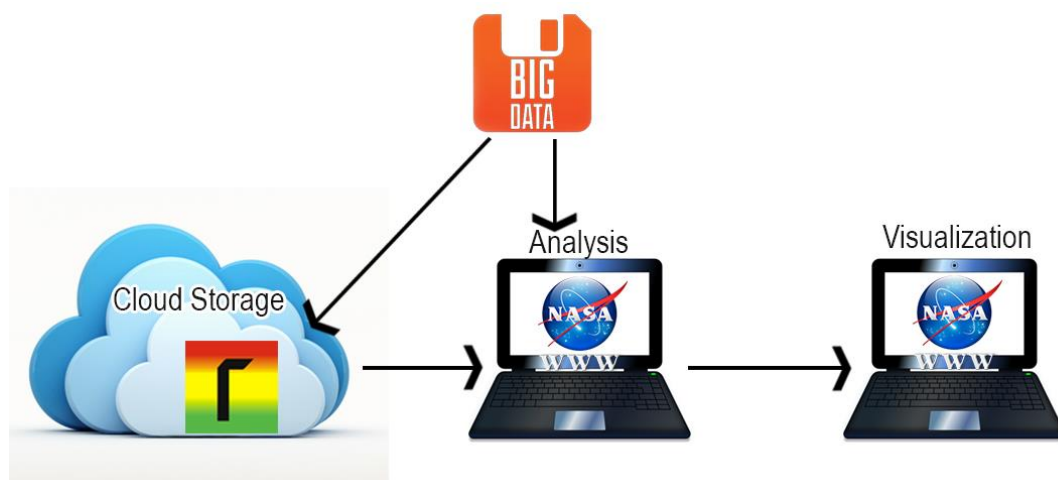


Figure 4 - Big Geo Data Architecture

³ maps.google.it/

⁴ <https://www.mapquest.com/>

⁵ <https://www.openstreetmap.org/>

1.5 Related Work

Not many applications to show n-dimensional environmental variables exists. Especially when we take in consideration web technologies.

Regarding general data visualization and not focusing on environmental variables, few solutions are available. Some are explained from C. G. Healey in “Effective Visualization of Large Multidimensional Datasets” [16].

Using virtual globes there are currently no web applications that allow users to show a custom dataset with more than three dimensions in a single view.

However, a similar goal has been achieved using NASA WorldWind, the Java version, so not available in a pure web application.

The solution is explained in the article “Environmental Space and Time Web Analyzer.” [17]

The aim of this paper is to show a reproduction of the application implemented by M. A. Brovelli and G. Zamboni with the Environmental Space and Time Web Analyzer in a browser compatible Virtual Globe - NASA Web WorldWind - and improve the previous product with other features.

Chapter 2

Virtual Globes

2.1 Available Virtual Globes

Several virtual globes are available, but there are several differences among them. We have analyzed many of them, highlighting some technical characteristics and their features, listing the OGC (Open Geospatial Consortium) standards. The points that we decided to focus are: the typology of the virtual globe, license, company running the project and introduction year.

Regarding the technical side, we also listed the technologies used in the development, the platforms for which they are available, the OGC standards available, their current state of development and the languages of the API (Application Programming Interface).

Name	Cesium	ArcGIS Explorer	Google Earth	WorldWind	Web WorldWind	SOS Explorer
Producer	AGI	ESRI	Google	NASA	NASA	NOAA
Int. Year	2011	2006	2004	2003	2015	2015
License	Open Source & Commercial	Freeware	Freeware	Open Source	Open Source	Freeware & Commercial
Typology	Web	Desktop	Desktop	Desktop	Web	Desktop
Language	JavaScript	.NET	/	Java	JavaScript	/

Table 1 – Technical characteristics of main Virtual Globes

2.1.1 AGI Cesium

AGI Cesium⁶ is an open-source JavaScript library for world-class 3D globes. It was founded by Analytical Graphics, Inc. in 2011. It is free and uses the license Apache 2.0. Cesium runs on any modern browser since it is developed using modern technologies: WebGL, JavaScript, and HTML5. Cesium implements many open 3D geospatial formats, some of them have been created from the Cesium team and are available only on Cesium. Two examples are the CZML, the Cesium Language, made by a JSON schema for describing scenes animated in time, and glTF which is a transmission format for WebGL engines.

Among the features, it implements a visualization of high-resolution terrains, layer imagery such as WMS, TMS and WMTS and vector formats, such as KML (Keyhole Markup Language), GeoJSON, and TopoJSON. Within Cesium it is possible to create

⁶ <https://cesiumjs.org/>

data-driven scenes with CZML, draw geometries as polylines, billboards, labels and so on. The CZML is defined as a language called “Cesium Language”. It is made by a JSON schema which describes data over time. Thanks to this language it is possible to implement lines, points, models and other graphical primitives, specifying their variation during the time. This makes Cesium easy to use thanks to CZML because any user having such file could import it into Cesium and analyze it during the time. Cesium is fully navigable from any device and also implements a 2D visualization. Some other features are available thanks to some external plugins, and some other DEMs are available to be visualized in it. Some features instead are available only within the Cesium Pro version, and these are not open-source.

2.1.2 ESRI ArcGIS Explorer Desktop

ESRI ArcGIS Explorer Desktop⁷ is a virtual globe developed by ArcGIS, allowing exploration, visualization and sharing of GIS information. It succeeded ArcExplorer that is now deprecated. ArcGIS Explorer is closed-source but free of use. It is available only on Windows OS. The virtual globe allows inserting of live data and services thanks to a built-in presentation tool. Data can be imported using the more common formats such as shapefiles, KML, and KMZ, GPX and raster imagery. Thanks to some integrated tools, it is also possible to insert photos and videos inside the globe and also embed them in popup windows. It supports some OGC standards, such as WMS (Web Map Service) and GeorSS. It also allows the user to switch between 3D and 2D mode, with data visualized in both ways. There are some base maps available inside: World Imagery, World Streets, and World Topographic map. Moreover, it supports a spatial analysis such as visibility, modeling and proximity research.

2.1.3 Google Earth

Google Earth⁸ has been one of the first virtual globes; it was created by Keyhole, Inc. as a CIA– Central Intelligence Agency - funded project and then acquired by Google in 2004. It is a well-established virtual globe and is available for Windows, Linux, and Mac OS. A specific version has also been developed for mobiles OS and runs on Android and iOS. It uses a freeware license, and in past had a proprietary one for the Pro version, that is now provided as freeware as well.

It is closed source software and all the images created using it are copyrighted from Google Earth. Thus, these images cannot be used for commercial purposes. Among the features there is support of 3D imagery for many models such as 3D trees, photorealistic buildings with 3D areas being automatically generated. Many users can contribute to creating 3D building and upload them to a common warehouse. Google Earth permits a Street View, which is a 360-degree panoramic view of the street in many cities. It also can display weather data such as clouds, radar and general conditions, provided by external services. The Pro version, available for free, can comprehend more features than the basic version, such as the possibility to print high-resolution images, import georeferenced images, and access layers that have information about traffic and demographic data. It also includes some measurement tools for lines, polygons and even circular data. Thanks to a CSV (Comma Separated Values) reader there is also the ability to read and import external databases. There

⁷ <http://www.esri.com/software/arcgis/explorer>

⁸ <https://www.google.com/earth/index.html>

was an API for developers to interact with the globe programmatically, but it was deprecated in December 2014.

2.1.4 NASA World Wind

NASA WorldWind was created in 2003 and released under the NASA Open Source Agreement license in 2004. The first versions up to 1.4 were available only for Windows and the developers used .NET as developing language. In 2006 it switched to Java and became available for every OS. Right from the start, it was an important point of reference for the GIS community. NASA WorldWind provides an API to customize the globe and create any number of applications. It can be used to represent not only the Earth but also the Moon, Mars and other planets. Apart from imagery, it supports the importing of several types of 3d objects, a large collection of geometric and geographic shapes and many graphical capabilities. It supports various data formats like JPG, PNG, GeoTIFF and others as imagery. NITF RPF, DTED (Digital Terrain Elevation Data) which are government formats. But also many GIS formats are available such as shapefiles, KML, GML, GeoJSON, GPX and much more. It has many analysis capabilities, useful for the understanding of the visualized data. There are measurements tools, available both for the geometries or following the terrain. There are tools to measure the intersections among objects and interacts with shapes. Since it is developed in Java, it is available as a Java Application, but also as a Java Applet and can be included in web pages as a Java Web Start Application, thus visualized on the web, with Java installed on the PC. The documentation provides an API and examples to customize it for implementing any number of new features. It can be fully configured and customized for the needs of the users. The API is well documented and there is also a community support forum, which is very active, and many users interact every day.

2.1.5 NASA Web WorldWind

NASA Web WorldWind⁹ is a 3D virtual globe API for HTML5 and JavaScript. It is a library and API rather than a stand-alone application. This enables it to be included in any web page or web application as a component. Thus is available on any OS which implements a web browser. It has a WebGL internal core but provides a JavaScript interface for operating with it. Web WorldWind is open source, and the repository is available on GitHub¹⁰. It uses the NASA Open Source Agreement (NOSA) 1.3 license. Web WorldWind is designed to be easily extensible, thanks to an interface on top of JavaScript and WebGL. Due to this geospatial approach used in the Web WorldWind framework, any user with some experience in JavaScript can create components for it.

Even though WebWorldWind is at the dawn of its development life cycle, different organizations around the world are using it in many applications, such as Smart Cities, Terrain and City Visualization, Vehicle tracking, Geospatial data analysis and more. Starting from 2016, a new collaboration with ESA¹¹ - European Space Agency - and Thales¹² is providing new features to Web WorldWind to implements many

⁹ <https://webworldwind.org>

¹⁰ <https://github.com/NASAWorldWind/WebWorldWind>

¹¹ <http://www.esa.int/ESA>

¹² <https://www.thalesgroup.com/en>

functionalities. It supports 2D map mode and several different projections are available, in this way it can also be visualized as a 2D WebGIS. It permits anyone to visualize terrain and geographic data from any public or private source. Among the features, there is built-in high-resolution imagery and support to show more thanks to REST services and WMS. Regarding the graphical capabilities, there are several geometric shapes as Paths, Polygons, Circle and others. It supports many OGC standards such as WMS, WMTS, KML, and more are continually being implemented, thanks to the collaboration with ESA and Thales.

2.1.6 SOS Explorer

SOS Explorer¹³ (SOSx) is a recent virtual globe developed by NOAA. It is a similar version of what is used on Science On a Sphere (SOS) that is a project in which a real sphere is available in a big room, and through some projectors, the images representing the virtual globe are illustrated on a sphere. It is a freeware software and is available only on Windows and Mac OS. It comes in two versions, SOS Explorer Lite, which is an introductory version and is available for free to download, while the other version SOSx is available by paying. The free version allows choosing a dataset among the 16 included. It permits visualization and analysis of complex data but does not provide an API for interacting with the globe.

Name:	Cesium	ArcGIS Explorer	Google Earth	WorldWind	Web WorldWind	SOS Explorer
API	Yes	Yes	Deprecated	Yes	Yes	No
Documentation	Good	Good	Good	Good	Good	Poor
KML	Yes	Yes	Yes	Yes	Yes	Yes
WCS	No	Yes	Yes	Yes	Yes	No
WMS	Yes	Yes	Yes	Yes	Yes	Limited
WMTS	Yes	No	No	Yes	Yes	No
WFS	No	No	Yes	Yes	No	Limited
NetCDF	External	No	No	External	External	No
GeoTIFF	Yes	Yes	Yes	Yes	Yes	No

Table 2 – Features of main Virtual Globes

2.2 Modern Virtual Globes

We can think of two core categories to differentiate Virtual Globes, the ones developed before the Web 2.0 – defined as such from Tim O’Reilly in 2005 [18]– and the modern Virtual Globes, the ones available for the web, running in a simple web browser. Two notable examples of modern Virtual Globes are NASA Web WorldWind and AGI Cesium, which run in any browser thanks to the support of JavaScript and HTML5 technologies. It means these online tools are cross-platform and can run on any device (PC, Mac, Smartphone, and Tablet) without the need of plugins or extensions, and without having to install any additional software.

¹³ http://sos.noaa.gov/SOS_Explorer/

What is especially interesting about these new Virtual Globes is the possibility to customize them without advanced programming knowledge skills. This means that almost any user could create their own Virtual Globe application and easily share it with everyone via the web. This increases the opportunity to create Web-GIS tools using common programming languages and lets users interact with them easily. Dealing with three-dimensional data has become much more manageable, thanks to these web-based virtual globes.

Web WorldWind, along with Cesium and their APIs, offers feasible ways to display a wide category of data. There are several differences between these two virtual globes.

1. Both frameworks are designed to display data available via web standards, but Web WorldWind also support formats used by the United States Department of Defense. Moreover, it also supports other formats and the ability to implement them in the framework without requiring expert coding skills.

2. Web WorldWind uses a geographic interface for configuring the objects in the virtual world, while Cesium's primary interface is 3D-centric versus geographic. Apart from the primary interface, a second interface using geographic coordinates is available but is not as good as the primary one.

3. Web WorldWind offers support for different elevation data, using WCS (Web Coverage Service) and DTED data sources. In the client, at run time, different data sources are used together and combined in the globe. Instead, Cesium supports proprietary data but allows using ESRI ArcGIS elevation data sources.

4. Web WorldWind is designed to enable users to extend its functionalities by adding new components written in JavaScript and WebGL. On the other hand, Cesium is made of two interfaces, a high level one to interact with JavaScript and another one to interact with WebGL, but to extend it, it is necessary to learn both.

5. Cesium offers some functionalities, which are available only in the "Cesium Pro" version, available a proprietary commercial version. Web WorldWind offers all functionalities free of charge and does not have a "Pro" version.

While Cesium is focusing towards supporting computer graphic features, Web WorldWind targets to maintain practical features to work with the standard geospatial data formats. Thus, Cesium can be considered mainly a graphical tool to visualize 3D data rather than being a geospatial visualization tool.

Thanks to the geospatial approach used in the Web WorldWind framework, any user with some experience in JavaScript can create components for it while the extensibility of Cesium requires a broad knowledge of WebGL.

Even though Web WorldWind is still at the dawn of its development life cycle, different organizations around the world are using it in many applications, such as Smart Cities, Terrain and City Visualization, Vehicle tracking, Geospatial data analysis and more.

2.3 Our choice: NASA Web WorldWind

We wanted to develop a responsive web application, and this led us to choose a modern virtual globe running on the internet. We aimed to develop advanced functionalities, and so we needed the complete freedom of working with an open platform. Also, we wanted an already existing platform since we had decided not to develop a Virtual Globe from scratch.

The options available were AGI Cesium and NASA Web WorldWind. Knowing all the advantages and disadvantages of both platforms, we focused on Web WorldWind. Our decision was mainly based on the target for the application. Since we had to represent complex structures to visualize the data, either framework could be used, but the need for a strong connection with geospatial data formats and the geospatial environment brought us to the choice of Web WorldWind. Moreover, we already using Web WorldWind in other projects, and so this choice facilitated our ability to develop a complex application using a technology we already familiar with.

2.4 Voxel Model within Virtual Globes

Being in a virtual 3D environment is fundamental to show multidimensional variables. An appropriate model for the 3D environment is the voxel model; it consists of an array of voxels – akin to cubes – to show data with a volume representation. Voxels have been used in many study areas. In the article “Computer Visualization of Three-Dimensional Image Data Using IMOD” [19], voxels are used to view tomograms. For modeling objects and structures, derived from volumetric data, a voxel model is proposed in “Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects” [20]

Several approaches based on voxels have been adopted to work with environmental data, in “Virtual globes for 4D environmental analysis” [21]. In this paper, using NASA WorldWind Java, there are several examples on how to handle environmental variables in multiple dimensions. An example is a 4D visualization of Lake Como water temperature at different depths, represented with different colors. Then there is the utilization of energy among the buildings in Olbia City thematizing the buildings by the color, and extruding the heights.

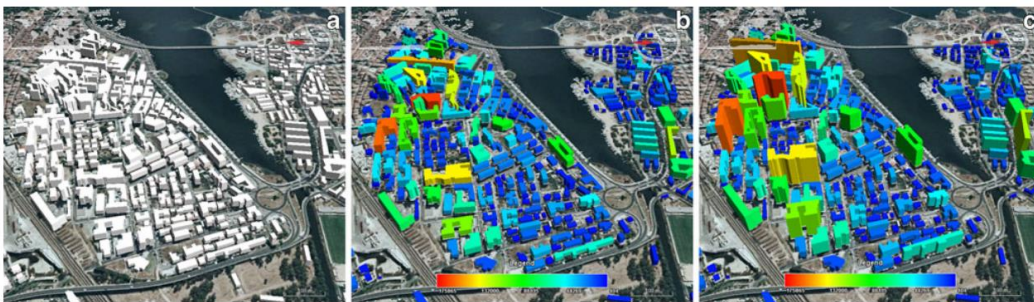


Figure 5 – Color thematization of different buildings in Olbia [21]

Another example developed in this paper is the implementation of the EST-WA (Environment Space and Time Web Analyzer) to manage environmental variables. In this paper a voxel is used to represent 4D variables, in this case, having more than three dimensions, it is a doxel – dynamic voxel –.

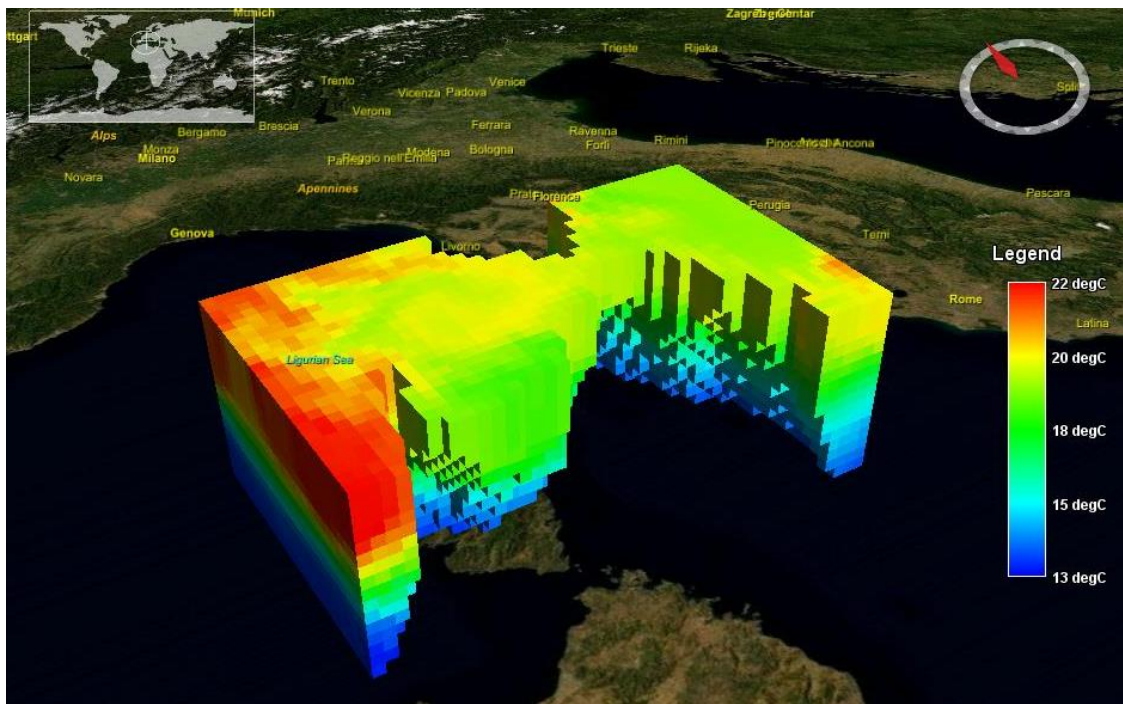


Figure 6 – Voxel representation of water temperature using EST-WA [21]

“A generalization of a voxel is the doxel or dynamic voxel. This is used in the case of a 4D dataset, for example, an image sequence that represents 3D space together with another dimension such as the time. Although storage and manipulation of such data require large amounts of memory, it allows the representation and analysis of spacetime systems” [22].

A 3D array of voxels is created from a NetCDF file and shown on the globe. Moreover, to view the internal voxels of the 3D object, the volume can be further sectioned with planes orthogonal each other, to focus on the voxels within the plane

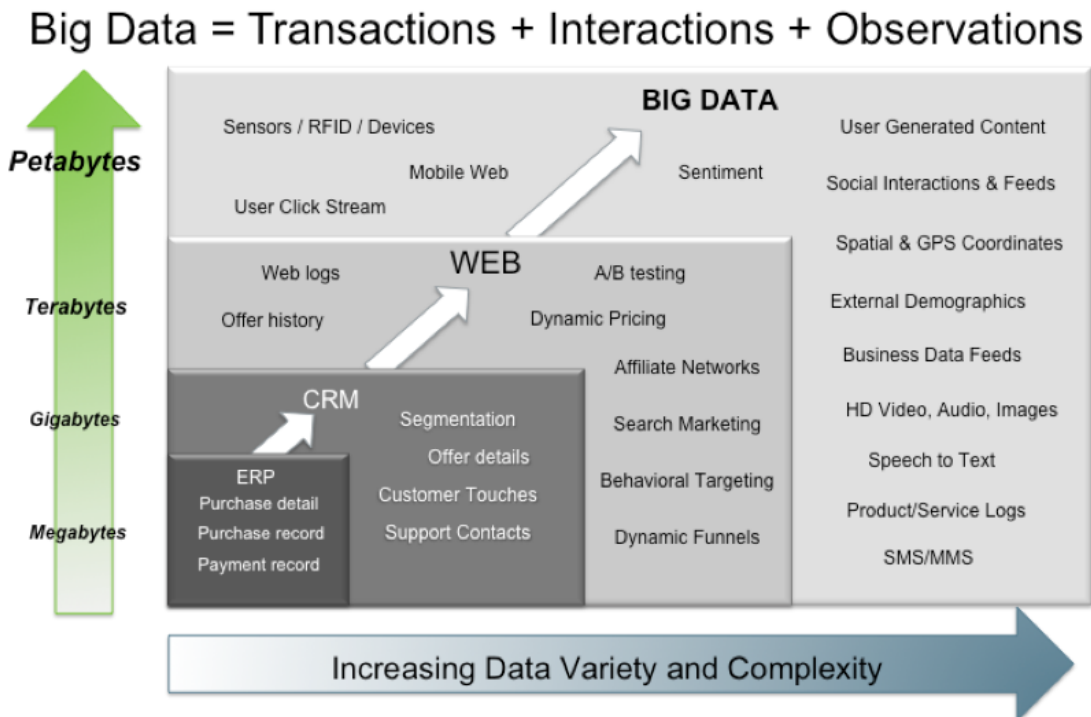
Chapter 3

Storing Solution for Big Geo Data

3.1 RDBMS: An Old Inefficient Solution

When we have to face Big Data, one crucial point is storage of that large amount of data. As we already mentioned, one solution is to use a Database Management System. Although to find the correct DBMS (Database Management Systems) for our data is not always easy, given that many aspects should be considered. The old solutions, thus the Databases used before the era of Big Data, often are not suitable for our case, in terms of performances and structure.

The most common DBMS were Relational DBMS, created specifically to handle relations among the data with a fixed structure. However, the need for a relation increases the complexity of the database structure and tends to decrease performance. The decrease of performances in a Database, increases exponentially when having to handle large amounts of data. Big Data can range from Terabytes to Petabytes. So, we need to carefully consider the optimal solution for storage.



Source: Contents of above graphic created in partnership with Teradata, Inc.

Figure 7 - Big Data dimension and variety [23]

There are certainly conditions when a relational database for storing Big Data is unnecessary. Three criteria that describe Big Data help to explain this issue, and are referred to as the three V's: Volume, Velocity and Variety [24]. Having large volumes of relational data makes it difficult for those relations to be managed. Given large volumes of data it is often not even possible to relate many kinds of data with each other, thus the classic Relational DBMS cannot cope with Big Data efficiently, as compared with more recent solutions developed specifically for them.

3.2 NoSQL DBMS

The term “NoSQL” stands for “Not only SQL” and refers to a new branch of DBMS that exploits different structures rather than the classic data tables. Among the companies implementing NoSQL DBMS are big names, such as Google, Facebook, and Amazon. The need for NoSQL Databases emerges from the Web 2.0, to address Big Data and to help the users to store them in an efficient way, rather than using technologies established in a different era, for different data and purposes.

The advantages of a NoSQL database are many in comparison with the standard relational DBMS. First of all, the performance is better, by not having the need to perform complex queries. Then it is the flexibility of the NoSQL database that plays a fundamental role, and since there is no need for a fixed structure, it is possible to have an unstructured database, making NoSQL quite suitable for Big Data. And another vital factor for Big Data management systems is scalability. In fact, databases that are scalable can accommodate immense amounts of data, without suffering from space limitations, and can be distributed among different nodes, in different servers and be extended easily.

When we refer to NoSQL Databases we also refer to OLTP – OnLine Transaction Processing – which is an online processing that facilitates the transactions inside an information system. All the NoSQL DBMS rely on this property.

The Relational DBMS were based on 4 key-characteristic called ACID (Atomicity, Consistency, Isolation, and Durability). However, we have to give up one of these to handle Big Data more powerfully, and that is the consistency. In fact, another acronym for the NoSQL was defined by Pritchett [25] and is BASE, which stands for Basically Available, Soft State and Eventually Consistent. We can see that we might lack the consistency property to gain more in terms of performances and scalability.

One interesting reason that helps to explain the performance increase comes from the fact that the types of queries being performed are different. Not having a relational database, the queries can be simpler and not involve joins among tables. Thus fewer operations are needed for these less intensive queries and thereby afford abundant advantages in querying Big Data.

Several Databases are available in the NoSQL field, each of them has some special characteristics and different approaches for structuring the data. We could differentiate these databases in 4 general areas: Key-Value stores, Document stores, Wide Columns stores, and Graph Databases.

3.2.1 Key-Value Stores

The Key-Value model, as the name anticipates, is made by a link between a key and a value. The value in this link can be slightly flexible, and thus can be a list, a payload or a set, depending on the database implementing it. These databases allow the classic operation of NoSQL databases of insertion, deletion and lookup.

The values can be stored both in Disk and RAM. The storage in RAM allows powerful performances and low latency on accessing the data. Some databases, such as Redis, store the most used value in RAM, and when the amount of data exceeds the limit of the available RAM, it automatically continues storing on the Disk.

The Databases implementing the Key-Value structures are often the most efficient ones since they implement low level commands and do not need to perform complex operations. Some samples of these databases are Dynamo, Redis and Project Voldemort.

3.2.2 Document Stores

The Document model is more complex than a simple Key-value because it implements a structure in which a document is stored for each entry in the database. Though there is a document, it does not need to follow a fixed schema, and is flexible, allowing different entries to have different structures. Some databases, such as MongoDB, even allows storage of a nested document inside a parent document. The kind of documents can vary depending on the database. Though some confusion can emerge when people refer to the word “Document”, even if it does look like a processed document, it might refer instead to a structured document, such as XML and JSON.

The databases always use an indexing systems that supports the querying on the entries without analyzing all the structure of each entry. In the document store, there can be more than one single index in a document to facilitate the lookup process. Some examples of Document Stores database are MongoDB, SimpleDB and CouchDB.

3.2.3 Wide Columns Store

The Wide Columns structure, as well known as Extensible Record Store, is a structure that might resemble a relational database. It is made by rows and columns, but only the rows are indexed. The internal structure is similar to the key-values, because the entries are made in the same way with a key and a corresponding value that could be a string, a list or a different structure. These databases differently from the relational one are grouped by columns. The number of columns and rows though are not fixed and in a same database we can have different rows with different columns. This structure can be partitioned both horizontally and vertically. Some famous examples are BigTable from Google, HBase and Cassandra. These DBMS are quite similar to classic relational ones, and are used often when porting a database from a SQL structure to a NoSQL one.

3.2.4 Graph Databases

The Graph Databases belong to the NoSQL databases, but rather than having their own fixed storage system, they implement one of the previously mentioned ones. The name comes from the structure that is defined as a graph. There are nodes, which are the points of references for all the entries, then edges, corresponding to the relationship among different nodes and finally properties, containing information about the nodes.

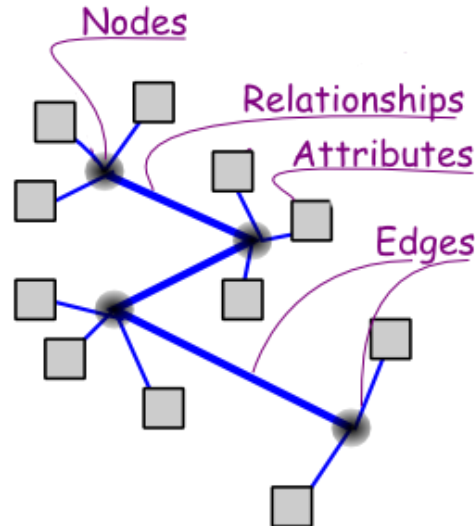


Figure 8 - Graph Databases Representation [26]

Querying a graph database is simpler than a normal relational database, because the relations are handled specifically with respect to the edges. The structure remains fully scalable, without any joins or a fixed schema. Each Graph database implements a different NoSQL structure, that can be a Key-Values, Documents or Wide Columns. Some sample are Neo4J, Arrango and OrientDB.

Listed below is a table with the database name, the typology of storing structure and the license type.

Name	Typology	License
Arrango	Graph	Apache
BigTable	Wide Columns	Proprietary
Cassandra	Document Stores	Apache
CouchDB	Document Stores	Apache
Dynamo	Key-Values	Internal
HBase	Wide Columns	Apache
MongoDB	Document Stores	GPL
Neo4J	Graph DB	GPL
Orient	Graph DB	Apache
Redis	Key-Values	BSD
SimpleDB	Document-Stores	Proprietary
Voldemort	Key-Values	None

Table 3 - Main NoSQL DBMS

3.3 OLAP: Multidimensional Analytic System

OLAP stands for OnLine Analytical Processing and is an approach used to answer multi-dimensional analytical queries [27]. There is a very close relation between OLAP and the databases management system. Since these databases are used to store data in a layer schema, the structure of these data can be organized following the OLAP system. OLAP techniques and tools to analyze and process Big Data can be described as follows:

OLAP operations include rollup (increasing the level of aggregation) and drill-down (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, slice_and_dice (selection and projection), and pivot (re-orienting the multidimensional view of data). [28]

The structure OLAP uses for multidimensional databases, the most common in case of Big Data is called OLAP Cubes. An OLAP cube, consist on a structure of data, organized in a way that performing queries is more efficient than a Relational DBMS.

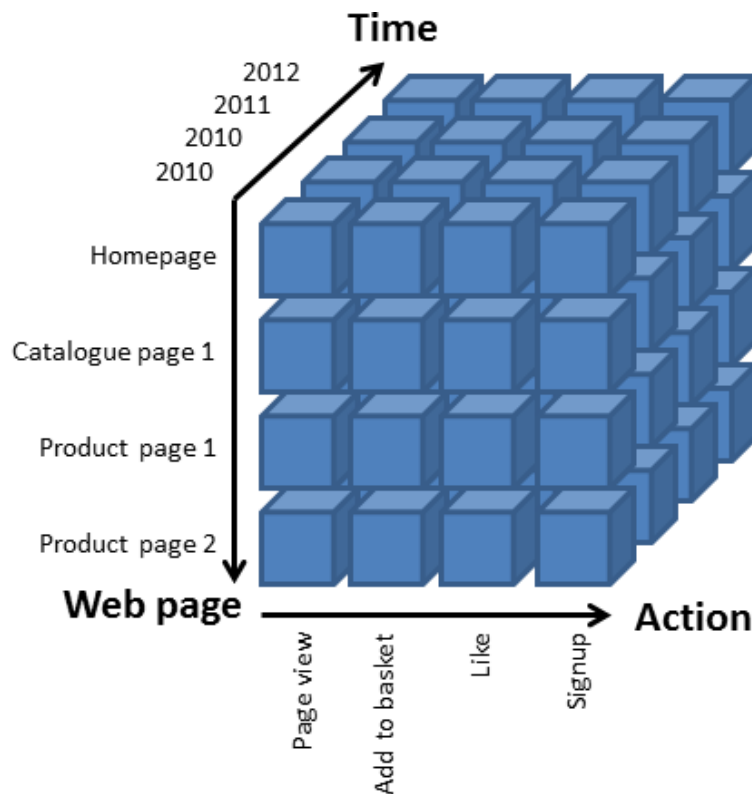


Table 4 - OLAP Cube Representation [29]

In each dimension of the cube there can be a different type of data. The cube, despite its name, is not restricted to three dimensions but can have several more. Thanks to this Cube structure, as named, the aggregation over several entries is easy, since the correspondence is only with the aggregation of a single column. Querying an OLAP cube gives us a Pivot Table, the report we obtain when querying specific data. This table can operate in an unrestricted number of dimensions, depending on the

available dimensions of our cube.

Thus, having a Multidimensional Discrete Data (MDD) solution for handling them is necessary. Only a few solutions are available that implement OLAP cube and handle MDD. A Sample is Apache Kylin¹⁴ whose architecture is described from the picture below.

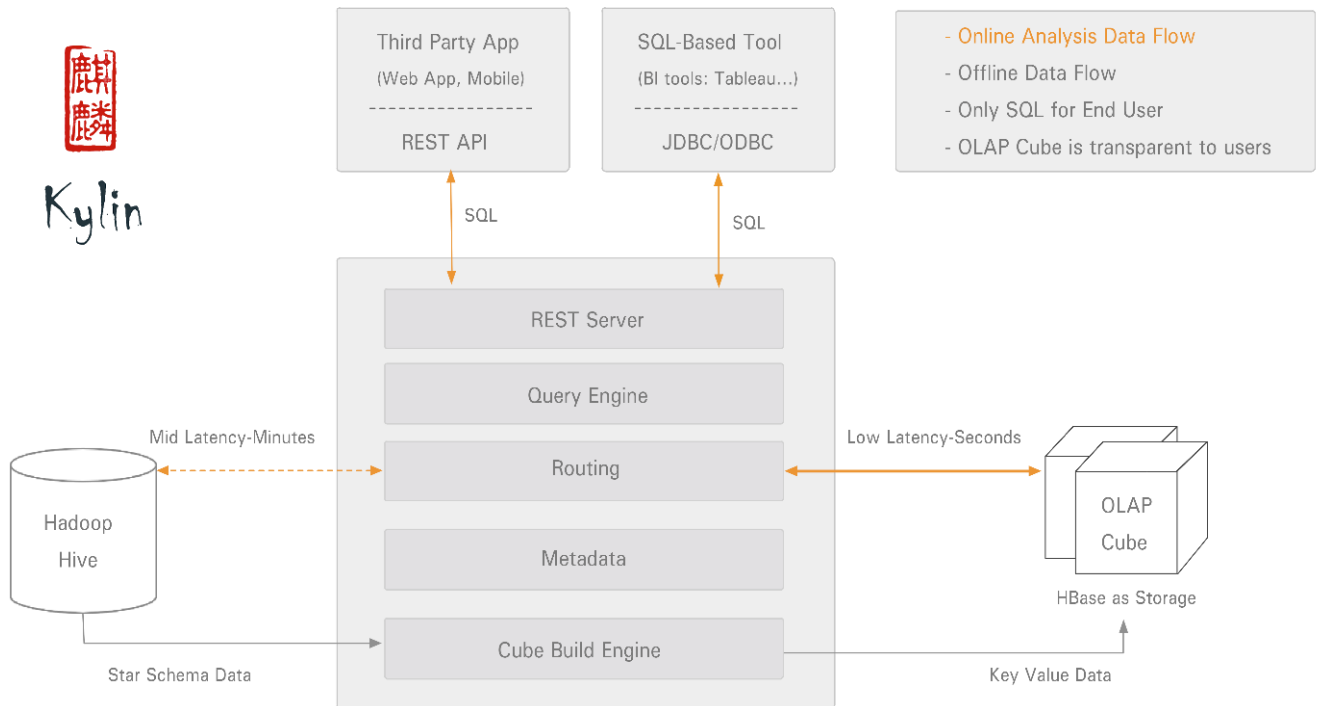


Figure 9 - Apache Kylin Architecture [30]

We can understand from the image that beside a DBMS, Hadoop Hive, there is a layer made of a REST server, providing a Query engine and other components to interact with the OLAP Cubes. This layer can be accessed by third-party applications, from a REST API via the web.

We have now seen how is possible to interact efficiently with MDD through the web and the systems used. However, we wish to focus on a specific kind of system to support spatial data. Even if Apache Kylin looks to be a feasible way for many kinds of data, a specific solution is more appropriate when treating spatial data in two or three-dimensions. A well-known solution for this is a system called Rasdaman.

¹⁴ <http://kylin.apache.org>

3.4 Rasdaman

Rasdaman¹⁵ stands for Raster Data Manager and is a solution for interacting with MDD thanks to classical database service in a domain independent way.

RasDaMan is a universal - i.e., domain-independent - array DBMS for multidimensional arrays of arbitrary size and structure. [31]

As we can see from the name, the Rasdaman system allows interaction with raster data, and thus images. Nevertheless, not only images are storable in the system, but also data in several dimensions, going from one dimension, like CSV files with a Key-Value, to two-dimensional images or even multidimensional GeoTIFFs. Thanks to this vast support for different kinds of data, it appears to be an optimal solution in the geospatial field.

Behind Rasdaman several NoSQL databases are available. In the picture below, we can understand further how Rasdaman interacts with the DBMS.

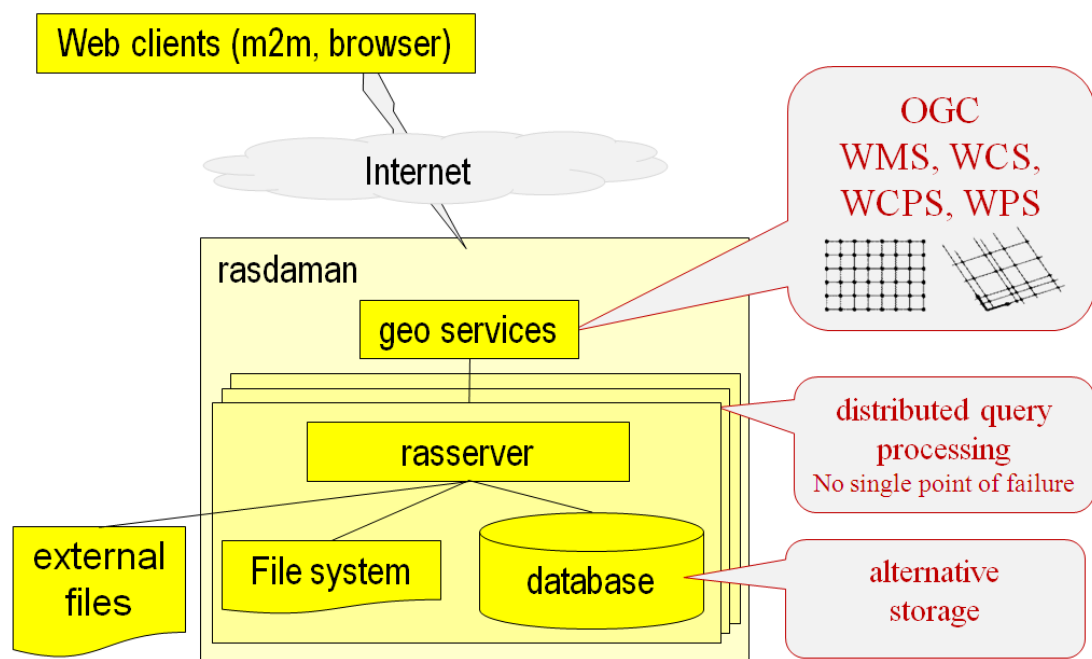


Figure 10 - Rasdaman Architecture [32]

Similarly, to Apache Kylin, Rasdaman takes care of handling all the files we want to store, but its structure wrapped inside of it and so it is also part of the database. However, a database is not mandatory because the system allows storage of the data directly in the file system.

¹⁵ <http://www.rasdaman.com>

Rasdaman gives to the end-users an interface to retrieve data in several ways. Everything can be accessed via a web client, processing the queries in different ways. It follows OGC standards and permits the interaction through some of them: WMS, WCS, WCPS (Web Coverage Processing Service) and WPS (Web Processing Service). We have seen that WCS is available, and this means we can retrieve just a portion of data, cutting an image specific to a bounding box and retrieve the exact data we want, without any of the unnecessary parts. Then, thanks to the WCPS and WPS, we can perform analysis and processing operation on the data, even before obtaining them. This means we do not need an external tool to perform this operation on our raster data, but allow Rasdaman to take care of everything, and provide us with the desired output.

Chapter 4

Application Development

4.1 Requirement Analysis

What we needed to achieve from this application was a simple way to browse through environmental variables, in multiple dimensions, and understand the data quickly. Below we list what our requirements were:

- **Dynamic Data:** The first goal to achieve was the implementation of a model to display several variables in multiple dimensions.
- **Data Interaction:** Having in mind our model, we wanted as well to give the users a way to interact with the data and retrieve statistical information from them. The visualization should have been also an interactive one, where users could set several parameters to create their view.
- **Customize View:** The model should have the capacity to show also customized view for different variables and being fully customizable on the graphical interface. With this idea in mind, we wanted to provide many customizations for the model, for the globe and the data.
- **Data Import:** Another important requirement was the importing system. With referring to it we wanted to give the opportunity to use different data sources and so do not use a specific format for the data.
- **Data Validation:** The last point was to prove that the system could work with same sample datasets and try to create simple use cases with those datasets.
- **Data Variance:** Moreover, it would be useful to analyze how those data are changing in time and get statistical information – such as the variance – among the variables and the datasets.

4.2 Application Audience and Potential Uses

We thought that the application we developed could have been the first point of reference when users need to visualize multi-dimensional data. Our audience so is very wide, since a wide category of users could use it for different purposes. All the uses of the application refer to a better understanding of the data thanks to a visual perception in a 3D environment and can be adopted in many study areas, where people need to understand the data inside a dataset. Thus the category of users that can get advantages from it could vary from studies in the telecommunication, to the

geological or environmental one. We will see furtherly some examples in the study case we created in this thesis.

4.3 Use Cases

The actions a user is supposed to do are:

- Upload a data set
- Show the dataset using the voxel model
- Show statistics for a group of voxels
- Compare two variables in a dataset on two different datasets
- Show statistics between two variables
- Filter the data in the dataset using different filters
- Browse the data through the time

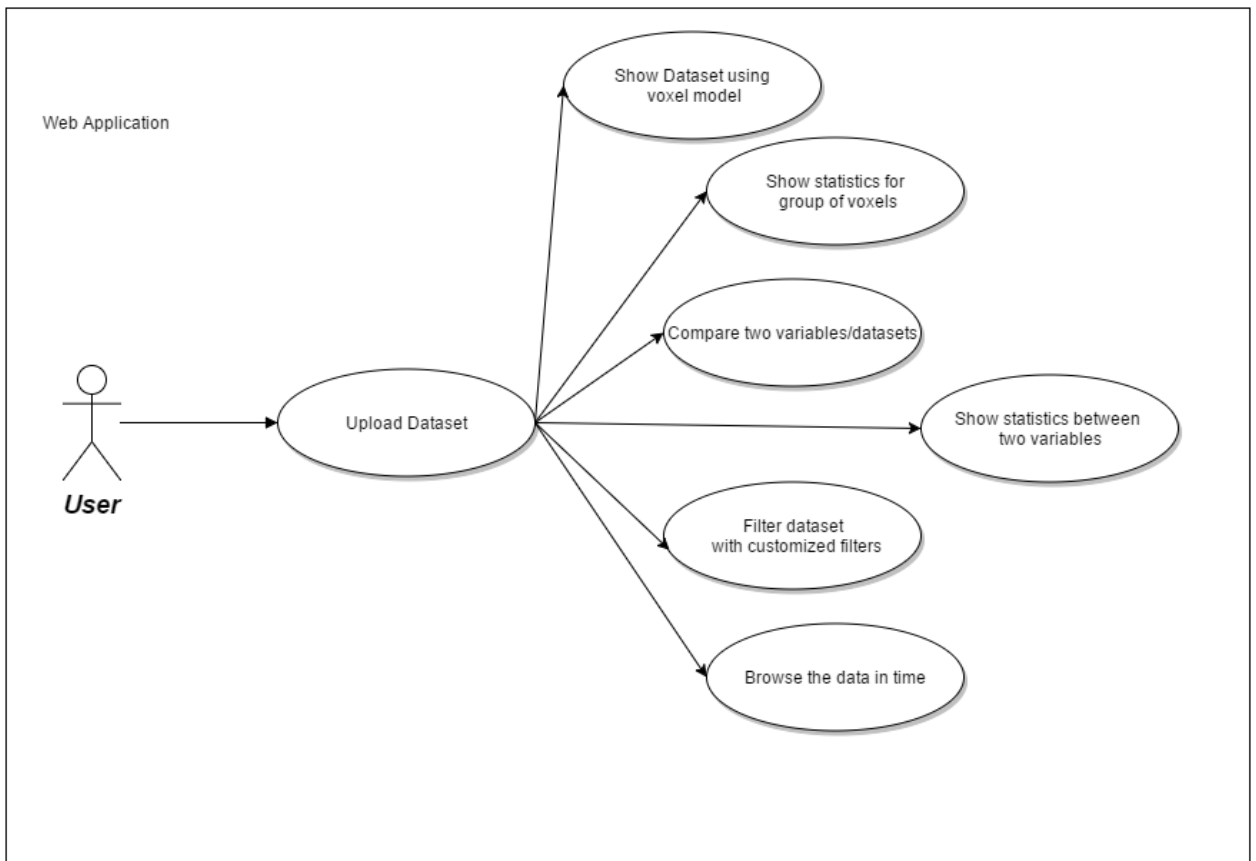


Figure 11 - Possible use cases of our application

4.4 Application Introduction

Taking into consideration all the requirements, the main goal of the application was to reproduce a similar work to the one achieved by the EST-WA from M.A. Brovelli and G. Zamboni [21], but using different technologies, considered more modern. We thus implemented a voxel model similar to EST-WA to show n-dimensional variables

in a 3D environment.

In this document, we use the terms “doxels” and “voxels” interchangeably, even if “voxel” refers to the geometric shape while “doxel” to the dynamic voxel to represent multi-dimensions.

In our model, we represented the fourth dimension by the color of the voxel. These doxels have a specific color belonging to a color range defined a priori, to represent a specific environmental variable. The physical character of the voxel/doxel could be used to describe any number of criteria. One idea could be to assign opacity as a variable to the voxel. The opacity could decrease with the more transparent doxel representing the minimum value. Since the voxel would fade to full transparency, representing the quantity of a variable The user could be offered different ways to represent the data that would seem most intuitive to their way of thinking.

A representation of a variable could be linked to the outline of the voxel, using a specific color or thickness for the boundary of the voxel. Although in this case, it may disorient the user’s perception of the doxels, having a small graphic element to show a variable, possibly creating confusions with the nearby doxels. Other techniques can be implemented to visualize various criteria, such as shading, different patterns and more. Despite the possibility to account for several variables simultaneously with different physical characteristics to the voxel, the user’s perception of a model having many variables might be difficult to understand quickly, as we can see in Figure 2 and Figure 3. So we have tried to limit the dimensions visualized –the number of variables – by limiting the user’s ability to customize only a few parameters of the data.

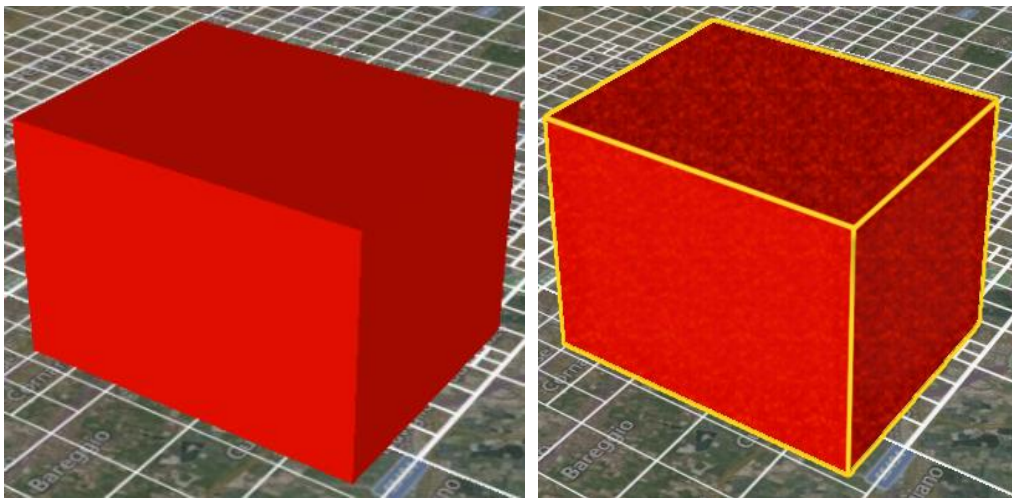


Figure 12 – Simple and complex representation of a Voxel

4.5 Application Architecture: Client Side

The architecture of the application is thought to be Object-Oriented. This kind of architecture is made of components that do not have strict dependencies on the other components; this means that any part of the application can be re-used. For instance, we can take into consideration the implementation of voxels: a voxel is an object made only by Web WorldWind functionalities, and anyone who needs to implement

a voxel model can easily copy the related file and import it in his or her application.

4.5.1 NASA Web WorldWind Framework

All the components available in a Web WorldWind application, as we can see in the following picture, are divided between internal component and external ones. The blocks in red are the outer ones; it means the developers can interact with them to create their own applications.

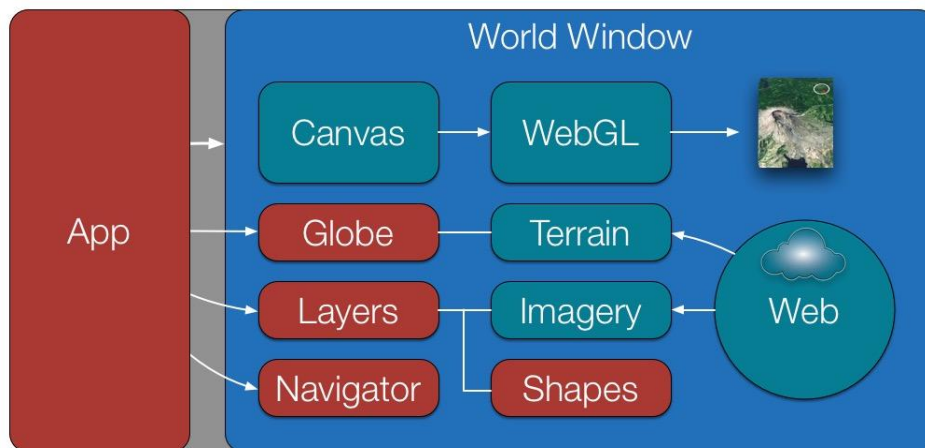


Figure 13 – Web WorldWind structure [33]

The World Window is an object, which contains all the Web WorldWind functionalities and connects them to an HTML container: the canvas. Each application implementing Web WorldWind should have at least one canvas with a World Window object.

The globe is the 3D representation of the Earth, a WGS84 ellipsoid containing terrain. Though it allows provides for viewing several different 2D projections in addition to the 3D mode. Usually, the applications do not interact directly with the globe, but implement methods to get information from it, even though it also possible to edit the globe itself.

Layers are fundamental objects to implement several features. Besides the World Window, they are typically the most important ways of representing spatial data. They permit one to insert imagery via GeoTIFF, JPG, and PNG. These permits shapes in 2D and 3D, also terrain conventional ones, to follow its elevation model. At a minimum there is the compass layer and the coordinates display, in addition to much more.

The shapes are objects, 2D or 3D, which display some information on the globe. Various objects belong to the shapes category, such as placemarks, 2D drawings, and simple or complex volumes. It is possible to import these into the globe or create them using the related APIs.

The Navigator is responsible for converting user actions to manipulations of the globe. It monitors mouse events and user gestures and translates them to pan, zoom or tilt operations on the globe. The Navigator can also be driven by the application to

move the view to a particular location or to set its tilt and heading.

The Navigator is an object automatically integrated in each World Window, allowing interacting with the globe to get all the information about its position and edit them. Developers can customize this to navigate the globe in different ways. Making it possible to customize mouse movement and gestures to manipulate the globe. The navigator can also be controlled by mobile devices, and can be personalized to assign any mobile gestures to a specific action in the globe.

Among the features, there are also Geocoders that permit discovery of geographic locations from a query string. Web WorldWind implements the one from Open Street Map's Nominatim geocoder at MapQuest¹⁶. Thanks to this geocoder, it is possible to retrieve information about a city, area or location, from its name or coordinates and then show all the desired information in the globe.

Web WorldWind contains several other objects, most of them run behind the scenes so it is not required that developers interact with them. Instead, when a developer wants to extend the WorldWind features, they only need interact with specific components, using the JavaScript's APIs. So it is quite easy extend the features already implemented or even create new ones.

4.5.2 jQuery

As an external library to interact with the user interface of the application, we used jQuery¹⁷.

jQuery allows to interact easily with the DOM (Document Object Model) thus with the use interface. It also contains a few functionalities that help with the creation of algorithms, simplifying the code. jQuery was first published in 2006 and is still an active project in continuous evolution. jQuery Core is made of different parts. The main components are listed below.

jQuery Core

Constructors allow easy selecting of elements in the DOM using shorter syntax than native JavaScript. It is also possible to obtain elements and mentioning a DOM element as a parameter. Moreover, the core allows creating new elements starting from pure HTML code.

Methods

Methods allow accessing of the element in any jQuery element. They provide functionalities to get the size and length of items, iterating through groups and edit native components from HTML. There are also several methods to handle list, queues and to extend the framework through plugins and create animations.

Selectors

Selectors allow access to HTML element in a page. The syntax is the same used in the CSS, Cascading Style Sheets. The selectors can be used to select an element by its "id", based on the "class", in a hierarchical way or thanks to the content or attributes.

¹⁶ <https://open.mapquestapi.com/nominatim/>

¹⁷ <https://jquery.com/>

Attributes

Attributes provide different functionalities based on the element type. There are methods for generic attributes to get all the attributes of an element. Then a method for classes to know if an element belongs to a specific class, and if it is possible to remove or add classes it. There are also methods for content, to get information of any HTML element, such as text, inner HTML or any values from a form.

DOM Traversing

The DOM Traversing permits to navigate through the DOM to reach parent and children of any elements. It is possible to go through it in different ways, thanks to several features the framework provides.

Manipulation

The manipulation of the DOM is already made possible by JavaScript, but jQuery simplifies this. It consents to add or remove an element from a page in a specific position. Moreover, it allows you to substitute elements or to wrap them with other elements. It also favors the deletion of a specific node and the copying of it.

CSS

There are several ways to control the style of the elements. It is possible to change the graphical properties of any selectable elements. It allows changing easily some elements that otherwise would be difficult to manipulate.

Events

The events that happen on the page are recognized by the framework. Is possible to intercept these events and change their behavior. The event handling also permits to assign different events such as click, load, and mouse events to various functionalities.

Effects

The effects of the framework permit to manipulate the visibility of the selected elements, the manipulation can be done with different effects, such as fading, sliding and more.

AJAX Effects

The AJAX (Asynchronous JavaScript and XML) calls are an asynchronous call to diverse pages. jQuery allows using them in an easy way. It is possible to load in a dynamic way some contents with HTML code. Moreover, to make GET, POST requests, and integrate them with JavaScript to load JSON objects and execute remotely script files.

Utility

Also, it provides several utilities to manipulate strings handle components and manage various JavaScript objects.

4.5.3 Bootstrap

To represent some elements in the web interface we used and Bootstrap¹⁸. Bootstrap helps to customize all the graphic elements in the user interface.

Bootstrap is a free and open-source library to create and customize items in the front

¹⁸ <http://getbootstrap.com/>

end of web applications. It is made by functionalities to design forms, buttons, navigation and many interface components. It favors the responsive web design; thus, all the contents are easily scalable to any device.

The architecture of Bootstrap is modular and is made of the different stylesheet to implement all the functionalities of each component. All the stylesheets are in "Fewer stylesheets" format, and it is possible to customize them to change the style to apply to the components. Moreover, Bootstrap provides not only HTML components but user interface elements. These interface elements are groups of buttons, drop-down options, navigation lists, navigation tabs and so on.

Some Bootstrap components use jQuery functions. These components provide several additional functionalities in the framework. Some of these are dialog boxes, tooltips and so on. Even some jQuery plugins are supported and automatically styled from Bootstrap, with one example being the Dropdown, Tab, Alert, Collapse, along with others.

4.5.4 jQuery UI

jQuery UI¹⁹ contains few user interface elements that are not available by default from HTML. It is an open source library, providing plug-ins for jQuery. It is made of jQuery components, and each plug-in is made following the architecture of jQuery; since it is made of jQuery elements, it is fully customizable. It provided interaction with the DOM and complex animations.

Like jQuery it has a structure in sub-modules. The interactions permit drag and drop, scaling and selection of components. The widgets are several UI (User Interface) elements, for different purposes, such as complex buttons, date picker or dialog windows. The effects are animated transaction among the elements. They permit changing the color of the user interface's elements, hide and show with some transitions and much more. The utilities are used to extend and interact with the components of jQuery UI, and to establish relationships among all the elements.

4.5.5 Google Charts

Another library used to create some graphs when showing statistics about the data, is Google Charts²⁰. It simply provides APIs – application user interface – functionalities to create graphs. It consists of a rich gallery of graphs; there are scatter plots, glyphs, and others. It is possible to customize them, changing each property in the chart, from the color to the axis shapes and name. It is cross-browser compatible and made of pure web technologies, so no external plugins or libraries are needed and it is provided free of charge.

The APIs allow the users to edit them in real time, connecting to the graph thanks to some functionalities and interacting with them. The most common way to integrate Google Charts in a web application consist on inserting a JavaScript snippet in the

¹⁹ <https://jqueryui.com/>

²⁰ <https://developers.google.com/chart/>

HTML page and then it is possible to use the functionalities provided by the library.

The charts are JavaScript classes, and when a user needs to edit the graph, he will interact with the class representing the specific graph. The customization is possible since the JavaScript classes are customizable, and all the parameters can be accessed and edited. To populate a chart, a specific "DataTable" class is used. This makes it easy to switch among all the different types of charts having a predefined class for the data. Within the DataTable class are methods for diverse functionalities. It is possible to sort and filter the data and can be filled from different databases.

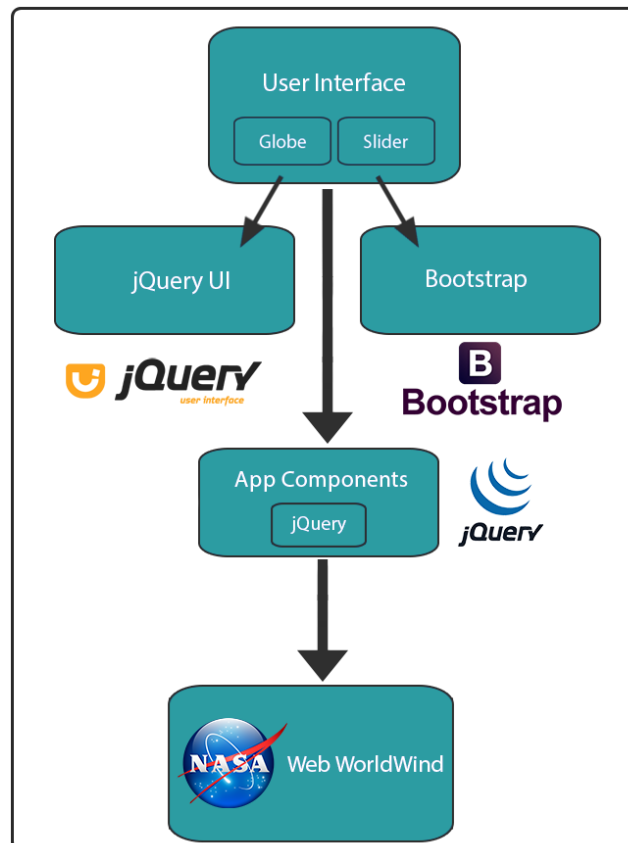


Figure 14 – Application Architecture

4.6 Application Architecture: Server Side

Our application runs fully on any recent browser, so on a client. However, the implementation of a server helps to manage the data and show our case study. We allow users to perform any operation from the client and handle their dataset on the client side, but following our implementation idea of the server may help to handle better the data. Having a server might help the users to upload the data to a database to require them easily. We show how we implemented the server and which techniques we have used to perform the queries to the database. We thus implemented two typology of databases. One for handling NetCDF files, so huge amount of data in multiple dimensions with a geographic component. While we implemented another typology for different datasets with smaller dimensions and

not georeferenced directly but with a reference to a cell ID that contains the information about the position. In the case studies we will see the structure of the data and it will be easier to understand the choice we have made for the typology of databases.

4.6.1 Rasdaman

We chose Rasdaman for implementing large coverages provided by Raster file or NetCDF datasets with several dimensions. As we explained before in Chapter 3 Rasdaman is the best options when handling huge datasets that have georeferenced information. Rasdaman provides WPC and WPCS that can be queried easily from our application to retrieve the data with AJAX requests. Thus it was the optimal solution to store a vast amount of information.

To use Rasdaman, we simply imported the desired dataset in the database, specifying all the information regarding the geographic extensions. Then, it is possible to perform queries through simple HTTP requests using the WCS and WCPS.

4.6.2 MongoDB

MongoDB²¹ is a document-oriented database. We choose it for datasets that are not directly georeferenced but implement relational information with the spatial reference. In this case the position can be associated to the data on the client side. MongoDB It belongs to the NoSQL database, which means it does not have relations among tables, like in the SQL databases. It is published under open-source license from 2009. It stores JSON-like documents, making the database highly portable to JavaScript application, for using the data in a client. The presence of a JSON-like document is an advantage in our case, having a JavaScript application, since all the documents work without any conversion of format, using pure JavaScript code.

Each record in MongoDB is a document. Each document has key-value fields, and each field can contain other children made of key-value. Those children can be not only single values but arrays, or documents as well. It has an easy query language that allows to, sort and aggregate data with simple instructions, without writing long and complex code Here is shown a sample of a MongoDB document.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```



The diagram shows a sample MongoDB document in JSON format. To the right of the document, four arrows point to the right, each labeled "field: value". The arrows point to the values of the fields: "sue", 26, "A", and ["news", "sports"].

Figure 15 - Sample MongoDB document

²¹ <https://www.mongodb.com>

4.6.3 Document Model

Since with MongoDB we wanted to store information for datasets not directly geo-referenced we created a sample model that the application is capable of reading. Taking as example the structure of a simple dataset we are using, made of telecommunication data, and having information about: a timestamp, an ID, and several variables, among them also a timestamp. This document may represent the sample for several datasets exploiting a grid and a cell ID to show data in the application. A sample model for the document is shown in the picture below.

```
1 {
2   "_id": ObjectID("559d286f27819c41006bf3c1"),
3   "cellId": 1,
4   "data_mi": [
5     {
6       "country": "0",
7       "sms_in": 0.005361930316498313
8     },
9     {
10      "sms_in": 1.0526344513332655,
11      "call_out": 0.16031366742441833,
12      "country": "39",
13      "int_traf": 13.474583498014898,
14      "sms_out": 0.5972066038316974,
15      "call_in": 0.08073835401865896
16    }
17  ],
18  "ts": ISODate("2013-12-08T21:40:00.000Z")
19 }
```

Figure 16 - MongoDB document model

4.6.4 Node.js

MongoDB offers several functionalities to the external client. To connect to it and perform some queries, there are thus different techniques. An easy way to query MongoDB is using Node.js.

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. [34]

The choice of Node.js²² derives from the requirements of scalability to handle different clients asynchronously and having good performances at the same time. The usage of Node.js is straightforward and intuitive and permit to create easily efficient REST APIs to allow the interaction with the client. Moreover, is useful to share the same programming language between client and server, enabling to transfer objects that will be treated in the same way from the functionalities of the application. Most important, we are designing a scalable web application, at high

²² <https://nodejs.org/>

levels of concurrency, so we can appreciate the management of non-blocking asynchronously request done by Node.js.

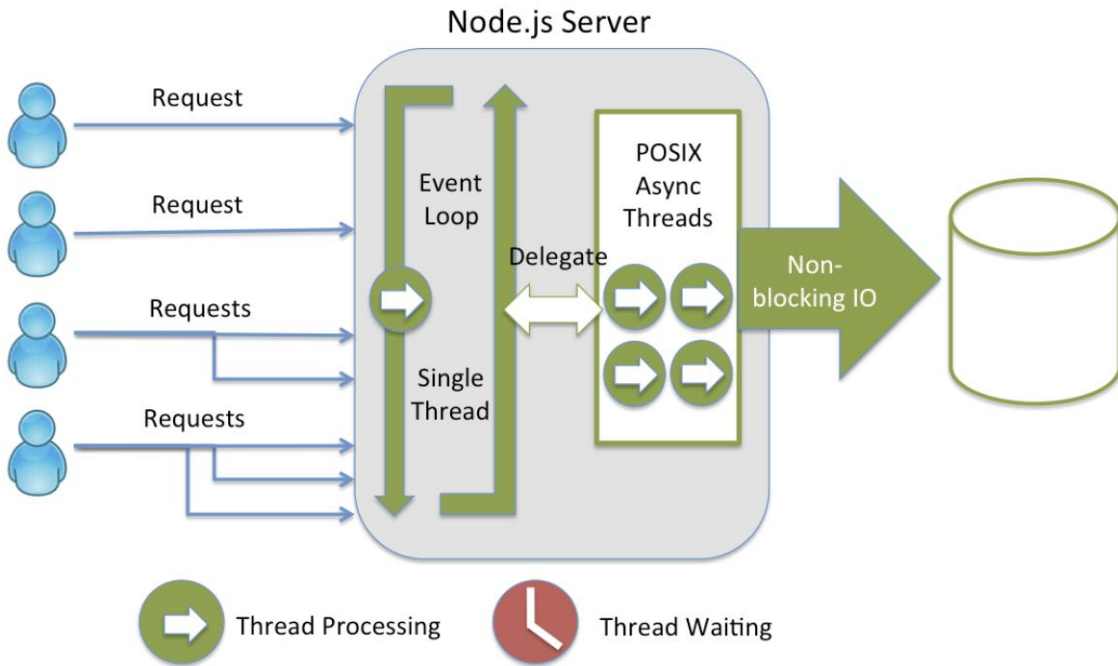


Figure 17 – Node.js Server with non-blocking IO [35]

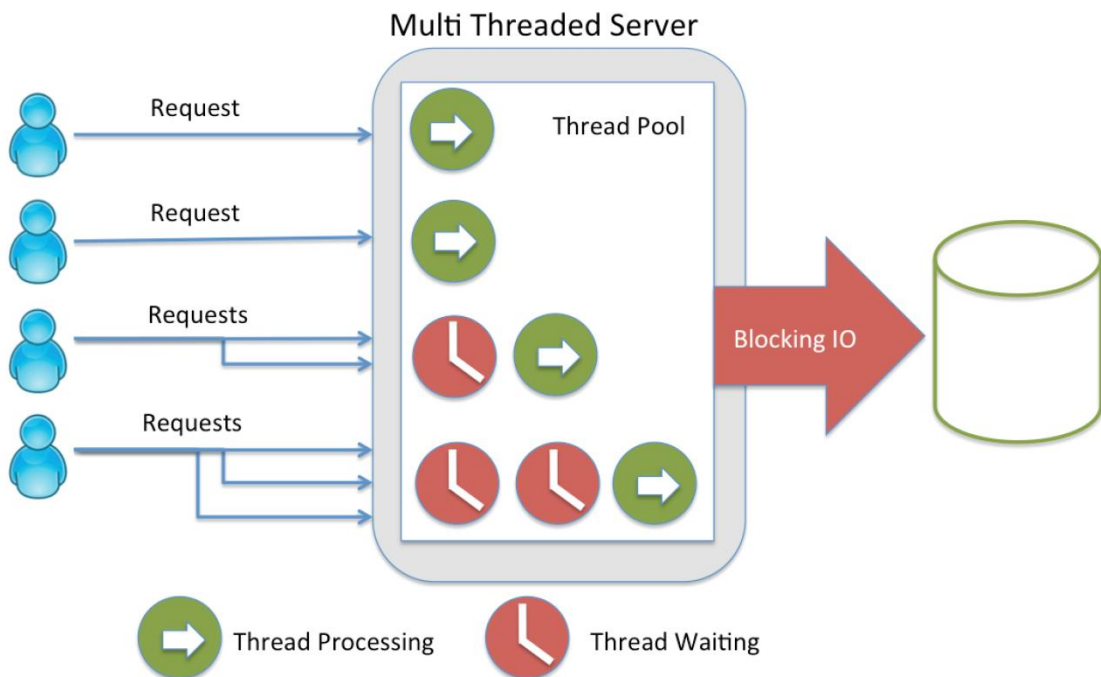


Figure 18 – Multi-Threaded Server with blocking IO [35]

4.6.5 Express

“Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.” [36]

The choice of Express²³ is almost mandatory with Node.js since it implements a lot of useful features, and together these two provide efficient ways to interact with the clients and handle GET and POST requests in a very easy way.

4.6.6 REST Interface

In computing, representational state transfer (REST) is the software architectural style of the World Wide Web. More precisely, REST is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. [37]

Since we need to query the database from the application, we had to implement a service to connect the database and the web application. We created a REST interface that permits the communication between the two, sending AJAX requests from the web page to the Node.js server.

²³ <http://expressjs.com>

4.7 Application Components and Flow

The following class diagram shows all the JavaScript classes implemented and their methods.

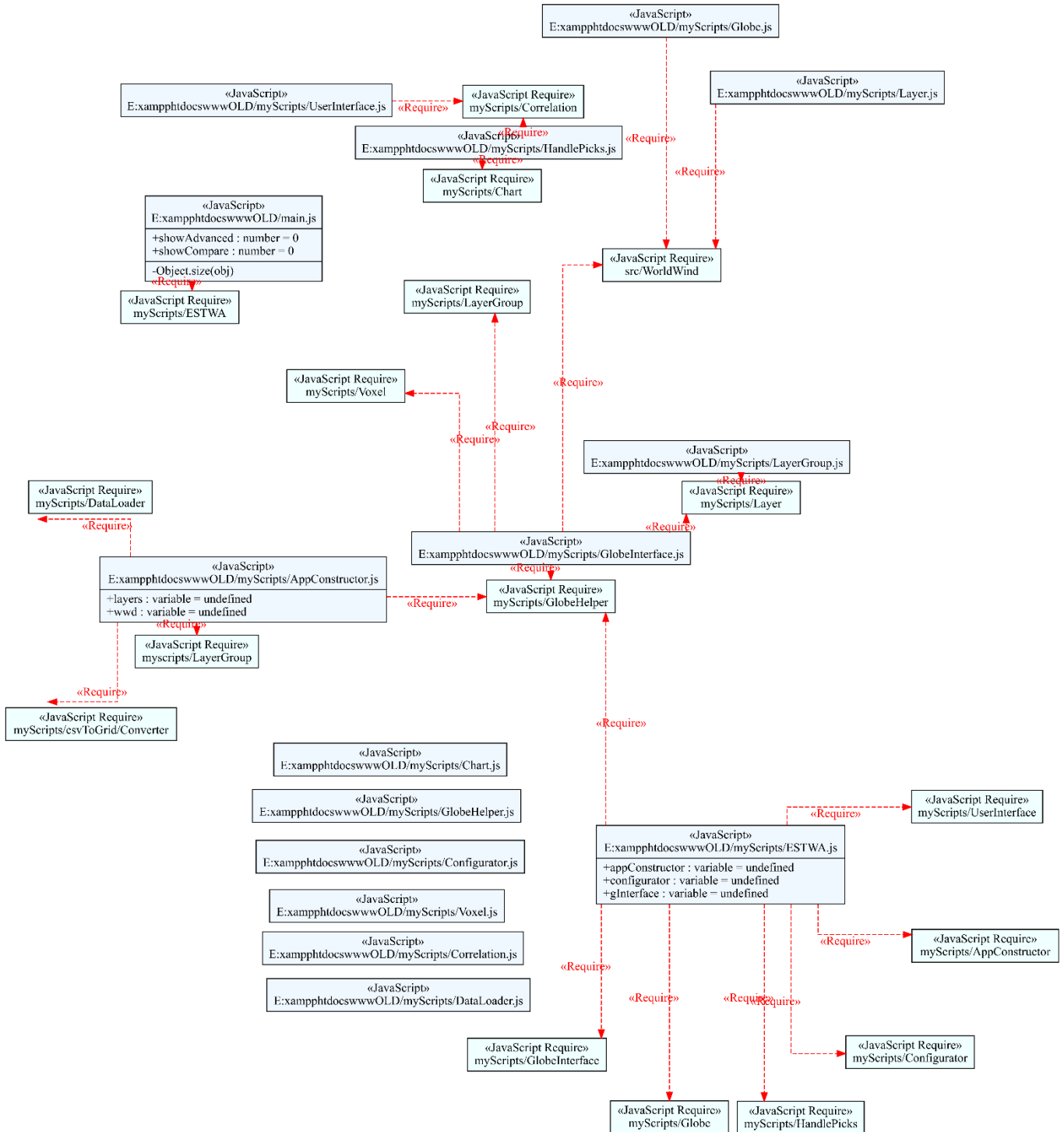


Figure 19 – Class Diagram of ESTWA Application

We can then observe a particular class *ESTWA* and the dependency of the class in the diagram below.

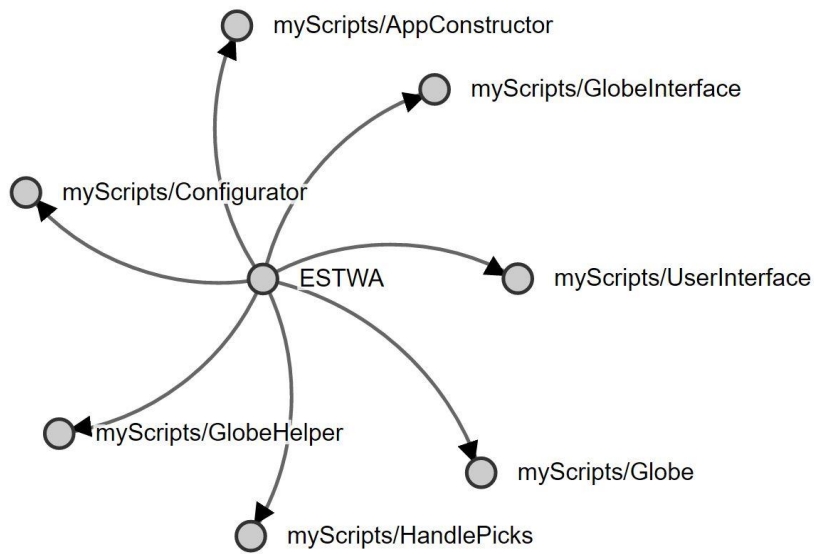


Figure 20 – ESTWA Class dependencies

The application has a linear structure rather than a hierarchical one. All the functionalities and modules can communicate with each other. We are going to show now all the classes and how their methods are implemented, to understand which functionality is supported by which class and method. We can see from the picture below, how the classes are connected each other.

4.7.1 Terminology and Code Convention

For ease of use, from now on, the word “layer” refers to a Web WorldWind layer object, made of a single array of doxels corresponding to a specific time range, depending on the time’s resolution of the dataset.

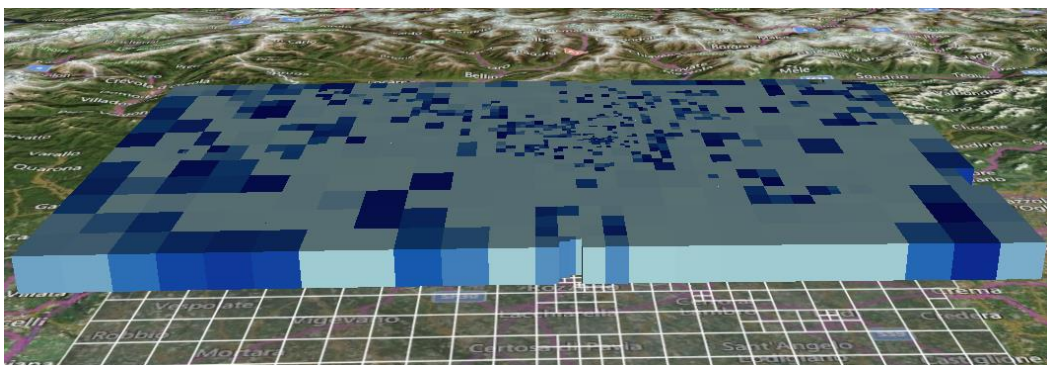


Figure 21 – Layer made of doxels

When we refer to a JavaScript “Class”, by the term class, we refer to a single file, named with the same name of the class and a “.js” extension. Moreover, by the term class, we mean a JavaScript Objects that has some functionalities and we will call

those functionalities “Methods”. In this document, every time we want to mention a class, we will write the name in italic with the first letter capitalized, following the CamelCase notation [38]. As well, for the methods we will use the same notation used for the Classes, but the first letter will be lowercase.

Type:	Notation:
Classes	MyClass
Methods	myMethod

Table 5 – Example of CamelCase notation

4.7.2 Data Importing from CSV

In this process, the user can select a dataset in a CSV format and a relative grid to import. The importation mechanism retrieves a sample of the dataset and the user can choose which column of the CSV file correspond to which variable in the application.

Figure 22 – CSV importing configurator

After the configuration of the file and grid is set, the system will request the data and import the grid. The request will go through a module “Data Importer” which will check the configuration created from the user interface when starting this process. Once we have imported the data we need to handle them. The application will now create an object “myData” to structure the data, grouping them by the timestamp. In this way, we have the following structure for each of our objects:

```
myData = {
  "3/1/2015 12:00:00 AM": [
    ["3939_0_0", "9.880.921", "1.570.428"],
    ["3939_0_1", "8.362.754", "1.051.640"],
    ["3939_0_2", "13.907.400", "18.715.431"]
  ]
}
```

Code Snippet 1 – Structure of the parsed data

In the meanwhile, the bounds of the data will be calculated, to know the minimum and the maximum values in our dataset.

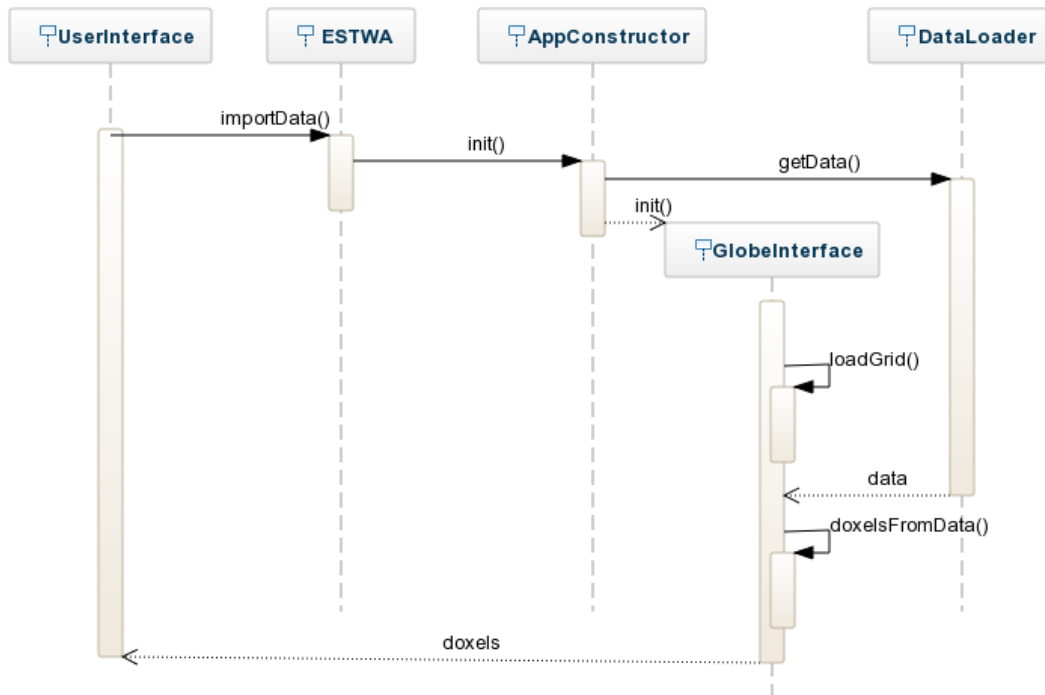


Figure 23 – Data Importing Flow

4.7.3 Doxels creation

After the importation, the new data will now be sent to the *GlobeInterface* to create the doxels on the globe. At this point, for each entry in the data, we will invoke the module to create the Voxels having the coordinates of each one, retrieved from the *gridId*, and the desired color.

To calculate the color, we take into consideration the minimum and maximum values available among all the entries in the dataset. We then assign a percentage value for each voxel to retrieve a specific color in a three colors range as shown in the image below.

```

GlobeHelper.getColor = function (weight, inputColors) {
  var p, colors = [];
  if (weight < 50) {
    colors[1] = inputColors[0];
    colors[0] = inputColors[1];
    p = weight / 50;
  } else {
    colors[1] = inputColors[1];
    colors[0] = inputColors[2];
    p = (weight - 50) / 50;
  }
  var w = p * 2 - 1;
  var w1 = (w / 1 + 1) / 2;
  var w2 = 1 - w1;
}
  
```

```

var rgb = [Math.round(colors[0][0] * w1 + colors[1][0] * w2),
  Math.round(colors[0][1] * w1 + colors[1][1] * w2),
  Math.round(colors[0][2] * w1 + colors[1][2] * w2)
];
return [rgb[0], rgb[1], rgb[2], 255];
};

```

Code Snippet 2 – Color Retrieval through *getColor* method

All the doxels will be placed in layers, each layer for each timestamp. This means that for each timestamp we have a single group of doxels placed at the same height. All the layers will now be positioned in the globe, although we will show just the desired ones, configured from the user interface. After all the layers are available in the globe, we will show the new user interface with the sliders and the settings to handle the doxels.

Involved Classes and Methods

To achieve the importing, we start from the user interface where the *ESTWA* class manages the configuration of the data in the interface. Therefore, the class *AppConstructor* is invoked through the method *init* to set the entire setup and initialize the *GlobeInterface* class. Moreover, the class *DataLoader* is instantiated.

Now thanks to the method *loadGrid* of the *GlobeInterface* class we load the grid. In the meanwhile, with the method *getData* from the *DataLoader*, we retrieve all the entries from the CSV. When the grid has finished loading, and the data is available, we call the method *doxelFromData* from the *GlobeInterface* to create the Voxels, passing the data we imported. Then the *UserInterface* class will start and also the sliders will be instantiated calling the method *startSlider*.

4.7.4 Data Importing from Database

Depending on the configuration, the data might be imported from the database or a CSV, file. When importing from the database the system will perform a query to our database to retrieve the dataset. In this case, no configuration is needed, since the database is already configured for our application.

4.7.5 Retrieving Doxels data

To retrieve all the information about each doxel we created a click handler to recognize when a user clicks on the globe and select a visible doxel. The click handler will be activated by a double click.

The double click action will activate the handler in the *ESTWA* class to recognize the specific doxel. Since each doxel is represented by an object, we store all the information inside it. Therefore, we first highlight the selected doxel is creating a simple white outline and slightly lifting it, changing its height, to show that it has been chosen.

Then we retrieve all the information about the doxel and the other doxels having the same position in different time. We thus display them in the user interface in a graph, showing the time trend. This is done for each variable available in the input file. We can observe thus a view with several variables in time. Also, we can see a correlation value taking into consideration the values of the selected. Besides the correlation among a single doxel, we also calculated the correlation for all the voxels in all the layers of the time.

The retrieving process of the information from the big doxels works in the same way. We cannot retrieve all the information about the doxels as grouped, but just the statistical index stored inside each of them.

Involved Classes and Methods

When clicking on the doxels, the *ESTWA* class calls the method *getDoxel* from the class *HandlePicks*. Within the method *getDoxel* is another helper that will be called to retrieve the information and the statistics of the doxels, such as the method *getCorrelation* of the *Correlation* class.

4.7.6 Big Doxels Creation

The creation of the big doxels will consider the configuration specified by the user interface when importing the data. To create them, a few steps are taken into consideration. The first parameter we have from the configuration is for subdivisions s . We use this to subdivide the area of the grid in $s * s$ equally spaced rectangles. Of course s should be less than the number of rows and columns of the grid.

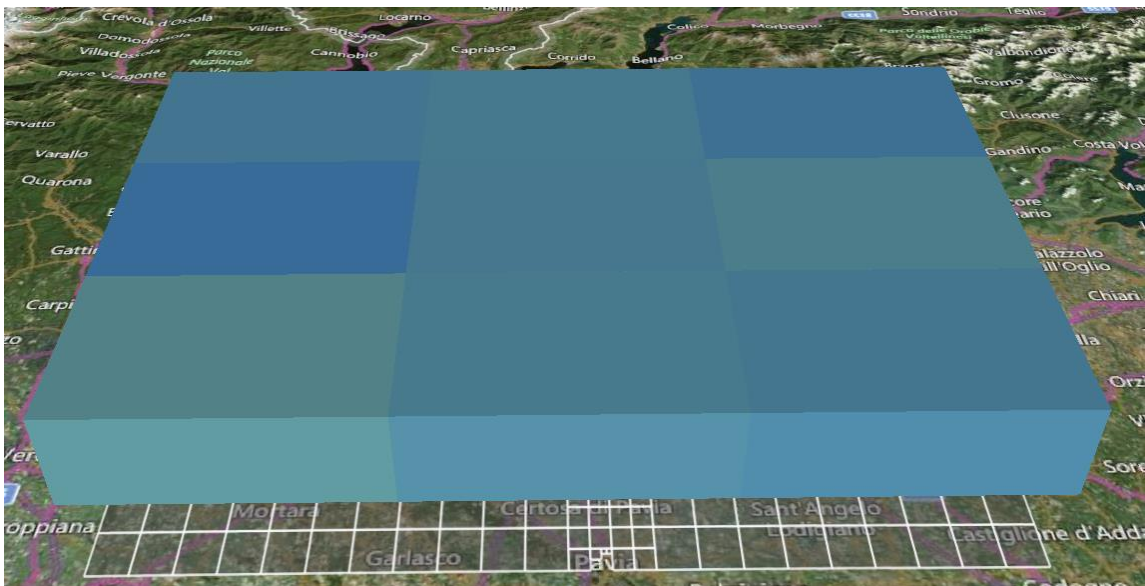


Figure 24 – Subdivision of a rectangle in 3x3*1 dimensions

In each layer we then linked, each doxel to the corresponding rectangle, by

intersecting the projection on the globe of each doxel with the rectangles and identifying for each doxel the related one. We then have $s * s$ rectangles and each has a direct reference to specific doxels. Also, a number h of subdivisions in the z-axis is selectable from the configuration that we can assign the doxels linked by these rectangles to some groups. Depending on the timestamp we subdivide the doxels into further groups. Once more h should be less than the visible layers of doxels.

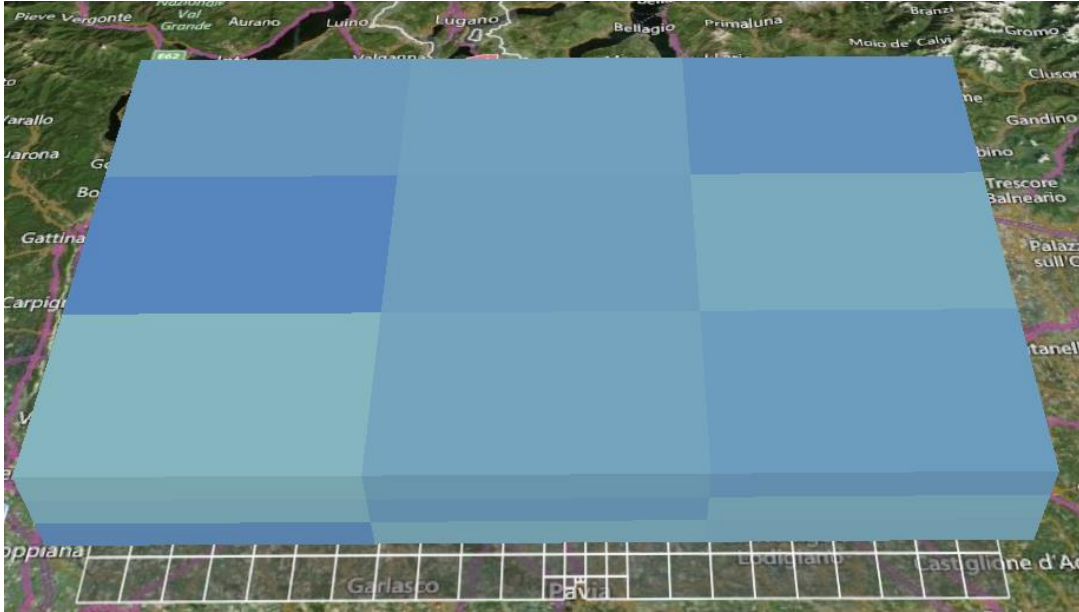


Figure 25 – Subdivision of a rectangle in $3*3*3$ dimensions

We will then have the doxels subdivided among $s * s * h$ groups. To now create the big doxels with the color of statistical representation of the data, we process all the data in each group and retrieve all the value information to represent the color. With this we can create the big doxels, each having the dimension of the specified group. The color will be calculated from the statistical index specified in the configuration, and this is possible by having the number of doxels, their dimensions, and all the data for each of them. In the same way, a value representing the statistic will be stored in each big doxel object.

Involved Classes and Methods

To first create the rectangles, we invoke the method *createRect* from the class *GlobeInterface*. Then the *GlobeInterface* will also call the method *makeBigDoxels*, which will hide the layers of normal doxels, check how to create the doxels and then create them with the method *getBigCubes*. To create them, we need to get the statistic information that is provided from the *GlobeHelper* with the method *getStatistics*

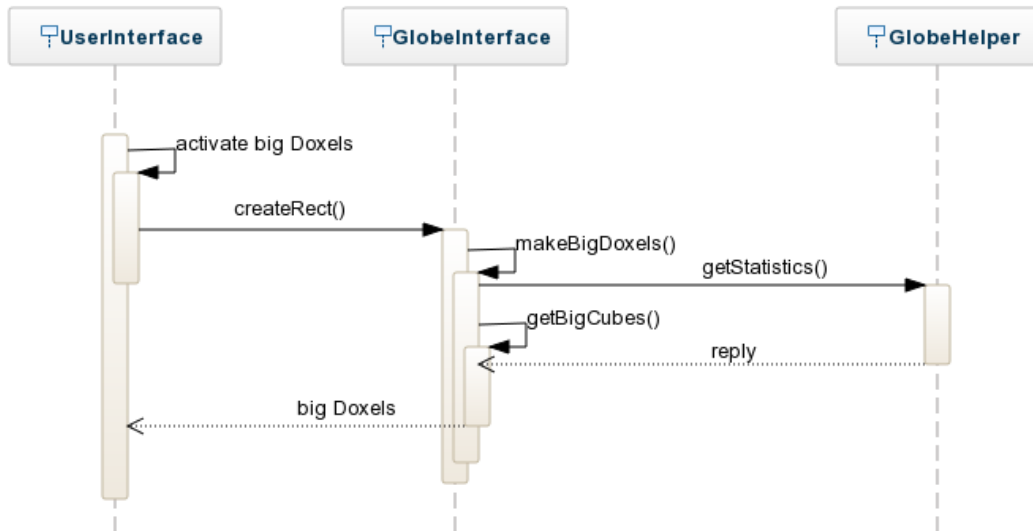


Figure 26 –Big Voxels creation flow

4.7.7 Big Doxels Showing & Hiding

Alongside the handler to recognize the click to retrieve the data, we created a second handler to allow users to switch between the big doxels to the original one. A radio button allows the users to select which handler to use. When the big doxel handler is selected, the big doxels will be created. From that moment, double-clicking on a big doxels will show the grouped doxels it contains. In this process, we select all the doxels that belong to the rectangle assigned to the big doxels and show them, given that no filters are hiding that specific doxel.

Involved Classes and Methods

A click on a doxel is handled from the procedure to recognize clicks. It is activated from the class *HandlePicks* using the method *getBigDoxels*, which instantiates a recognizer for the click from the *ESTWA* class.

4.7.8 Filtering

The filtering functionalities to hide the data works in different ways. We explain below the technique adopted to create them. Other techniques might have been used, but we implemented the best ones regarding performances. All the filters are activated from the interface on the web application, which is handled by the *ESTWA* class and then coordinated with the *UserInterface* class.

4.7.9 Latitude & Longitude Filter

To filter the values of latitude and longitude we created a hidden moving rectangle with the dimension of the grid. From the sliders when we move the handles they will change the dimension of the rectangle. Each step of the slider's handles will trigger the process to modify the dimension of the rectangle and start the function to decide

which doxels to show or hide.

The process will start by setting the new dimension and then check for each visible doxel if its projection is inside the area of the rectangle, in the same way, used for the creation of the big doxels. If a doxel is not inside the rectangle, the process will hide it. During this process, we also check if the big doxels are visible, and if so, instead of hiding or showing the single doxels, if the big doxels is placed inside or outside the area, it will be shown or hidden.

```
gInterface.changeSize(ui.values, 1);  
var direction;  
if (ui.values[0] > self.oldValLng[0] || ui.values[1] <  
self.oldValLng[1]) {  
  direction = 0;  
} else {  
  direction = 1;  
}  
gInterface.moveWindow(direction);
```

Code Snippet 3 – Process to filter the longitude from *UserInterface*

Involved Classes and Methods

All the filters are called from the *UserInterface* class that instantiates some handlers to get the movement of the handles in the interface. When the handles are moved, after retrieving the value we call two methods from the *GlobeInterface* class. First, we call the *changeSize* method to change the size of the rectangle that controls the doxels visibility. Then we call the *moveWindow* method to check inside the rectangle which doxels should be shown or hidden and to check if the big doxels are active.

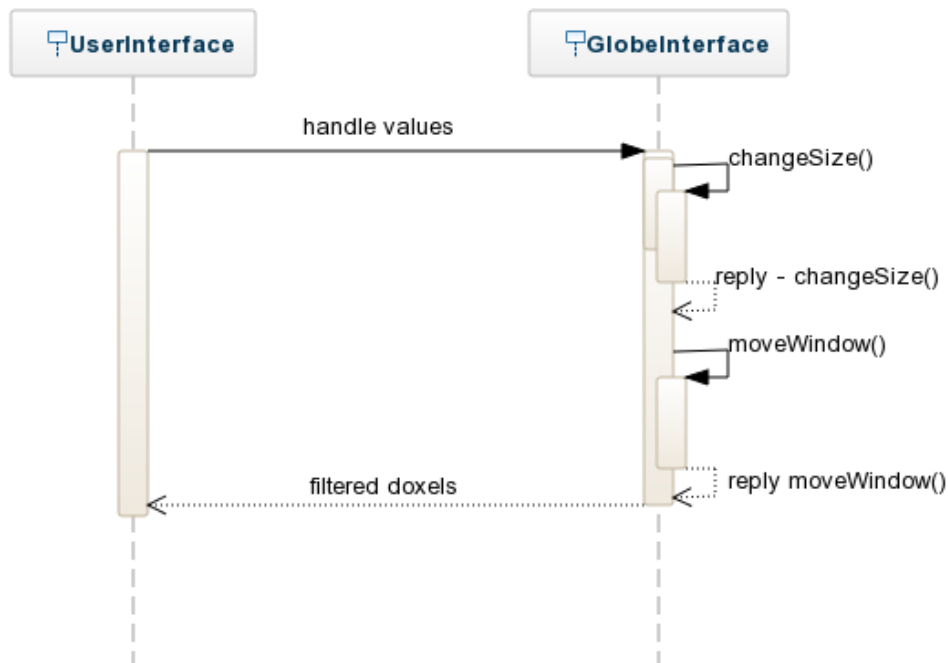


Figure 27 – Latitude and Longitude filter flow

4.7.10 Altitude Filter

The slider on the altitude works in an easy way by hiding the layers of doxels in the range of the slider. Although it is possible to change the time view and display different layers, the application detects which layers are visible at the current time and can operate alongside the other filters.

Involved Classes and Methods

Similarly, to what happens in the latitude and longitude filters, moving the handles to control the altitude, we trigger the handler for the altitude, which calls the method *changeAltitude* from the *GlobeInterface*. This method will execute all the changing in the altitude and show the new doxels on the globe.

4.7.11 Values Filter

The slider for the values will retrieve the bound of the data, and operate within the range between the minimum and the maximum in the data. Each time we move the handles, it will create a range of minimum and maximum. The process will then go to all the available layers and check if the data stored in each doxels is within the range; if not the doxels will be hidden.

Involved Classes and Methods

The changes on the handles for the values trigger the recognizer in the *UserInterface* to get the values we want to filter and pass this information to the *GlobeInterface*.

4.7.12 Time Browsing

The opportunity of going through different time ranges is given, thanks to the slider that allows users to select a different time. It will choose the starting time and show a predefined number – designated in the configuration interface – of layers after that interval. When changing values, we start a process from the *Globe Interface* that changes the new starting time range. Each step performed by the moving of the handle will increase or decrease the beginning time. The values of the starting time are available in the “myData” object. The process will thus set a starting time that is the next or the previous of the current time.

```
gInterface.changeTime(ui.value);
if (gInterface.autoTime) {
    var compare = $("#checkCompare").is(':checked') ? 1 : 0;
    gInterface.compare = compare;
    gInterface.UI.resetFilter();
    gInterface.makeBigDoxels();
}
gInterface.changeAltitude(self.oldValAlt);
self.oldValTime = ui.value;
```

Code Snippet 4 – Changing time from the *UserInterface*

In the view, what happens is a shifting of all the doxels in the z-axis. Increasing the time, a process will start to go through all the doxels in all the layers and change the position of each doxel by decreasing their height by an amount equal to each doxel's size. This will look like that the values are moving in time. We also had to hide the least bottom layer – the old first time value – and show the next one on the top – last time value on the range – to keep the same number of visible layers.

During this process, all the previous filters will keep their effect. This means that if a filter on the latitude were applied, all the doxels that were hidden would stay hidden now. To achieve so, each time we cover up a doxels we had to specify which filter hid it, to avoid the interaction with the other filters.

Involved Classes and Methods

The time browsing is more complex than the filters and involves more methods. It is still activated from the *UserInterface* but each movement of the handle. Thus, a changing in the initial time will first call the method *changeTime* on the *GlobeInterface* along with *changeAltitude*. Then another set of processes will be activated in case the big doxels are visible. If so, the *UserInterface* will first reset the current filters with *resetFilter*, and then the *GlobeInterface* will call the method to create the big doxels again at each iteration with *makeBigDoxels*.

4.7.13 Updating Options

To allow the user to customize their experience, we give the user the possibility to change a few options even after the importing of the data.

4.7.14 Automatic Big Doxels

Selecting this option, each time the time-browsing slider will change, a new array of big doxels will be created. All the processes executed in the Big Doxels creation step will be reproduced. In this way, when changing the time, we will achieve in real time a statistical representation.

4.7.15 Starting Height

This will replace the starting height of the layers, setting the first layer shown at the selected height. The process that takes place here will go through all the doxels and change their position. All the position will decrease in the z-axis by a factor equal to the previous value, minus the new one. And again, once more, all the big doxels will be re-created.

4.7.16 Statistical Index

This option will set the flag that is checked during the Big Doxel's creation process. In this way, any time a new big Doxel is created, the corresponding formula will be computed to achieve the desired statistic.

```
switch (index) {  
  case 0: //weighted average
```

```

    value = sum / sumweight;
    break;

    case 1: //arithmetic average
    value = sumValue / iteration;
    break;

    case 2: // variance
    var aritAvg = sumValue / iteration;
    var variance = 0;
    for (n = 0; n < rect.cubes.length; n++) {
        if (rect.cubes[n].heightLayer == height) {
            var val = rect.cubes[n].data[compare];
            variance += (val - aritAvg) * (val - aritAvg);
        }
    }
    variance = variance / (iteration - 1);
    value = Math.sqrt(variance);
    break;

    case 3: //median
    median = Math.ceil(iteration / 2);
    value = rect.cubes[median].data[compare];
    break;

    case 4: //max
    value = max;
    break;

    case 5: //min
    value = min;
    break;

    default:
    value = sum / sumweight;
    break;
}

```

Code Snippet 5 – Switch for the statistic in GlobeHelper

4.7.17 Dataset Comparison

The comparison of two datasets changes the standard behavior of the application. From the interface, we can import the second dataset. When we select this option, the Creation process will invoke once more the Data Importer module, and we will have so two instances of “myData” object. Not all the previous modules can work with two cases of data, in the same way, so we created a reference parameter, referring to which “myData” the other processes have to take into consideration. Since the two data sets may have a different number of doxels in each layer, we create only the doxels that belong to both datasets. We create the doxels with the half color of one dataset and half of the other. We retrieve the color once per dataset and then in the creation of the doxel we set an option to create an image made of two colors, to use it as a texture for the doxel. Also, we store all the information about the two datasets inside one doxel. This means that even if it looks there are two doxels in one; it is just one doxel, with a texture and some data inside.

When we use filter functionalities to show the big Doxels, they will take into consideration the reference dataset and not the second one. If we want to operate with the second dataset, there is a checkbox to switch the reference dataset, it will change this flag, and so all the functionalities will know that the new reference changed and used it for the filtering. To get the information about the doxels when we have two datasets, the same click handler works, and we retrieve both the information for the two datasets.

Involved Classes and Methods

The flow to compare the dataset starts from the *ESTWA* where the user specifies all the parameters to import the data, calling the method *newData* from *AppContrusuctor*, which will import the data in the application calling the *DataLoader* class and the *getData* method. Then when the data are available, the *GlobeInterface* will start the creation of the doxel with *doxelFromData* and then *makeDoxel*.

4.7.18 Doxel Extrusion

The extrusion of the doxel is possible with one layer per time. All the functionalities work like the original flow, except the altitude slider, since we cannot change the range of altitudes. During the creation of the doxels, we take into consideration the second information from our dataset. We represented it modifying the default height specified in the configuration, but creating a range of minimum and maximum height, as we did for the colors. We obtain the weight thanks to the minimum and maximum value in the dataset, and we then multiply this by a constant value of height, to achieve the desired height for each doxel. This is the same flow made during the assignment of a color for each doxel but reproduced to modify the height.

Involved Classes and Methods

The extrusion of the height is a special case that will change the normal behavior that the application follows when importing the data from the interface. Thus, the same classes and method are executed. The exception flow is on the *makeDoxel* method in the *GlobeInterface* class. It will check if the option to extrude the doxel is active and extrude them based on their value.

4.8 User Interface Demonstration

Below all the functionalities previously explained are shown with a detailed description on how to get them and where to find them. Some images are available to see the result of them in the user interface.

4.8.1 Data Importing from CSV

To configure the importer for data we first select the CSV file for the dataset and clicking on “Load Configuration”, we will import the first line to let the user select from dropdown and input boxes the right parameters.

Time	1425164400
Grid Id	3939_0_0
Data	<div style="border: 1px solid gray; padding: 2px;"> <div style="border-bottom: 1px solid gray; padding: 2px;">9.880.921.898.890.530</div> <div style="border-bottom: 1px solid gray; padding: 2px;">39</div> <div style="padding: 2px;">1.570.428.368.965.030</div> </div>
Data Separator	.
Grid Separator	_

Figure 28 – Importing data interface

4.8.2 Personalization Options

Height of Voxel

Allow to choose the height of each voxel, thus the height of each layer.

Maximum layers visible

Will limit the number of layers shown in the view after importing the data.

Initial Height

Allows selecting an altitude from which the least bottom layer of doxels will start to be visualized. Inserting a value of zero all the doxels will start from the ground, but not following the terrain elevation model, so attached to the highest point in the elevation model; this is done to have all the doxels corresponding to a time layer at the same height.

Max layers in view

Permits to select a maximum number of layers to display simultaneously in the view. The minimum is one while the maximum is the maximum number of timestamp available containing data.

X/Y- Subdivisions

Indicates the number of row and columns in which one layer would be subdivided on the x-axis and y-axis (latitude and longitude) to represent the big doxels; thus indicates the root square of the number of large doxels represented. The minimum is one, to have a statistical representation over the whole layer. The maximum is the smallest number between the number of rows and columns in the grid, although having a number close to the maximum would not be much useful since no aggregation would take place hence no statistical representation of data.

Z – Subdivision

Refers to the number of layers of big doxels in which we desire grouping our layers. Once again, the minimum is one, to have a single layer of big doxels, while the maximum is the same as the maximum number of layers in view. Even in this case choosing the maximum value would be the same as not grouping them in the z-axis.

Color Range

Consists of three colors selectors from which the users can select the color range to use for displaying their data.

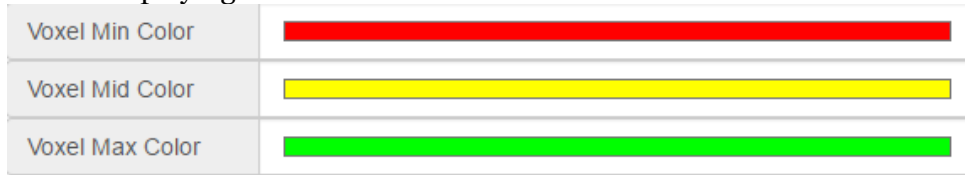


Figure 29 – Color Range interface selector

Statistical Index

A drop-down menu in which there are several statistical indexes available to choose; in order to show the desired statistic while grouping the data. The statistic will be represented following the same color range selected in the color-range option.

We implemented the following indexes: weighted average, arithmetical average, variance, median, maximum and minimum. Naturally, the weighted average and the arithmetical one will be the same in the case of a regular grid in which the cells have all the same dimension.

4.8.3 Interface Options

The second type of choices available are the ones for the interaction with the globe. They will be available only after having imported the data and thus have all the doxels available.

After inserting all the input values, clicking on the “Start” button we can process the data and display them in the globe.

A new Interface will appear now where we have more controls and other options.

Retrieving Doxels Data

Clicking on a doxel when the appropriate click handler is selected, will show a graph with the information about the doxel during the time and some statistics.

Big Doxels Creation

On the user interface, we placed a radio button to choose the handler, for showing and hiding the big doxels or selecting a single doxel and retrieve the information.

Clicking on the button for the big doxels the process will start the creation of them.

Big Doxels Hiding

When the handler for the big doxels is selected, double clicking on a big doxel will show the doxels inside, and vice-versa, clicking on a single doxel will show the big doxel to which it belongs.

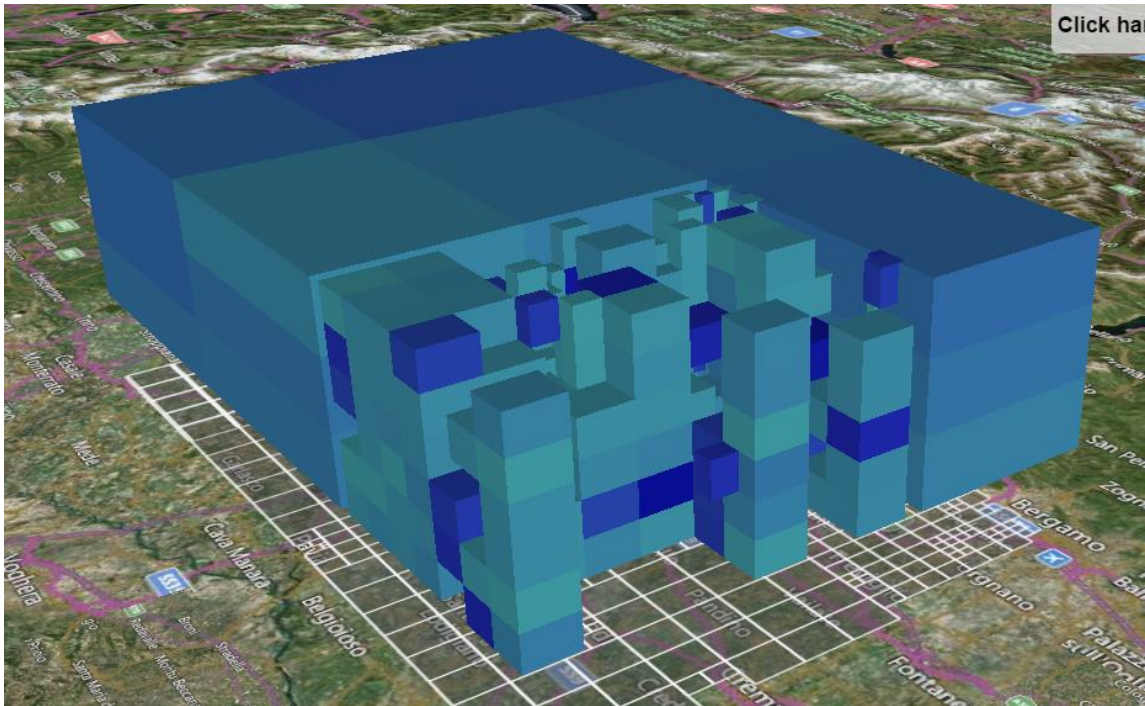


Figure 30 – Big Doxels partially hidden

Latitude & Longitude Sliders

We can drag and drop the left and right extent of this slider to limit the doxels we want to display, creating a range of values on the latitude.

The filter is the same as the latitude filter but permits to restrict the number of doxels to be displayed, creating a range of longitudes. It can operate alongside the latitude slider to limit the view up to a single doxel per layer.

Altitude Slider

This slider can restrict the number of layers to be displayed in the view. It is possible again to select a range; in this case, a range on timestamp to be visualized in the view. In the default case, the maximum number of layers available in the view are visualized.

Values Slider

This slider permits the user to set a range of values, between the minimum and maximum of the data set. The doxels that will be outside the range of values will be hidden.

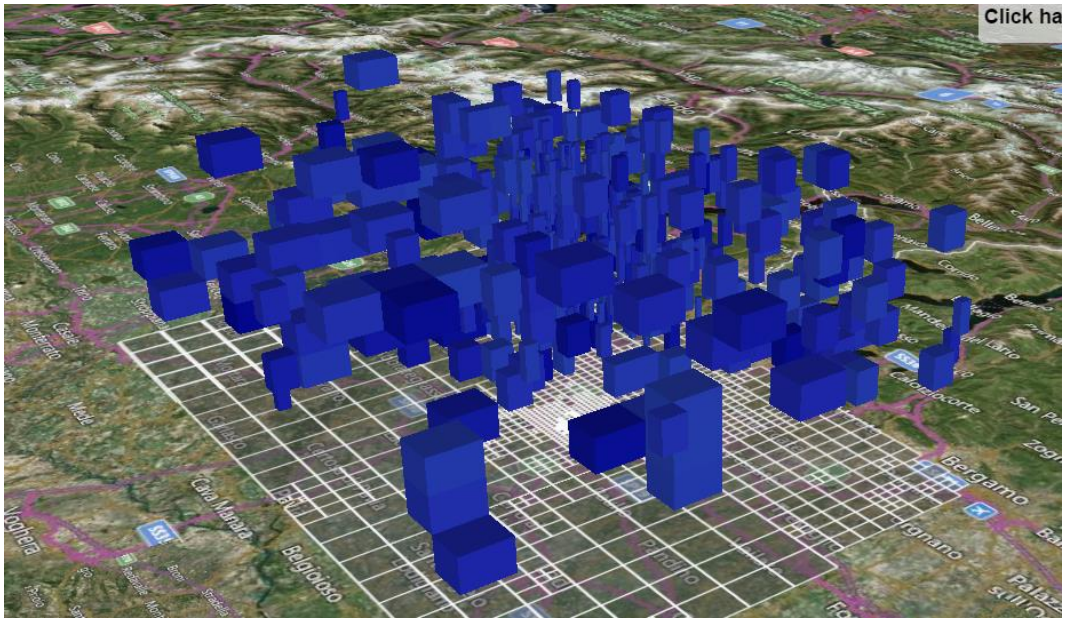


Figure 31 – Filtered Doxels on value

Time Browsing

This slider is the one to control the fourth dimension. It allows selecting a range of timespan to visualize. In this case, since many layers of timespan can be visualized in the view, it permits to navigate through the time. The initial value of the slider is the first available timestamp; it is thus possible to change this value up to the maximum timestamp available among the layers.

Moving this slider will create a sort of animation of what is happening to the layers during the time.

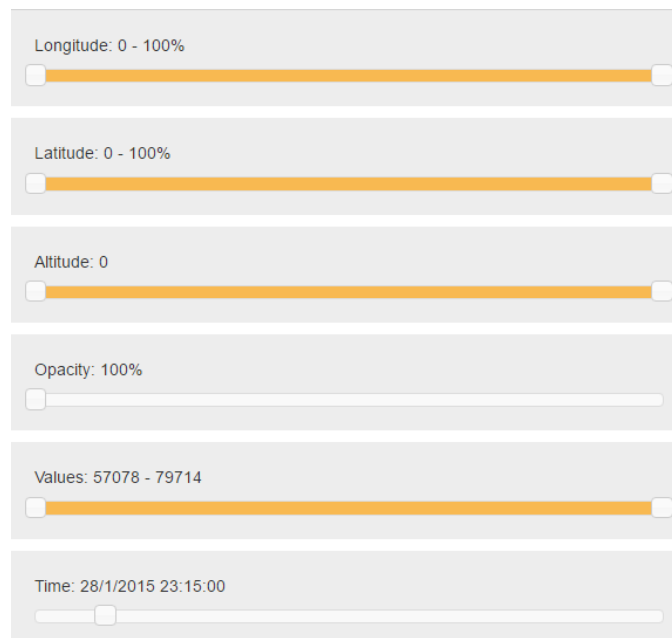


Figure 32 – Interface to control the filters

Updating Options

From the interface, it is possible to change few of the options we set during the importing of the data. Not all the options are available to be modified since some will require creating the voxels again from scratch.

Among the options is possible to change the initial height of the voxels, the statistical index for creating the big voxels and setting the automatic creation of the voxel when browsing through the data.

4.8.4 Dataset Comparison

The alternative flow of the dataset comparison consists in showing two datasets, or two variables, of the same dataset and comparing them in a single view. To do so, we need to limit the number of layers to one to appreciate the view.

After having a dataset visible from the options, we can upload a CSV representing the second dataset. It should contain the same time values and grid indices in order to be visualized beside the first dataset.

When the upload is completed, as we can see in the image below, it will appear and shows the doxels subdivided into two parts.

As regarding the filtering functionalities, they are available both for the first and second dataset. By default, the filtering, and all the options are set to the first dataset, but from the interface, selecting the box to use the second dataset instead, we can swap all the functionalities to have the second dataset as a reference.

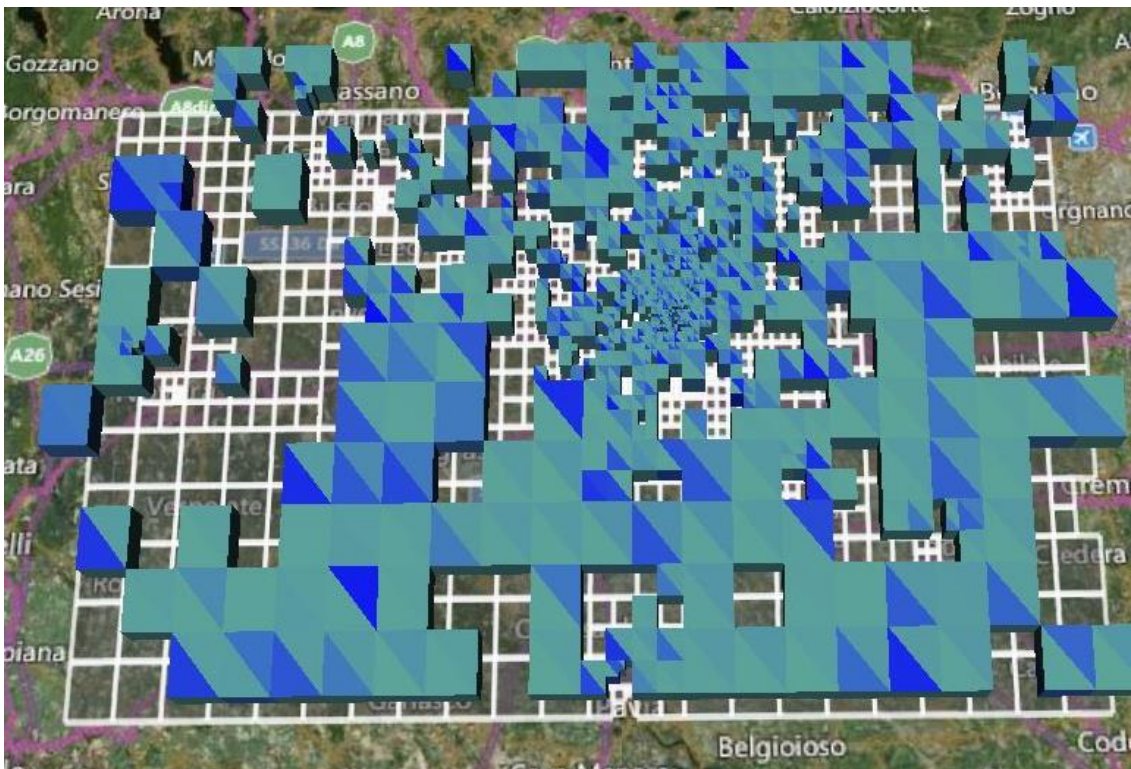


Figure 33 – Doxel comparison over two variables

4.8.5 Voxel Extrusion

The voxel extrusion, it is an alternative flow that can be shown instead of stacking the layers, one on top of the others. When we want to extrude the voxel, we lose the time variable in the same view, giving space to another variable, if the dataset comprehends more than one. Although it is still possible to go through the time with the slider time. We can thus observe five variables in the application: latitude, longitude, time, and two variables that we choose.

To achieve the voxel extrusion, we need from the user interface, before importing the data, to select the voxel extrusion checkbox. It will automatically set the number of maximum layers to one.

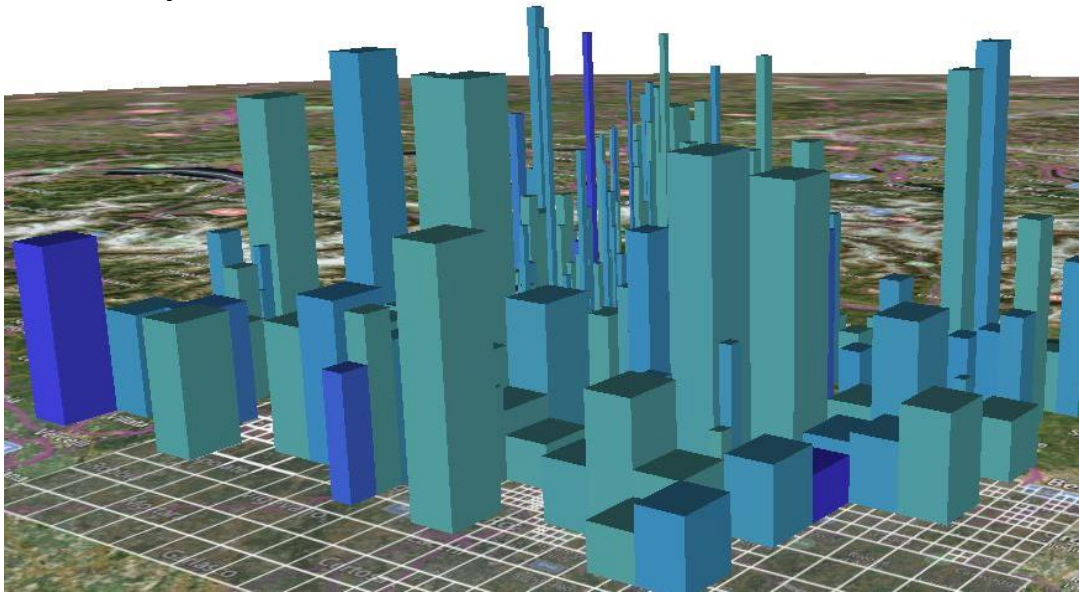


Figure 34 – Doxel extrusion using two variables

When imported, we will have a similar view, and still use all the filtering that were working before. Moreover, it is possible to add another dataset for comparison and show in the view, to have once more another variable in the application.

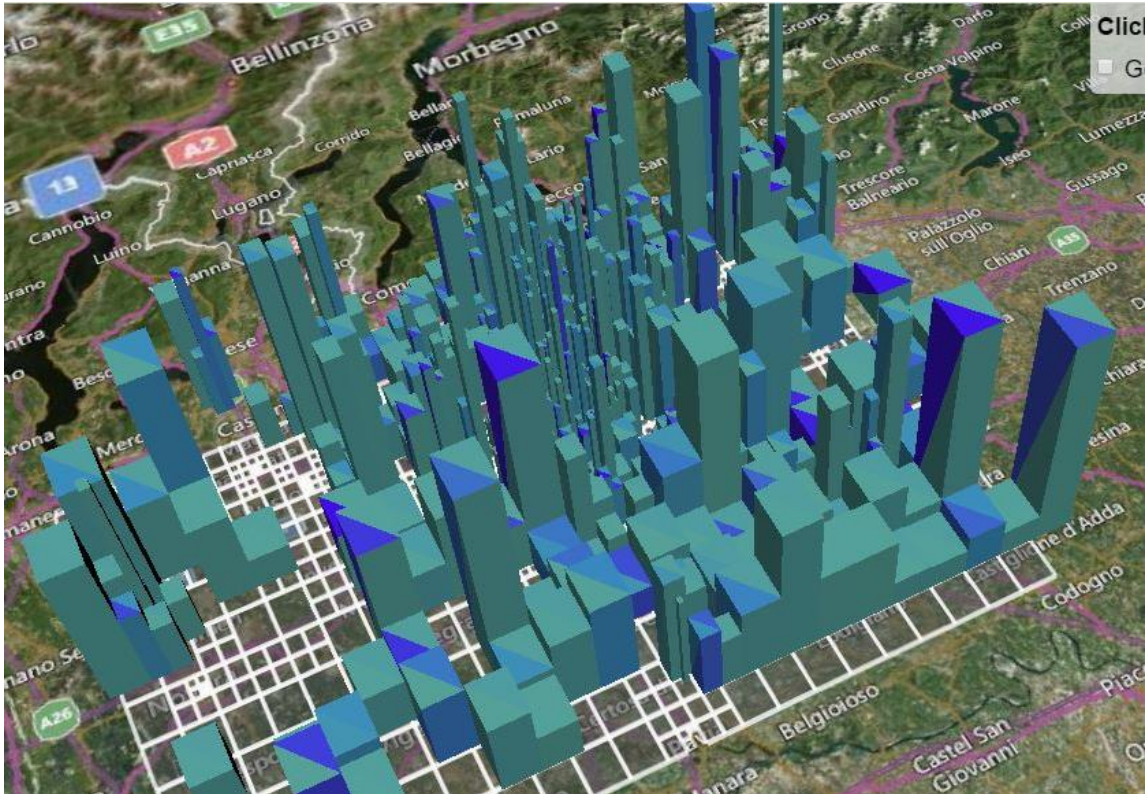


Figure 35 – Doxel Extrusion and Comparison with three variables

4.9 Multiresolution Grid Creator

Besides developing the application for importing and visualizing the data, we created a side tool, to generate a grid for visualizing georeferenced data.

The system permits to generate a grid that can be imported in the visualization application to be a point of reference the visualization of the data.

Thanks to this system any geo-referenced dataset can be visualized in the application, even if not based on a grid.

An interesting study case could be the weather stations that are represented by points. However, in our system, they cannot be visualized using a voxel model unless converted to a suitable format for the application.

4.9.1 Grid Typologies

Referring to a grid, we usually imagine a two-dimensional matrix with equally spaced cells. That is a regular grid. Although there are several kind of grids.

The cells in a grid can, for instance, have different dimensions among each other. Also, it is possible to have non-squared cells.

An unstructured grid or irregular is made by cells with simple shapes: triangles, rectangles and so on.

A particular example of a grid is the grid having a Quadtree structure, which is a 2D square divided into four squares. Some of them are in turn subdivided into other four squares and so on.

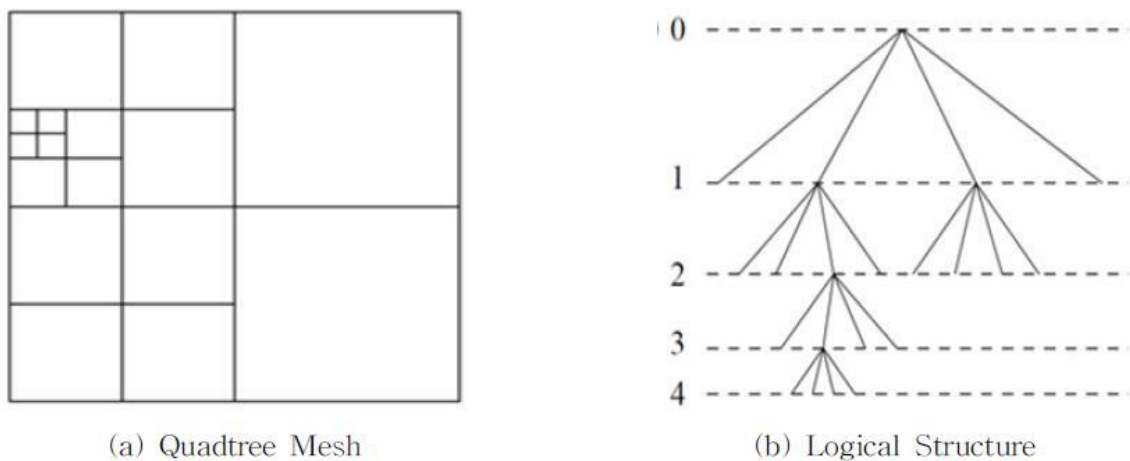


Figure 36 – Quadtree structure explanation [39]

Another sample is the subdivision following a Voronoi diagram structure, which subdivides a space into several polygons, as shown below.

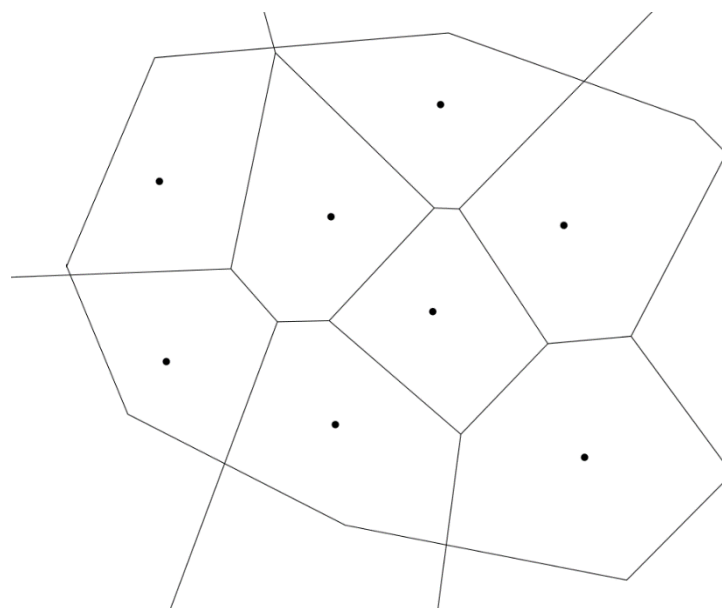


Figure 37 – Voronoi Diagram structure

4.9.2 Voronoi Diagram

The Voronoi diagram is a way to partition the Euclidean space thanks to points. The subdivisions are based on the distance between the points.

The partitioning of a plane with n points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to

its generating point than to any other. A Voronoi diagram is sometimes also known as a Dirichlet tessellation. The cells are called Dirichlet regions, Thiessen polytopes, or Voronoi polygons. [40]

4.9.3 Quadtree

A Quadtree is simple data structure where there is one parent with four children. Each child can have 0 or 4 children in turn.

This data structure is used in many studies area, such as computer graphics, game development, computer vision and for visualization.

It was created by Raphael Finked and J.L. Bentley in 1974."Quad Trees: A Data Structure for Retrieval on Composite Keys". [41]

To insert data in a Quadtree, we start from the root node and determine which of the four quadrants contains our point. Then we continue until we explore all the children in that quadrant and so on. Although it is possible that we want to define a maximum number of elements that a quadrant can have. In that case, we could split the quadrant itself among four children.

In our case, we needed to generate a grid having some points in the space. A Voronoi diagram representation is the one which fit most with the scenario, but the visualization of the shapes generated from the Voronoi is harder to visualize rather than the Quadtree one with rectangles.

Moreover, the space occupied by the cells of the Quadtree is proportional to the distance between the points, meaning that if we have for instance four close points, they will have a small area, rather than a big one from the Voronoi.

We can see an example in the two images below.

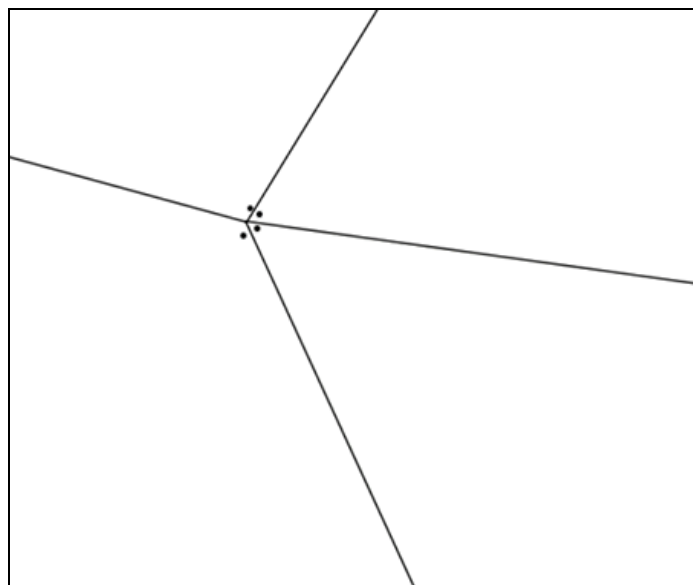


Figure 38 – Voronoi Sample with four nearby points

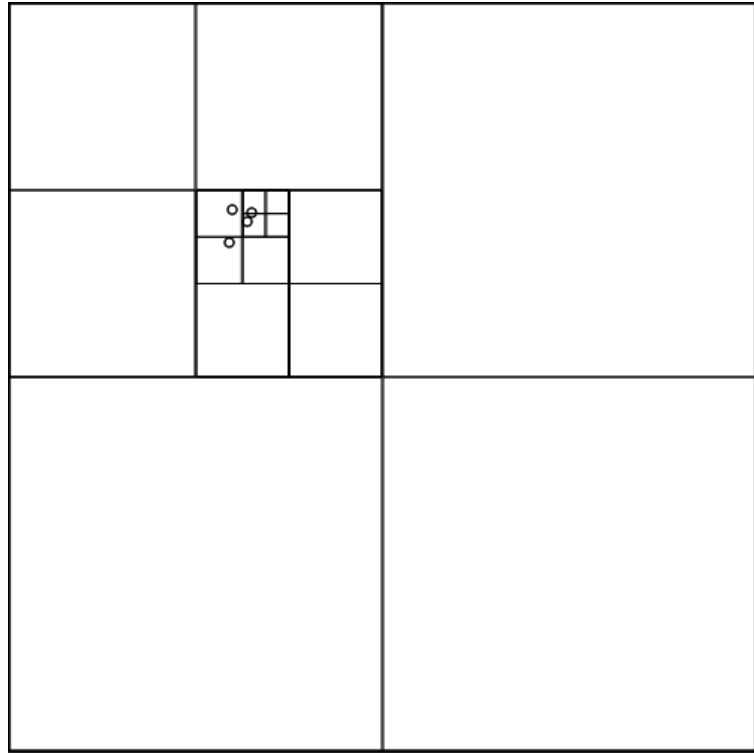


Figure 39 – QuadTree sample with four nearby points

In our case, since we wanted to represent with the voxel, only the areas of the grid occupied by them, the Quadtree is the most appropriate.

4.9.4 Quadtree Grid Generation

To generate the Quadtree, we followed a JavaScript implementation of Quadtree developed by Mike Chambers²⁴ and released under MIT license.

In our case, we wanted to set the maximum number of element in each child to one. So that we could generate a grid, from a Quadtree, made of cells that are smaller where there are more points and larger where fewer points are available. In this way, we show a high concentration of data with a better resolution, While the areas where we have less data will be larger. Looking at a grid like this we know that we can present a dataset without a fixed resolution. Just looking at a sample dataset represented in a similar grid, we can understand where is the best resolution and which are the relative points.

To create a grid with this structure, we start reading the data from the dataset to create the initial rectangle. We look for the maximum latitude and longitude and as well the minimums. Thanks to this four information we can define the original rectangle. To do so, we take the maximum longitude and minimum latitude to obtain the top-left corner, then the maximum latitude and longitude to get the top-right

²⁴ <https://github.com/mikechambers/ExamplesByMesh/tree/master/JavaScript/QuadTree>

corner, and so on. Creating a rectangle in this way, we can be sure to have all the points inside it.

Then to insert the points, we take all the points from the dataset in consideration, and we insert them one by one. To add a point, we follow the algorithm of the Quadtree insertion, thus, after inserting one point verify if another point is present in that quadrant. If so we split the quadrant in four more. If the new quadrant still contains another point, we keep splitting it until we have exactly one point in the desired quadrant.

After all the point have been inserted we can generate the grid. To do so, we retrieve all the quadrants and produce a GeoJson file from it. In this way we will have, possibly, more quadrants than the number of points, since each insertion of a point could generate four quadrants.

Below we can see a Quadtree grid that we generated thanks to this system, in the case the QuadTree is almost fully populated, so the points are well distributed over the grid.

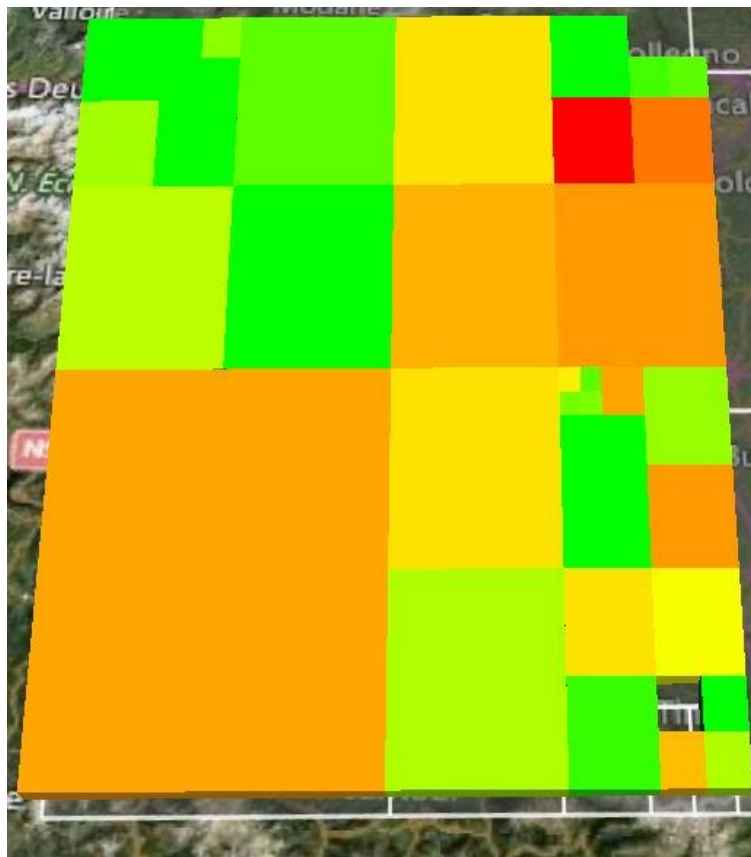


Figure 40 – Fully populated QuadTree

Another sample instead, that happens more often, when there are several points not well distributed, and in some case some close each other, will generate a similar grid, as the one in the image below.



Figure 41 – Real example of QuadTree from points

The above image represents a sample of a QuadTree generated by the precipitation over the city of Torino, recorder by some weather station from ARPA Piemonte²⁵ (Agenzia Regionale per la Protezione Ambientale) – on the 3rd May 2016.

The grid can be furtherly exported in GeoJson, thus, can be used for our system where a grid is needed. In this way, with a grid, and the data, related to the grid, we can generate the voxel model as mentioned before.

This method shows an alternative way to import data in the application and visualize them, feasible for greater study cases.

4.10 Development Environment - WebStorm

In our case, to develop the application and furtherly to create the tests we used the same environment, which is WebStorm²⁶ from JetBrains.

WebStorm is an IDE (Integrated Development Environment) and it provides several functionalities. Inside it, there is a big set of tools oriented to programming and allow supports of different web technologies. It is available for Mac and Windows.

It supports client side and server side languages such as JavaScript, HTML, CSS and similar ones. Besides the programming languages, also major frameworks are

²⁵ <https://www.arpa.piemonte.gov.it>

²⁶ <https://www.jetbrains.com/webstorm/>

supported, such as Angular JS, React, and Meteor. Also, Cordova, Ionic are supported for developing mobile solutions. Nonetheless is available a developing environment for Node.js as regarding the server side.

The interface is one of the points of strength of WebStorm. It is quickly possible to create new projects, import other projects and manage them

We can also find supports for several key combinations to speed up each operation. Moreover, it supports functionalities for finding classes, methods and files through shortcuts. The code completion for all the languages client and server side speed up the coding and simplifies it.

One of the best features, and well implemented is the orthographic corrector, which allows checking comments, string and anything helping to reduce errors. Several inspections are automatically performed on the code and when an error or problem is found the IDE provides a detailed explanation of the error and some quick fixes to solve it.

WebStorm moreover allows some features for debugging client-side code and has many features for it, such as supports for breakpoints, steps and expression evaluation.

Another important feature is the capacity to support unit testing; it integrates two famous test runners: Karma and Mocha. It is thus possible to run inside the IDE all the tests and set up them easily.

Spy-js is another interesting feature to trace the JavaScript code. Thanks to Spy-js it is possible to explore the files, their connection and identify performances issues. As well is possible to solve them and to determine bottlenecks in the code.

Many tools are available for WebStorm; it is possible to integrate it easily with the most popular tools for web developing. Some examples are the code quality tools, like JSHint, ESLint, JSCS or JSLint. We can add these inspectors for the code and get on the fly all the suggestion from the linters.

Project templates are supported as well; we used the Express generator on the server side to create the template for our simple server.

WebStorm is based on the open-source IntelliJ platform; it is then a customization of it. It provides so support for VCS – Version Control Systems – for working with the most popular versioning systems like GitHub, SVN, and Mercurial. In this way, having a local history is possible any time to roll back to a previous version of the code and navigate through all the different versions. Another point of interest for WebStorm is the opportunity to customize it, changing colors, fonts, themes and placing all the components in the place we want to.

WebStorm also provides a built-in terminal to run command line instructions without going outside the IDE.

Besides all the features and functionalities, thanks to the support of plugins available in the IDE Plugin Repository is possible to extend it and integrate any functionality we want which is not included by default.

4.11 Unit Testing

Making unit-testing means creating some tests to make sure the code behaves correctly.

The word "unit" refers to the unit of code that we have to test. When we want to test a specific part of the code is significant to test the entire class piece by piece, to do so, our code has to be divided into small pieces.

In general, but especially in JavaScript is problematic to have always a standalone piece of code. Often the methods call other methods, and they interfere with the DOM. Especially in client side projects, like in our case, might be difficult to break the code in the necessary pieces to test.

4.11.1 When to test

Having a client-side JavaScript, the project would be more efficient if we manage to test all the method one by one during their creation. Creating the tests side by side with the functionalities would also assure better performances the absence of bugs.

Doing the unit tests in the end, after having all the code, will instead be difficult to achieve. Moreover, breaking significant parts of the code and several classes into pieces is sometimes one of the most arduous tasks for a programmer.

4.11.2 Why testing

Writing the code, having in mind the testability of it, it favors the creation of a modular design, which is the most suitable for this kind of application. Modular design, as already mentioned, is a design style that allows having several modules that can be reused without altering the code consistently. It also reduces the complexity of the code, because having several part separated we do not need to have a long code which can cause some problem to be read and understood.

Testing the code and creating proper unit tests gives various advantages. One of the most important is to avoid the code to have a bug in the future, moreover, when we alter the code or implement new functionalities, the tests permits to follow the proper structure and keep coding in a modular way without making it complex. It also facilitates the code to be integrated everywhere. As well, any modules are easily importable in other projects.

4.11.3 What to test

When it comes to which part of the code we want to test, depending on the content, it is possible sometimes to test a high percentage close to 100%, even if when we want to create tests for JavaScript is hard to achieve this level.

It is not useful to test just the main behaviors of all the modules in our code, but sometimes it might be necessary to test unexpected behavior or conditions that should not be expected.

To test the code, we start from the highest level, and we go then into details.

To do so, we take a class in consideration, and inside we choose the first method. From this method, we create an implementation of it, inserting an input and getting an output.

4.11.4 How to test

The test consists of creating a proper input and expecting a specific output, in a way that we can predict what the output should be, and check if the condition is verified. The input of the method we want to test can be created, as we want or also imported from other classes. In case, we want to take the test individual and not being dependent on the other classes we can create a mock object that simulates the behavior of a complex object that is what an output of a method from another class could be.

After having, the input we know what is the expected behavior of the method. Also, we know what the output should be. We then compare and assert they are equal, the expected output with the real one. If the assertion is true, the test is passed. The assertion we do can be made just on some property of the output or even in many aspects, taking into consideration a wide knowledge of our output.

4.11.5 Testing Environment

In order to create our tests we set up our testing environment using the testing framework Jasmine²⁷ within the environment of Karma²⁸.

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests. [42]

Thanks to Jasmine is possible to set up all our test with a good and clear syntax for the testing code. It allows describing each test and create for each method a testing file with the description of many kind of tests. A particular aspect of Jasmine is that it doesn't invade the application or the integrated development environment. The syntax used to create a simple test is the following:

```
describe('Hello world', function() {
  it('says hello', function() {
    expect(helloWorld()).toEqual('Hello world!');
  });
});
```

Code Snippet 6 – sample test structure in Jasmine

We can see that the “describe” part is useful to indicate the general functionality to test, while with the “it” part we can define the exact part we want to test.

Jasmine also implements several pre-defined matchers to use in the code. The example matcher used in the above example “toEqual” is used to guarantee the equality condition. Some other matchers are: “toBe”, to represent the exact equality condition, “toMatch” that call a regular expression, “toBeUndefined” to test the “undefined” condition and much more.

²⁷ <http://jasmine.github.io>

²⁸ <https://karma-runner.github.io>

To implement the test environment of Jasmine, we used Karma. Karma allows to test the code in any browser and doing it in multiple ones. It allows testing during the development and has the possibility to track any change to the code, to run the require tests when a class is edited.

Also, setting-up the environment is very easy thanks to Karma, because it simplifies the creation of the configuration files thanks to some simple questions from the command line, and gives out a working environment, ready to work.

Moreover, it allows the use of RequireJS²⁹ to call external dependencies easily.

RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code. [43]

4.11.6 Tests

We created some tests to reduce the possibility of having bugs in the future. We created a specific folder “test” where we placed all our tests. We followed the paradigm of starting from high-level tests going into details. A sample test is provided below.

As we can see in the snippet below, the testing of a method is given. In the output, we take into consideration the type of the output and the result of it. In this case, we are testing two properties of the method. The more we test, the better it is, because it allows identifying bugs, and avoiding the unnecessary behavior.

```
describe('Color test', function () {
  it('Color white', function () {
    var color = GlobeHelper.getColor(0, [[255, 255, 255], [0, 0, 0], [0, 0, 0]]);
    expect(typeof(color)).toBe("object");
    expect(color).toEqual([255, 255, 255, 255]);
  });
});
```

Code Snippet 7 – Testing two properties of an object

It shows the testing of the class “GlobeHelper”, showing a test for the method “getColor” to verify a sample conversion of color from Hexadecimal to RGB. We created an assertion of the type of output, in our case an object and then compared the expected output with the real one.

Running the test, the IDE shows that it passed successfully, and no problem was found.

²⁹ <http://requirejs.org>

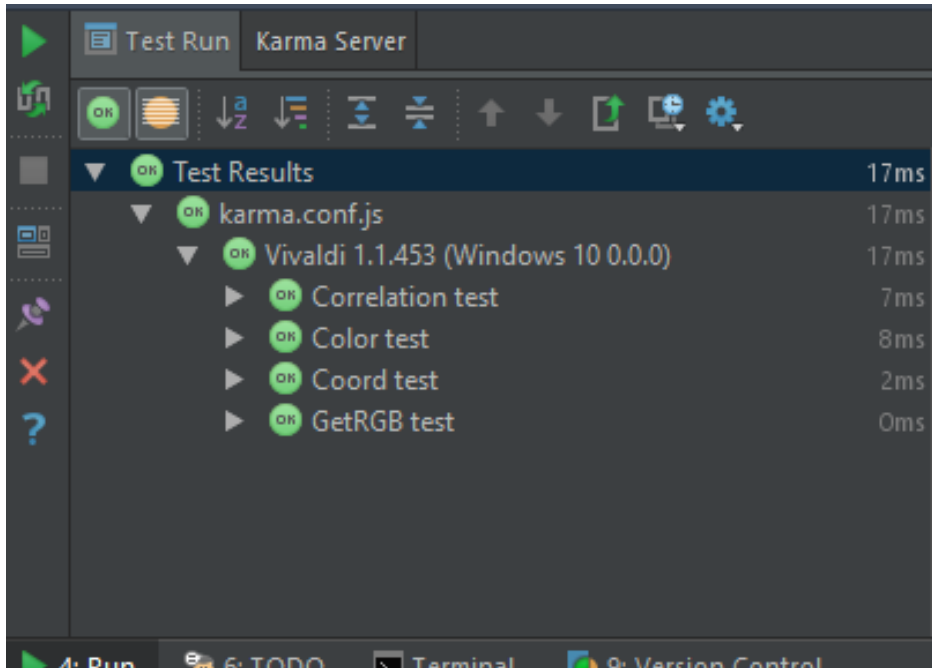


Figure 42 - Testing running successfully

4.12 Versioning System

The versioning is the management of different versions of accurate information. In software development, it refers to the changes in the versions of software which is in the development phase.

The version control system - VCS - allows keeping in memory the changes made to any file, to be able to go back any time to a previous version, or check which changes have been made.

When we "version" a file it means that we are going to create a copy of that file each time we make a change, and we commit this change to the system. These changes will be stored in the system we want to use. We will use the term "repository" to refer to the group of files for a specific project.

4.12.1 Local Version Control System

It is possible to create a version control even manually copying and pasting the file before doing any edit, but this technique is time-consuming, very prone to error and occupies additional memory.

Using a VCS, we have a database to keep track of all the single changes made to a file, in this way we do not have to make a hard copy of the file, but just of the applies changes.

4.12.2 Centralized Version Control System

A big leap from the local VCS is the centralized one, which favors the collaboration of more people on the same project. This consists of a server that hosts all the files related to a project, and the users can connect to it to download and upload changes.

4.12.3 Distributed Version Control System

The advantage of a Distributed VCS is that the client makes a copy of the repository, so that if a server goes down, it is still possible to work on the files and commit them when the server is available. In the same way is possible to create a copy on another server if necessary.

The VCS is mostly used in software development. It has several features, depending on the system used to support the edit of the files. Some systems use a technique to lock the files that are being edited from other people so that one developer per time can make changes to the file. Another structure, especially on the distributed system, allows many people committing the changes to the same file, and if there are no conflict among them, is possible to merge them easily. In the case of conflict among some lines from the previous and the actual file are present, the supervision of the user who committed for last is necessary.

4.12.4 GitHub

GitHub³⁰ is a website which provides a service to host repositories. The name "Git" comes from a VCS called Git developed in 2005 from Linus Torvald.

The website provides social network functionalities, allowing users to check the development of the projects.

It provides support for free accounts for public repositories and private repositories for a fee. Mostly, open source projects are hosted on public repositories, such as in our case.

The website supports many features for each repository, an example in the documentation that is possible to provide for each project showing much information about it. In the case of collaborative projects, there is an issue tracking system to keep people updated on the bugs, improvements or requested feature on a project.

Our project³¹ was thus hosted on GitHub, and all the changes to versions, are kept updated on this system.

4.12.5 SourceTree

SourceTree³² is a free Git client for Windows and Mac. It simplifies the handling of repositories in GitHub. Instead of using Git directly from command line, it provides a graphical user interface to manage all the repositories, hosted or even local.

It can be connected to GitHub to manage all the projects, without the necessity to go to the website. It supports different branches of the same projects, visualizing all the versions of a file and comparing files in case of conflicts. Moreover, it provides support for external plugins to extend the features, such us conflict management tools.

³⁰ <https://github.com>

³¹ <https://github.com/GabrielePrestifilippo/EST-WA-Javascript>

³² <https://www.sourcetreeapp.com>

Chapter 5

Case Studies

5.1 Milan: Telecommunication Data

In our study case, we wanted to address the visualization to telecommunication data and social media data.

“Global telecommunication services create an enormous volume of real-time data” [44]

Visualizing these data is becoming a significant challenge, especially when we have several variables to represent. The use of a 3D visualization can ease the user's understanding of the data and their interpretation. We took into consideration an important dataset: Big Data Challenge 2015³³ from Telecom Italia. The data provided are available for few cities in Italy: Bari, Milan, Naples, Palermo, Rome, Turin and Venice. There is information about telecommunication events: SMSs, calls and internet usage. Apart from telecommunication events some data about other fields such as demographic information of people living in those areas.

Referring to telecommunication data, all the entries have a timestamp expressed in milliseconds, and each one represents information for 15 minutes from that timestamp. Therefore, each entry refers to 15 minutes of events in the time interval. The data refer to the year 2014, and we have 2 months of data for this year, March and April. In particular, for the SMSs, we have information about the number of received SMSs, sent SMSs, and country code of the sim card used. Regarding the phone calls, we have a number of incoming calls, outgoing calls, and country code.

Regarding the internet data, we have the number of started connections, closed connections and amount of data transferred. The social media dataset contains geolocalized tweets originated from Milano. Each entry has an anonymous user identifier; thus, different entries could have a repeated user-ID in the case of repeated tweets.

Telecom Italia provided a Geo-referenced irregular grid, for some cities in Italy, where an ID represents each grid cell. An irregular grid means that each grid element has a different size. The concept of rows and columns for regular grids does not apply anymore, and more peculiar is to have a grid not circumscribed by a rectangular shape, but open to any geometric contour. In our case, the grid is made of cells varying from a dimension of 255 x 325 meters to 4080 x 5200 meters as shown in Figure 5.

³³ <http://www.telecomitalia.com/tit/it/innovazione/big-data-challenge-2015.html>

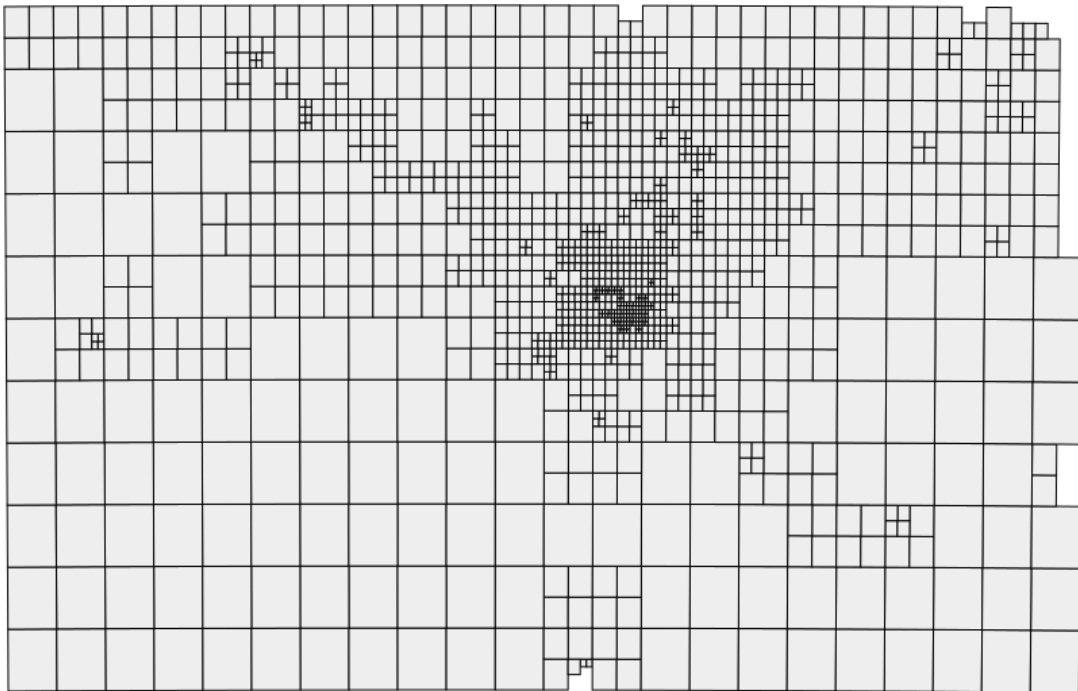


Figure 43 - Milano Grid, BigData Challenge - Telecom Italia

In the data, each entry has a grid-ID number to link the value of the data to the corresponding grid cell, a timestamp, and value for each of the mentioned data. A singular fact is that there are several “holes” in the data. This means some data are missing, in particular, in each timestamp some values for the grid elements are not present. The case study presented considers the city of Milano, and we show data about call-in and call-out. However, since all of the data has the same structure and refer to the same grid, we could work with all of the data in the same way.

As we mentioned before, the data have no information on the altitude, and we could exploit altitude (z-axis) to represent the time. This permitted the user to view an accessible three-dimensional representation of the data within a time range. In this way, four dimensions are represented for a specific meta-data: value, latitude, longitude, and time. To do so we imported some data in the application from a CSV file, selecting the right parameters from the configuration. In this case, we imported the outgoing calls from Milano, in the day 01/03/2014 at hour 00:00. We also selected the colors for the representation, as shown below.




Height Extrusion	<input type="text" value=""/>
Height Voxel	3500
Max layer shown	1
Initial Height	10000
X/Y Subdivisions	2
Z Subdivisions	2
Max layer	10
Statistical Index	Weighted Average
Voxel Min Color	
Voxel Mid Color	
Voxel Max Color	

Figure 44 – Settings to import data in our study case

After importing, we obtained the following representation with the doxels. We can see that some doxels are missing since the data were not available. Moreover, having an irregular grid, we can see that the dimension of the doxels is not uniform. It follows the dimension of the blocks in the grid, with smaller blocks in the center of Milano, so with a higher concentration of data in the center.

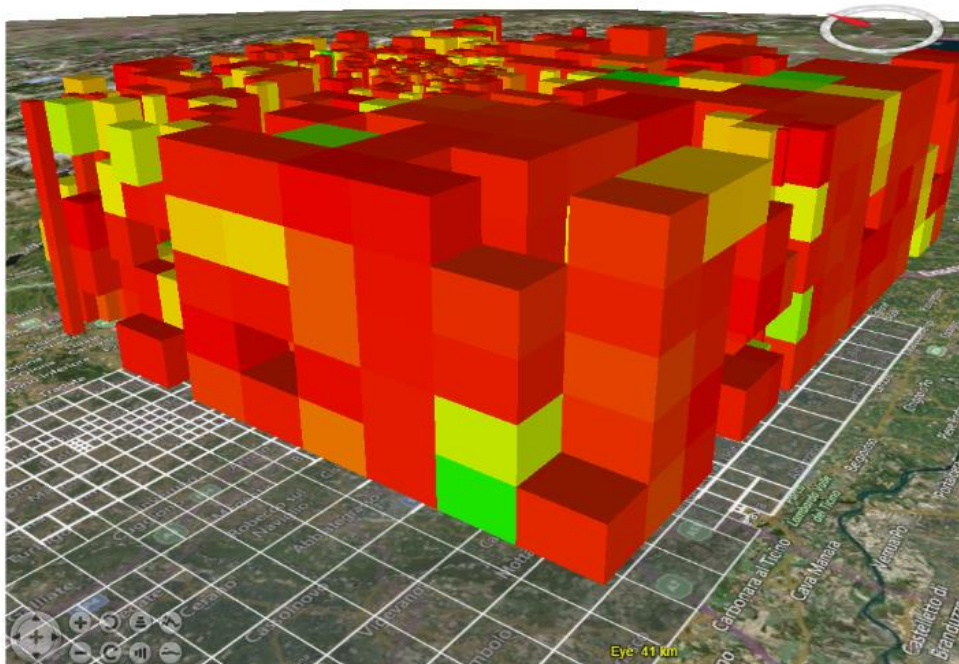


Figure 45 - Doxels representing outgoing calls over Milano grid

What is novel in this case is the irregular grid and non-continuous entries for each timestamp.

NASA Web WorldWind offers good performances to represent the significant amount the doxels; without the need of any simplification, and in any case not possible for our study case. Thanks to the web application we could observe the data and filters out doxels within a range in the three axes.

The sliders in the application interface permit us to limit the extension of latitude, longitude, and time that we want to show on the globe. We have the possibility to cut the model using three cutting planes and visualize slices of the original volume. We can thus observe fewer doxels reducing the complexity of the model.

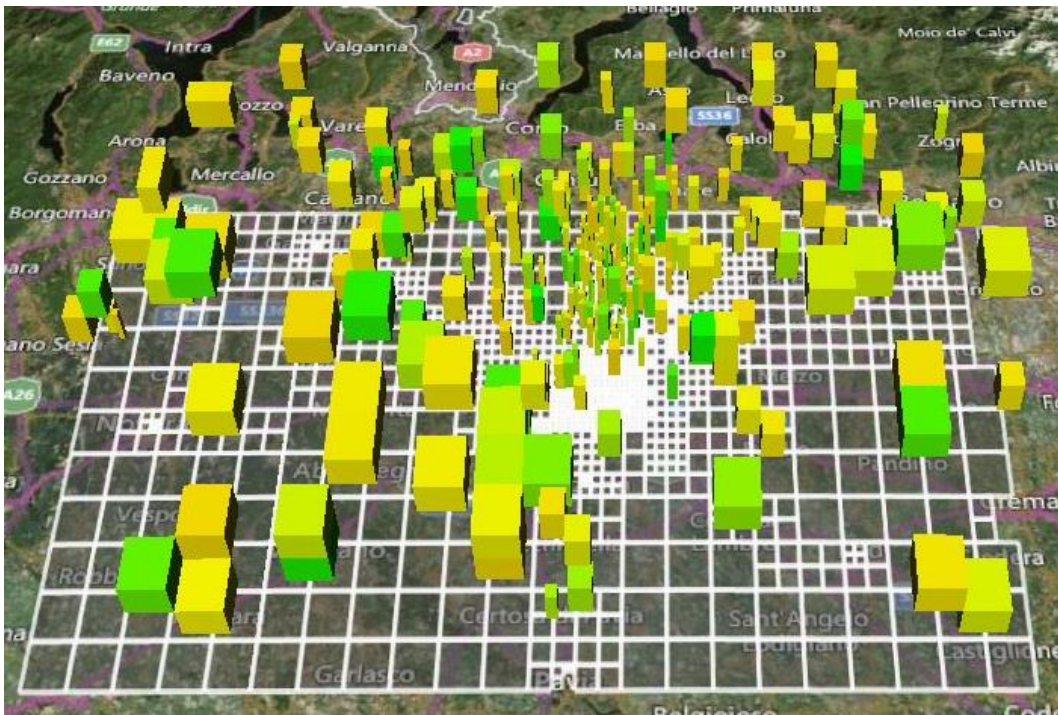


Figure 46 – Filtering out low values in our study case



Figure 47 – Values filtered out in our study case

Using an outgoing call dataset and filtering on the values, we could observe the number of calls that is increasing or decreasing in the X part of the grid. Applying the filter, we can see from the picture below the high concentration of calls, represented with the Y color.

We adopted a heuristic approach to face with a statistical representation of data – weighted average, variance, median, maximum and minimum. We could subdivide the data into subsets, choosing a number of groups to present the data; the grouping is possible for the altitude, longitude and time.

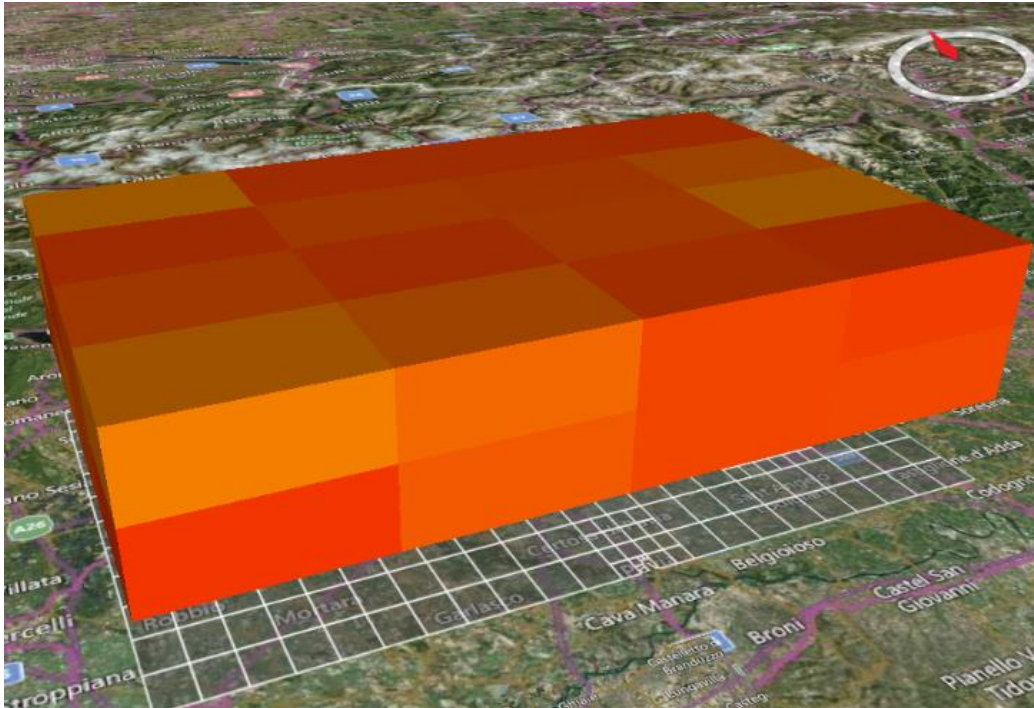


Figure 48 - Group representation of average outgoing calls

Moreover, performance improvements came in handy as well. Having a lower number of doxels to represent - since we put them in the subgroups - allows better performances on the virtual globe. When we want to show the doxels contained in a big doxel, so in a group, we could activate the appropriate click handler and click on a big doxel to see all the doxels it represents the statistic for.

Thanks to the click handler, any user can interact with the globe and the data. It is also possible to select a single doxel or a subgroup and obtain the information regarding the available variables. Clicking on a doxel, we could retrieve the information about the variable it represents and show time behavior of the variable for the selected doxel on a 2D graph. Although the chart also shows other variables available in the dataset. We can then compute the correlation between two variables. In our case, we compared the number of outgoing calls with the length of the calls.

Taking as an example the 3rd January 2015 at noon, we saw a high correlation is present in the dataset. While comparing the single doxels we noticed some doxels have a stronger correlation, while other not.

Correlation between datasets in selected voxel: 0.23129

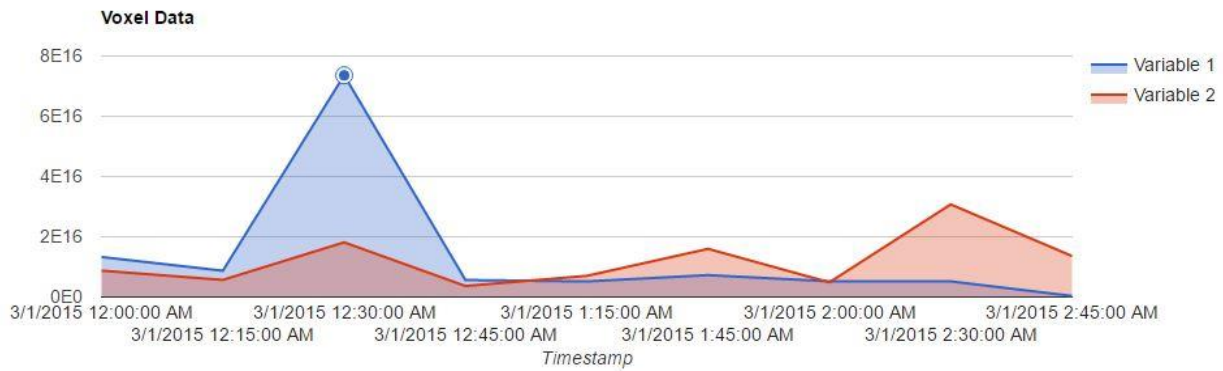


Figure 49 – Information about a single doxel in our study case

Forasmuch as we represented the time using altitude, not many layers of doxels can be appreciated because inserting more than a few, make it difficult to observe in a meaningful way and also begin to cause some slowdown of the globe performance. To solve this issue, we could keep visible a limited number of time layers for a fixed time range and, from the interface, move this range to animate the desired time range view.

We demonstrated the introduction of another way to show the time, but this technique can be as well used to handle another variable in the scene if needed.

In our study case, we tried comparing the dataset of outgoing calls with a demographic one, representing the number of people in a given area, thus in a grid block. We then imported the two datasets and obtained a representation of the two datasets in a split view of the doxels.

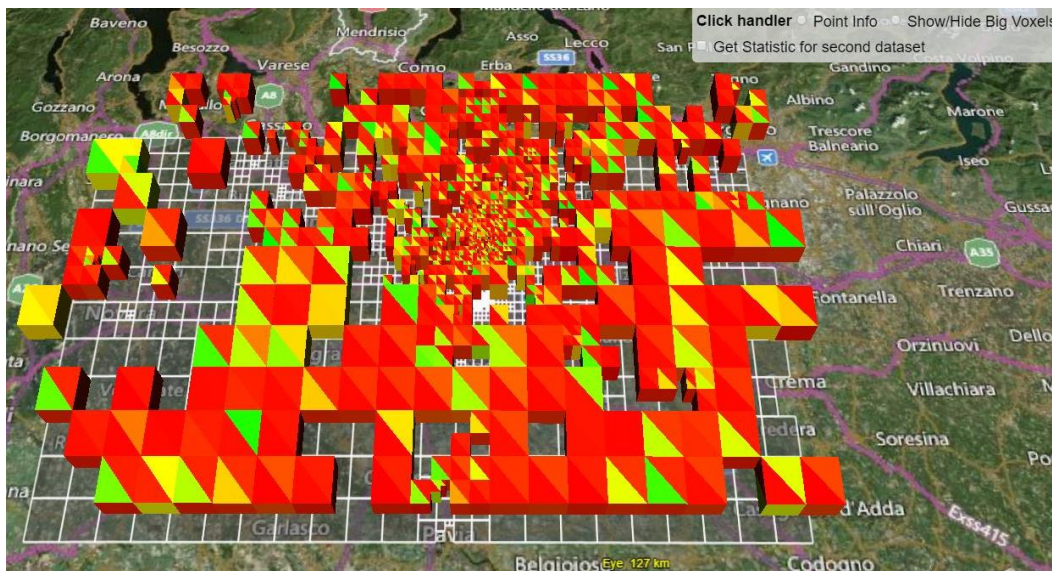


Figure 50 – Comparison of two variables in our study case

Using doxel extrusion, we then showed the same variables used in the first representation, the outgoing calls and the length of the calls. Extruding the duration of calls, while color represented the number of calls. We can observe in the picture below what one layer looks like. We again used 3rd January 2015 at 12:00:00 as sample day. We can clearly see the correlation of the observed variables

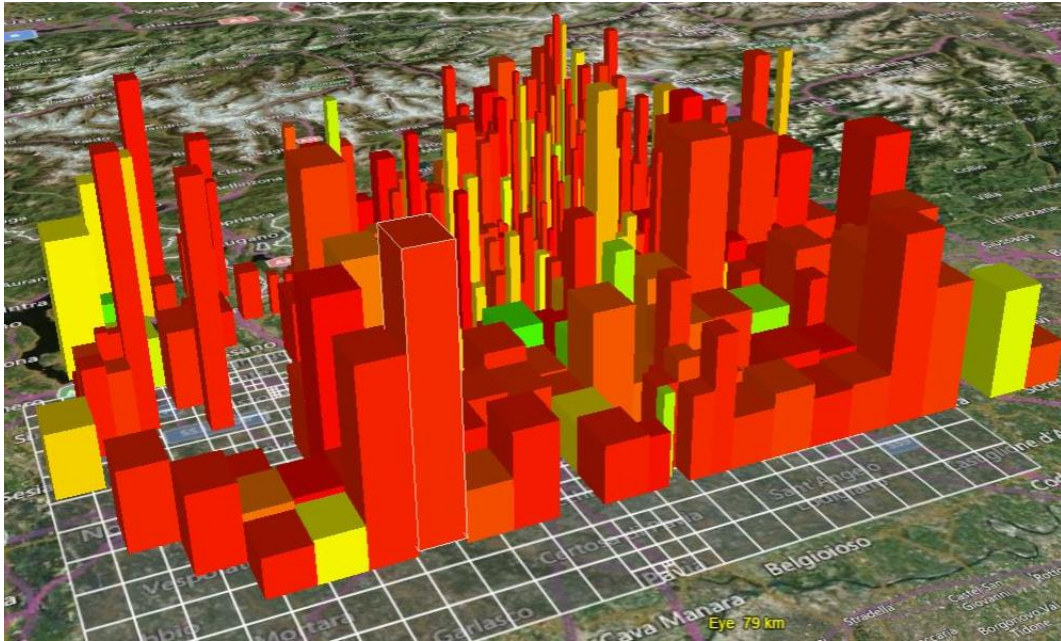


Figure 51 – Doxel extrusion in our study case

5.2 Turin: Point Features Data

Another case studies involves a different typology of the dataset. In this case we wanted to visualize a dataset made of point features. The dataset used comes from ARPA Piemonte³⁴ and is made by point data for about 100 weather stations in the surrounding area of Turin. This weather station data contains wind speed and precipitation, thus only two variables. The dataset contains information for a week and has a temporal resolution of one sample per hour. Below is a 2D map showing a possible representation of such data. Although this representation cannot show some information available in the dataset. Especially we cannot analyze the variation during the time and the values of the variables.

³⁴ <https://www.arpa.piemonte.gov.it>

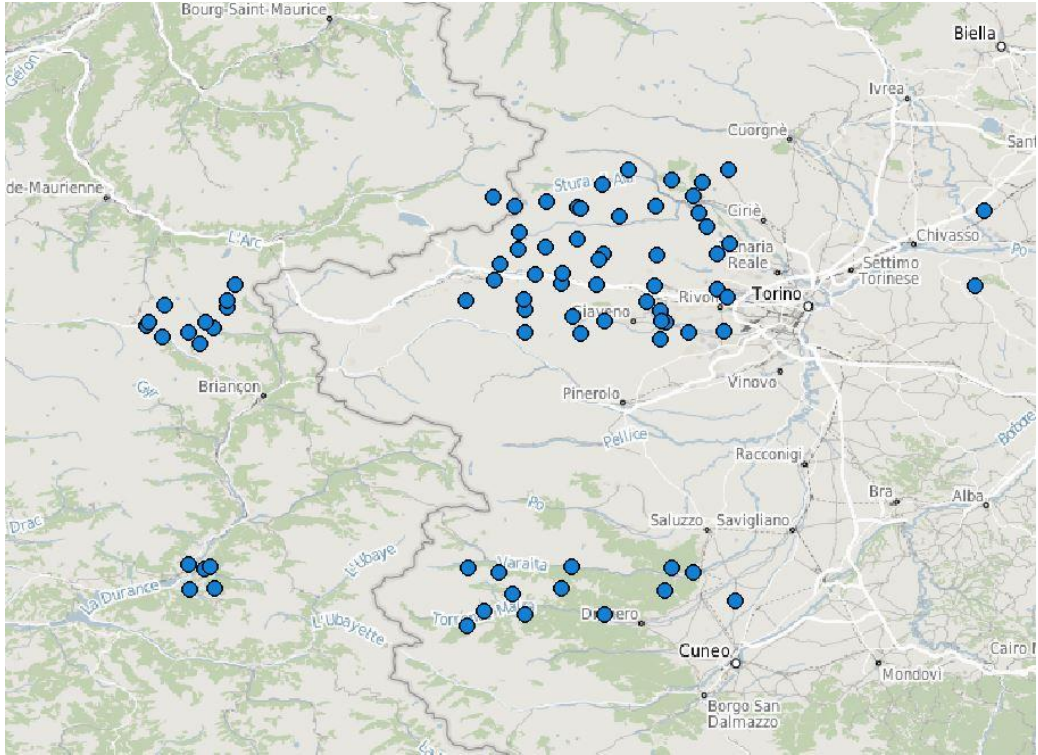


Figure 52 - 2D representation of point features

Thanks to our Multiresolution grid creator, we could import the dataset and create a QuadTree representation. In this case we set a maximum limit of subdivisions to 5 quadrants and obtained the following representation:

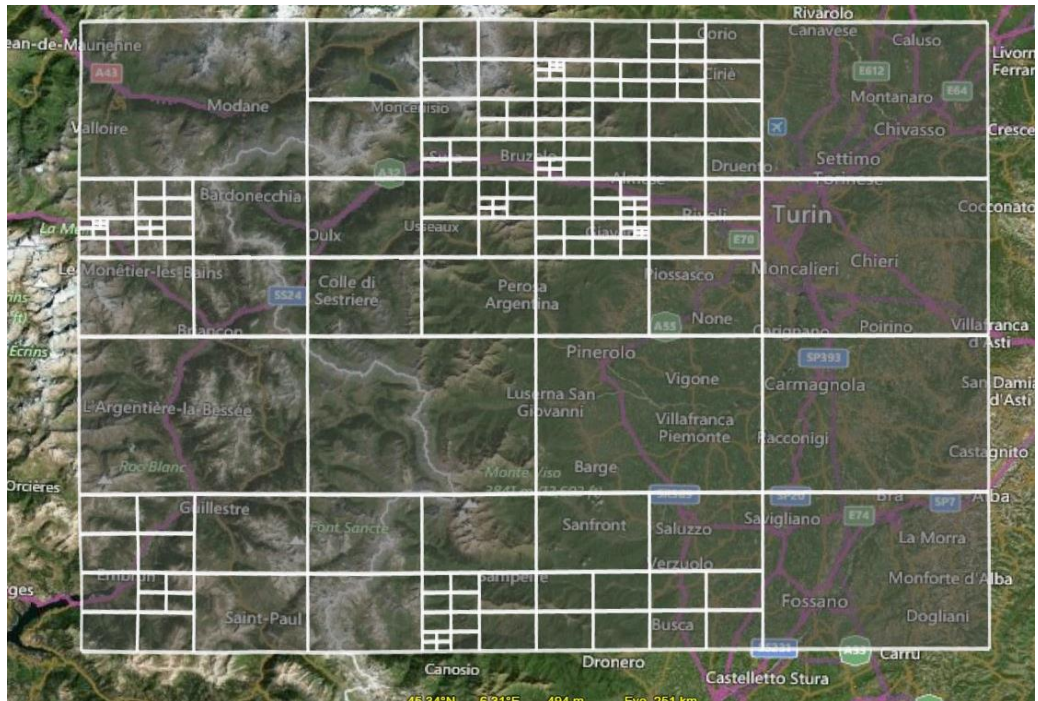


Figure 53 - QuadTree representation of Turin Dataset

Having the grid in the application and the dataset we could then import the data and display them using the voxel model.

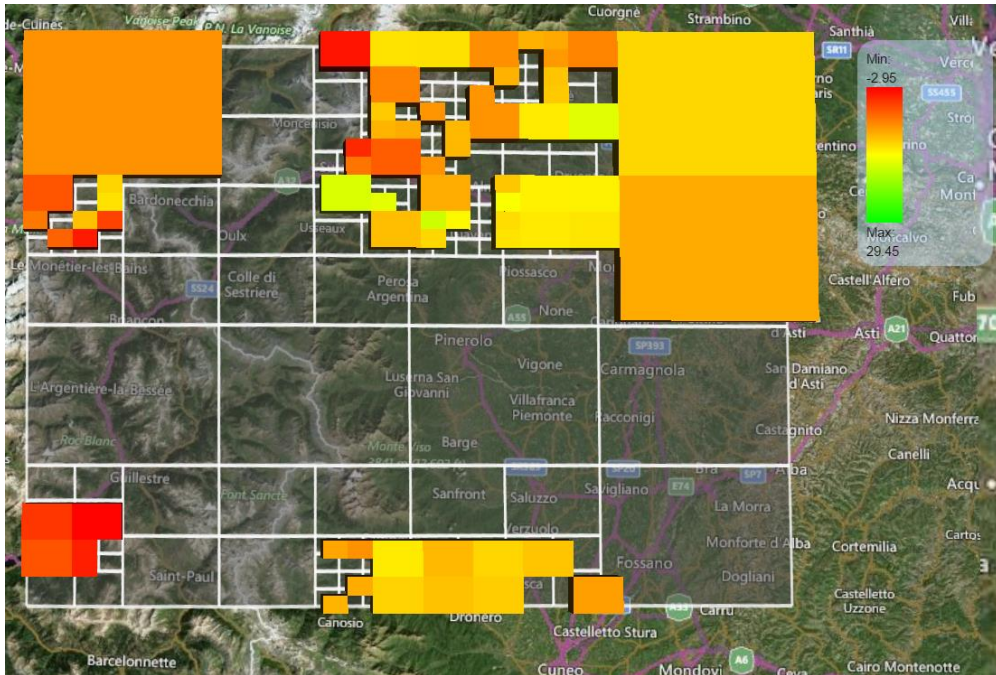


Figure 54 - Doxel Representation of Turin Dataset

We can see how a single layer of data is represented. However, all the representations shown before are possible to analyze and display the different data. Below is a representation in time of the dataset, showing the variation for a single day.

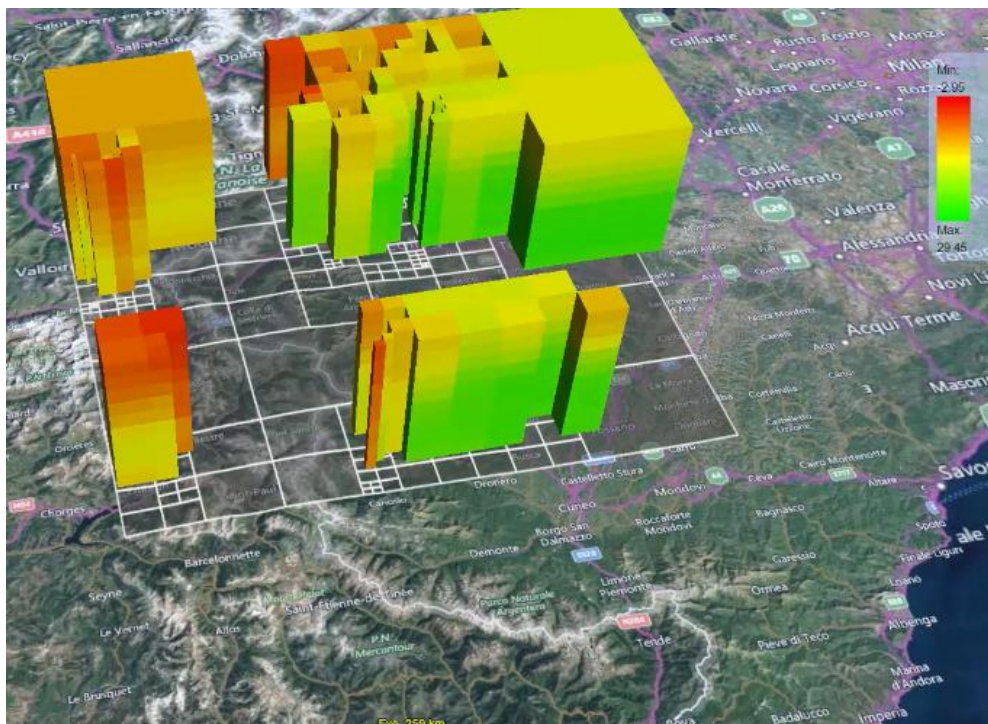


Figure 55 - Time representation of Turin Dataset

To also show the second variable we can represent the dataset with the extrusion representation and analyze the correlation of the two variables, wind speed and precipitation. We represented the wind speed with the color and the precipitation with the extrusion in the altitude.

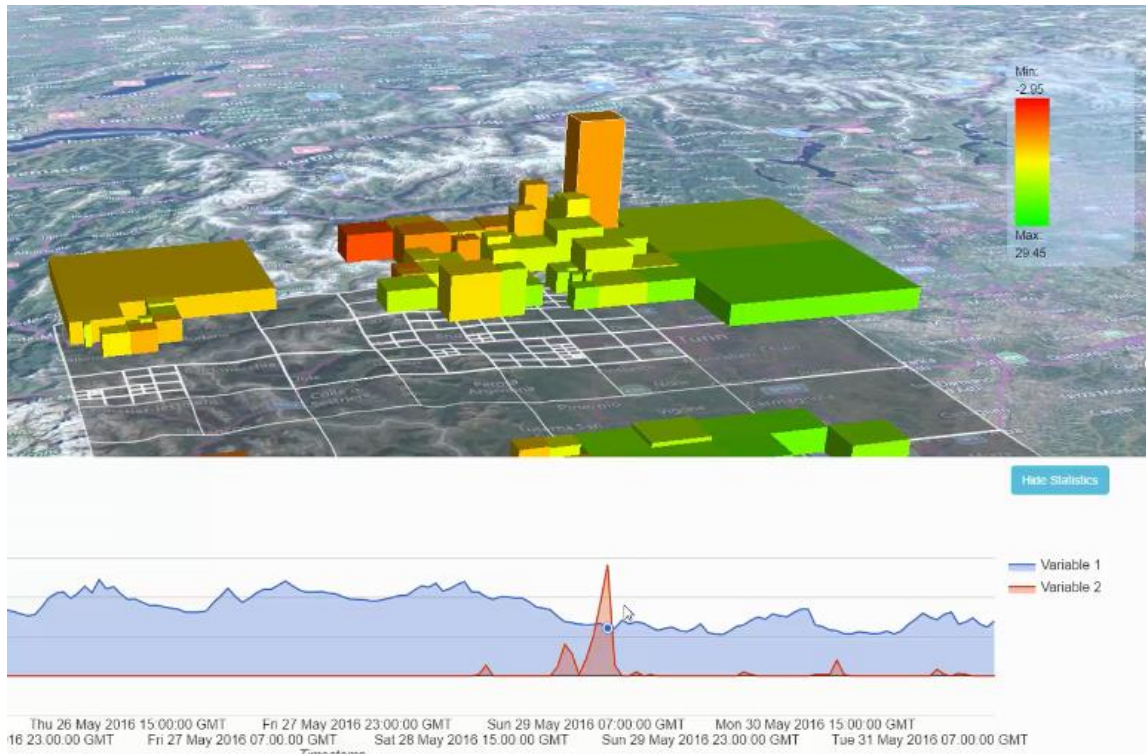


Figure 56 - Extrusion representation of Turin Dataset

We have seen how any dataset made of point features can be represented in the application we made and how the dataset can be easily analyzed to obtain information and view statistic from the dataset.

5.3 West Turrock: Geological Data

An interesting case study, to show the wide set of data that can be shown of the application, refers to a case study in which we wanted to visualize some geological data. From the British Geological Survey website ³⁵ we downloaded some samples of LithoFrame files.

LithoFrame is a concept to produce models of Britain's subsurface geology; the 3D equivalent of the geological map. LithoFrame models

³⁵ <http://www.bgs.ac.uk/>

have been created at a range of scales to answer specific questions about geology and the composition of the subsurface. The adjacent map of Britain shows our coverage of current geological models at varying resolutions. [45]

In particular, we retrieve a model representing bedrock geology in West Thurrock. The model covers about 10 square kilometers of area with a resolution of 10m. The bedrock of the area is dominated by the Chalk which extends to a depth of -150m under the ground.

The presence of Chalk shows the effect of excavation over a large area [46].

The LithoFrame was originally in an ASCII grid format, preventing us from importing it directly into the application. Thus we converted it to a CSV, in order to show that data in our application.

Since the data in an ASCII grid are equally spaced and the information represented by point data, we had to use the CSV importer for point feature data, as in the case study of Turin. During the importation we have chosen to represent two variables. The depth of the base for the chalk and the thickness of it.

We thus imported the data selecting a maximum subdivision of quadrants in our QuadTree to four. To import the two variables, we decided to use the extrusion model and we associated the color to the depth of the chalk deposit while the height for the thickness, obtaining the following representation.

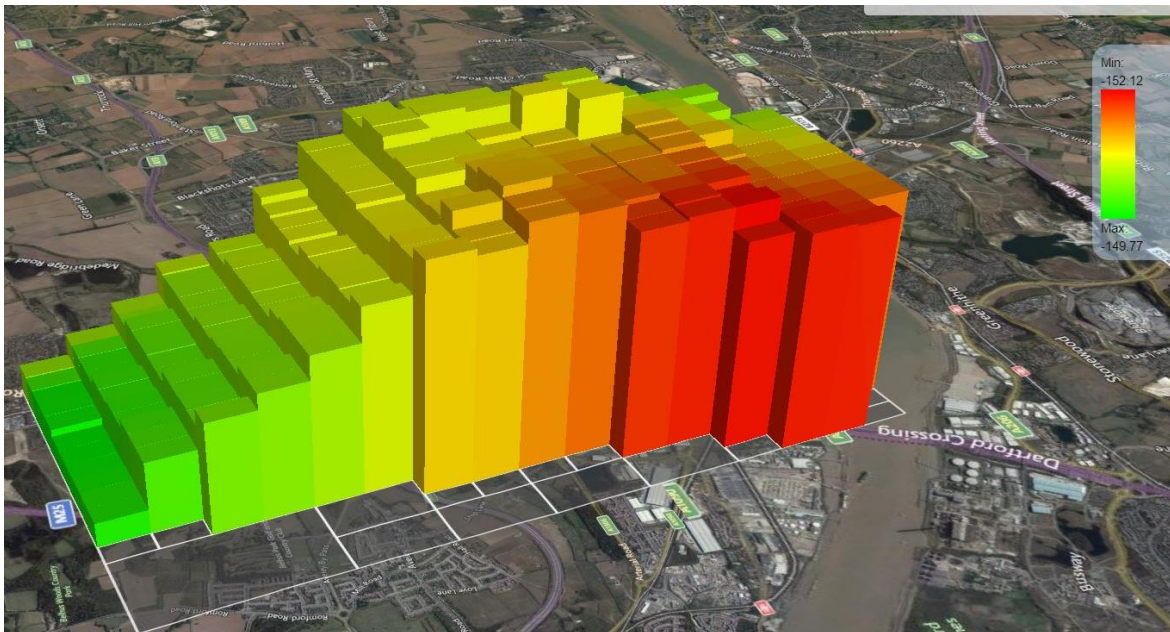


Figure 57 - WestThurrock: Chalk Deposit in 4 Quadrant subdivision

We can observe from the image that the depth of the deposit varies from -152.12m to -149.77m becoming deeper close to the river. In the same way the thickness

increases while going close to the river. The thickness, represented by the height of the doxels, is exaggerated on the extrusion order to appreciate the image from a distant view.

To have a more accurate representation, we can subdivide the quadrant in five or six sub-quadrants obtaining smaller voxels.

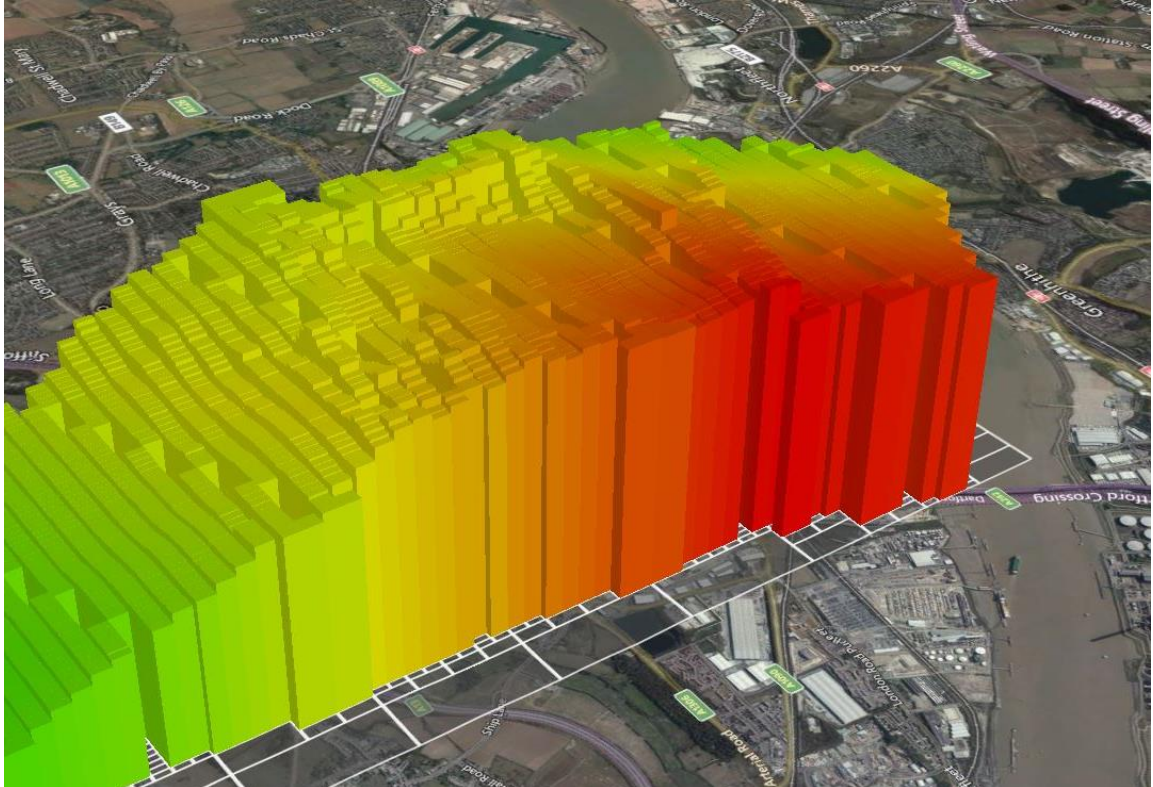


Figure 58 - WestThurrock: Chalk Deposit in 6 Quadrant subdivision

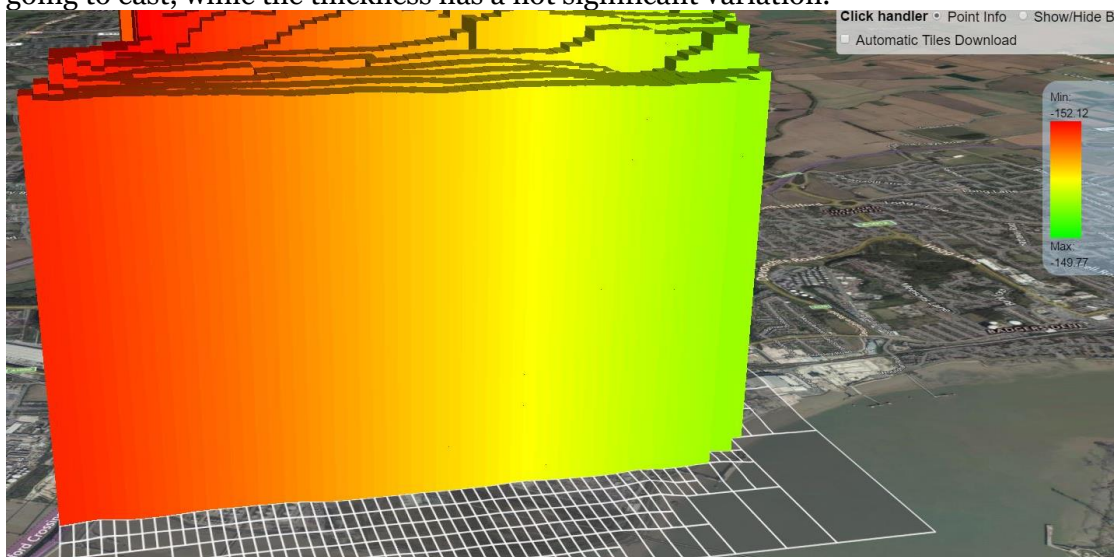
The representation with a subdivision in six quadrant reaches a resolution per doxel of about 10m x 30m which is the maximum we can obtain for such model. Having a better resolution, we can appreciate better the thickness of the chalk deposit, observing very small variation of height in the center of the model (the orange part). Moreover, we can retrieve the data from the doxels, as shown before, clicking on each doxel.



Figure 59 - West Thurrock chalk data

The figure above shows the value of the thickness of a single doxel, in this case corresponding to a single entry of the LithoFrame. For the Variable 2, the thickness has a value of 134.53m over a base of 150.82m.

Is also possible to observe the fine variation to the depth of chalk's deposit, varying from east to west (left to right) in the below image. We can see how it becomes deeper going to east, while the thickness has a not significant variation.



Thanks to this case study we witnessed the big opportunities available within the application we created for showing data available in multiple fields of research.

5.4 World Coverage: Average Land Temperature

This case study is different from the previous for many reasons. Firstly, because of the dataset which has a full coverage of the globe. We used a dataset provided by NASA for the Average Land Temperature during the year 2014. This dataset is accessible, along with some sample, from some the Big Data Standards website³⁶, using as sample Rasdaman. The dataset has a resolution greater than 10km.

In this case, we implemented a system using Rasdaman to store the dataset. This allowed us to access it using the Web Coverage Processing service, thanks to an HTTP request using AJAX calls from the application.

Though the coverage is available for the entire globe, we are not able to visualize it entirely in a single view. Thus, we use the application to query the dataset and retrieve data within the boundaries of our selected view. Selecting a view which includes almost all Europe, and specifying as date January 2014, we can retrieve a similar result.

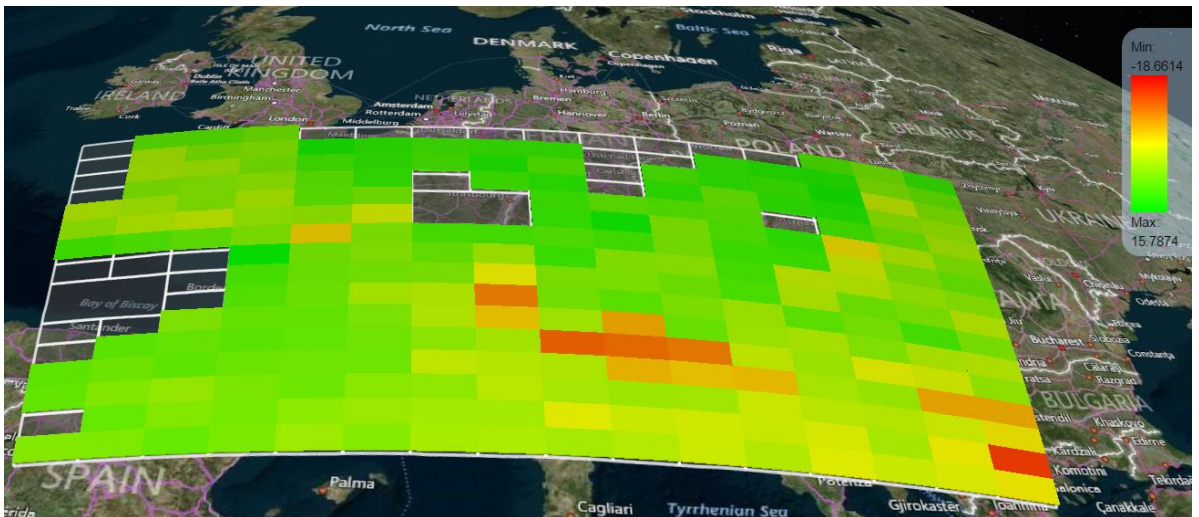


Figure 60 - Average Land Temperature, Europe

The above image shows the queried dataset, using a voxel model and the QuadTree creator. This means, we have an aggregation of values, in this case using a maximum subdivision in four quadrants. Though, if we want to query other part of the globe we can move around it and select an option to automatically download the new data. Just moving across the globe, and selecting the western part of Africa, in few seconds we automatically get the result for that view.

³⁶ <http://standards.rasdaman.com/>

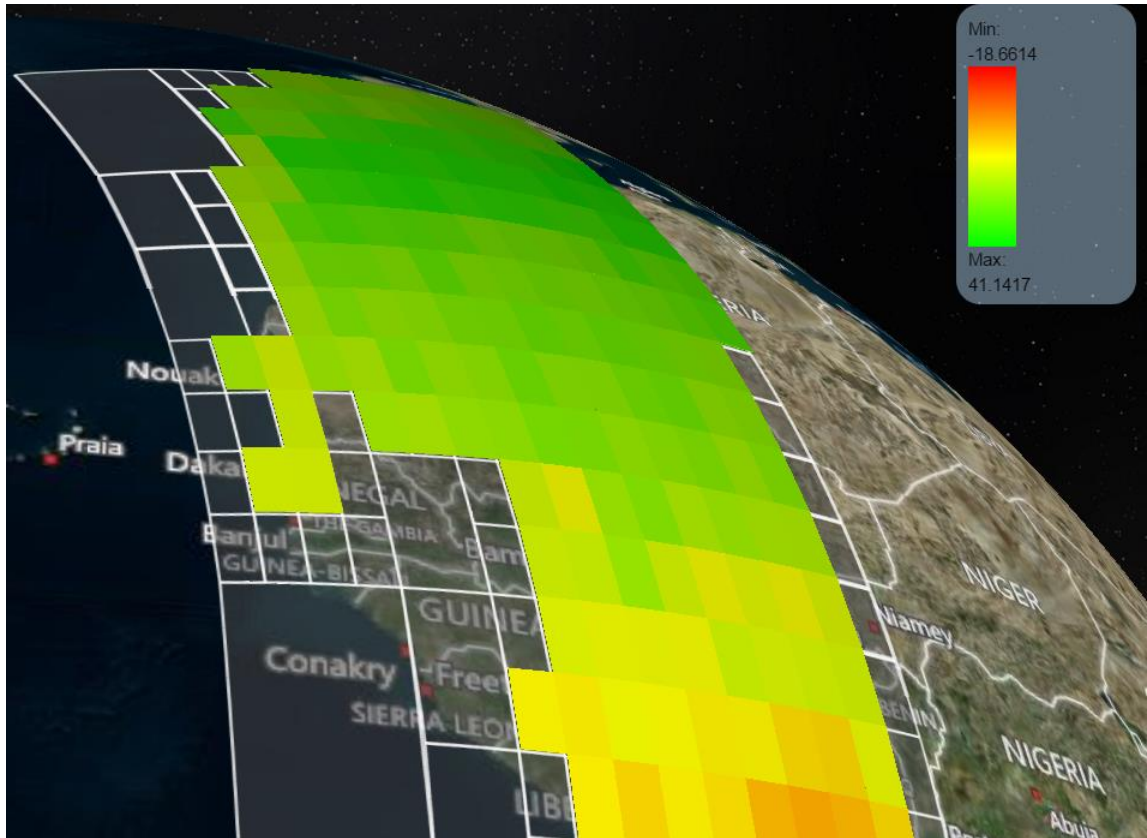


Figure 61 - Average Land Temperature, Western Africa

We can also notice how the color scale has recalculated the range, reaching now a maximum value of 41° , while before it was 16° .

Since the resolution of the dataset is higher than this, we can also zoom in some areas to retrieve detailed information, keeping still a maximum of four quadrant subdivision in the QuadTree. In case we also want more detailed information we can increase the number of the subdivisions, obtaining more voxels. In the image below, we zoom-in further to the north part of Italy and increase the subdivision of the QuadTree up to six.

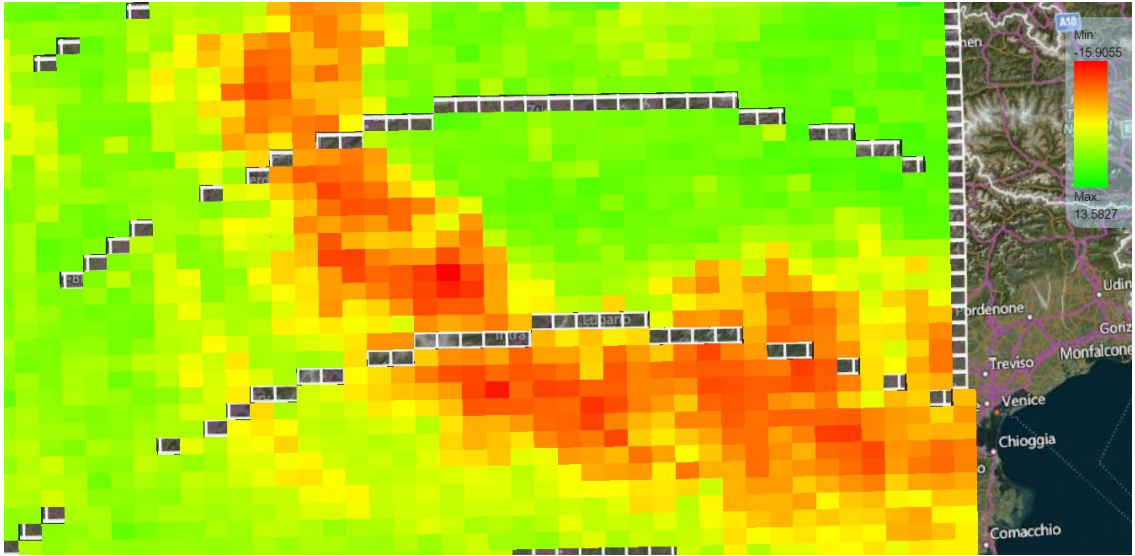


Figure 62 - Average Land Temperature, North Italy

Thanks to this image we can understand that there are several customizations available when querying big dataset like the one we selected. Moreover, all the techniques we have shown before are all still valid in each use case, thus, if the dataset permits, we can navigate through time, showing different times with differencing height, comparing two variables, grouping the data in Big Voxels, filtering out other data and obtaining relevant statistics.

Below, we can see a picture representing all the data for the twelve months of 2014. In the chart below the image, we obtain the statistics for a single voxel, seeing that we reached a maximum temperature of 28° for the month of June.

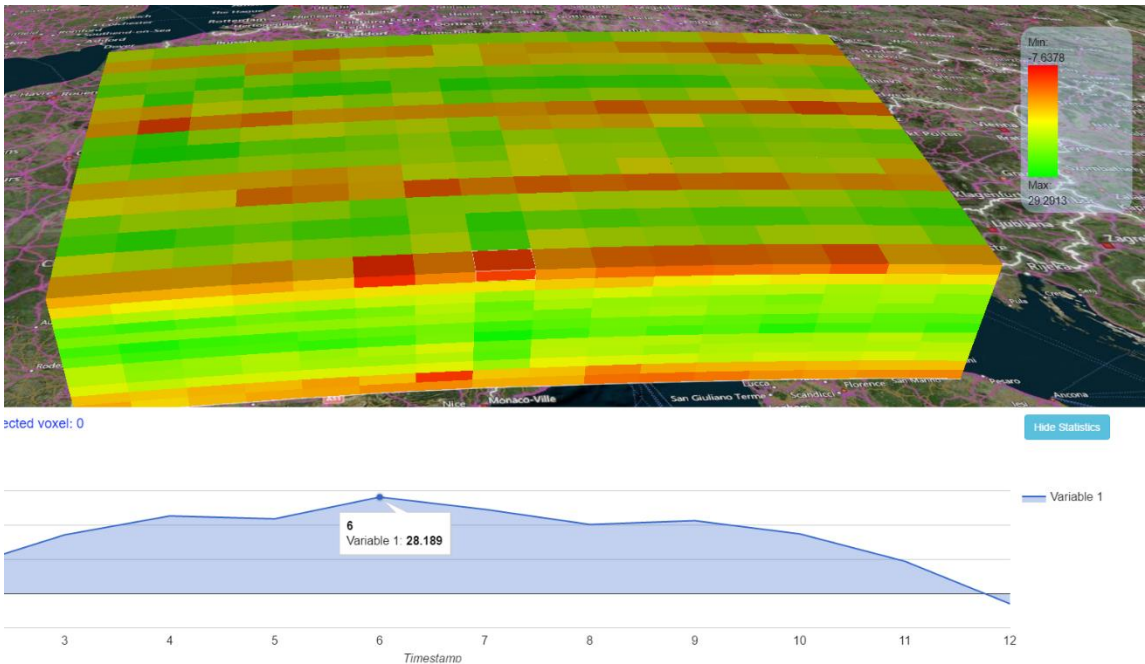


Figure 63 - Average Land Temperature, North Italy, 12 Months

Chapter 6

Conclusions

6.1 Evaluation of Experiences

The newly developed application satisfies the expectations for visualization features and performance. We experienced some performance issues when visualizing a significant amount of data, but with the simplifications we applied, we can still process large quantities of data without performance issues.

In our case study, the application shows data from sample datasets, but we can visualize any data source in this CSV format since the application is predisposed to any other source of data, or the application can also be configured to connect directly to a database in order to retrieve the data.

As shown, the application works both with a reference grid in which we can place the doxels, but also works without the use of the grid, as shown in the QuadTree implementation. This allows users to import any geo-referenced data and display a doxel model without knowing the bounding boxes of each doxel a priori. In this way, we found several datasets that could be shown in the application, and many possible study cases are available.

We implemented several features and all of them can be shown, specific to the desired use-case by simple customization of a dataset, or adjusting a few options to visualize any number of datasets.

6.2 Future Developments

To extend the application more, new ideas could be implemented. A useful feature might be to import NetCDF files into the application. However, currently there are no pure JavaScript readers for these files. Without a JavaScript reader, a fully client-based solution would need a server side to read the files and convert them to the desired format. This is the reason we implemented the Rasdaman system, so as to query the data in an effective and efficient way.

Also working more on the importing system would allow making more general the input file the application accepts, even if at the current state, the most common formats are available.

Bibliography

- [1] D. Butler, NATURE | Vol 439 | 16 February 2006.
- [2] D. S. S Sagioglu, "Big data: A review," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, 2013, pp. 42 - 47.
- [3] "How Big Data Analysis helped increase Walmart's Sales turnover?," Dezyre, 23 May 2015. [Online]. Available: <https://www.dezyre.com/article/how-big-data-analysis-helped-increase-walmart-s-sales-turnover/109>. [Accessed 19 06 2016].
- [4] M. Goodchild, "NeoGeography and the nature of geographic expertise," *Journal of Location Based Services*, vol. 3, no. 2, pp. 82-96, 2008.
- [5] J. D. S. G. W. C. H. D. A. W. M. B. T. C. A. F. R. E. G. Fay Chang, "ACM Transactions on Computer Systems," *ACM Transactions on Computer Systems*, vol. 26, no. 2, 1998.
- [6] R. A. N. M. Sunita Sarawagi, "Discovery-driven exploration of OLAP data cubes," *Lecture Notes in Computer Science*, vol. 1377.
- [7] R. A. N. M. R. S. Ching-Tien Ho, "Range queries in OLAP data cubes," in *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*.
- [8] V. Marx, "Biology: The big challenges of big data," *Nature*, no. 498, pp. 255 -260, 2013.
- [9] D. D. J. Peuquet, "Representations of Space and Time," Guilford Press, 2002, p. 119.
- [10] G. L. S. N. M. T. C. T. Q. L. Nuzzo, "Application of 3D visualization techniques in the analysis of GPR data for archaeology," 2002.
- [11] P. B. Tamara Munzner, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," in *Special issue of Computer Graphics*, 1995.
- [12] S. B. C. C. T. S. E. F. Codd, "Providing OLAP (on-line Analytical Processing) to User-analysts: An IT Mandate," *Codd and Date*, vol. 32, 1993.
- [13] S. M. Y. Z. V. C. M. L. Min Chen, *Big Data Analysis*, Springer, 2014, pp. 51 -58.
- [14] "HTML5," [Online]. Available: <https://en.wikipedia.org/wiki/HTML5>. [Accessed 20 05 2016].
- [15] "Java Applet," [Online]. Available: <https://docs.oracle.com/javase/tutorial/deployment/applet/>. [Accessed 20 05 2016].
- [16] C. G. Healey, "Effective Visualization of Large Multidimensional Datasets," 1996.
- [17] M. A. Brovelli and G. Zamboni, "Environmental Space and Time Web Analyzer," in *Proceedings of the 2014 conference on Big Data from Space* , 2014.
- [18] T. O'Reilly, "What Is Web 2.0," 30 09 2005. [Online]. Available: <http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html>. [Accessed 15 05 2016].
- [19] M. D. M. J. Kremer JR1, "Computer visualization of three-dimensional image data using IMOD," *PubMed*, pp. Jan-Feb;116(1):71-6., 1996.
- [20] S. F. Gibson, "Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects," *Visualization in Scientific Computing' 95*, 1995.

- [21] M. A. Brovelli and G. Zamboni, "Virtual globes for 4D environmental analysis," in *Applied Geomatics*, 2012, pp. 163-172.
- [22] "Voxel," [Online]. Available: <https://it.wikipedia.org/wiki/Voxel>. [Accessed 15 05 2016].
- [23] S. Connolly, "7 KEY DRIVERS FOR THE BIG DATA MARKET," [Online]. Available: <http://hortonworks.com/blog/7-key-drivers-for-the-big-data-market/>. [Accessed 16 06 2016].
- [24] D. Laney, "3D Data Management: Controlling Data Volume, Velocity, and Variety.," META Group, 2001.
- [25] D. Pritchett, "Base: An Acid Alternative," vol. 6, no. 3, pp. 48 -55, 2008.
- [26] "Databases: relational vs object vs graph vs document," [Online]. Available: http://www.cbsolution.net/techniques/ontarget/databases_relational_vs_object_vs. [Accessed 21 06 2016].
- [27] "Online analytical processing," [Online]. Available: https://en.wikipedia.org/wiki/Online_analytical_processing. [Accessed 21 06 2016].
- [28] S. K. M. K. N. Di Sourav S. Bhowmick, *Web Data Management: A Warehouse Approach*, Springer Science & Business Media, 2006.
- [29] "Converting Snowplow data into a format suitable for OLAP reporting tools e.g. Tableau, Qlikview, Pentaho, Microstrategy," [Online]. Available: <http://snowplowanalytics.com/guides/tools/olap/>. [Accessed 21 06 2016].
- [30] "APACHE KYLIN™ OVERVIEW," [Online]. Available: <http://kylin.apache.org>. [Accessed 21 06 2016].
- [31] A. D. P. F. R. N. W. P. Baumann, "The Multidimensional Database System RasDaMan," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998.
- [32] "Rasdaman," [Online]. Available: <http://www.rasdaman.com/product.php>. [Accessed 21 06 2016].
- [33] "Web World Wind," [Online]. Available: <https://webworldwind.org>. [Accessed 20 05 2016].
- [34] "Node.js," [Online]. Available: <https://nodejs.org/en/>. [Accessed 5 20 2016].
- [35] "What Makes Node.js Faster Than Java?," [Online]. Available: <https://strongloop.com/strongblog/node-js-is-faster-than-java/>. [Accessed 20 05 2016].
- [36] "Express," [Online]. Available: <http://expressjs.com>. [Accessed 20 05 2016].
- [37] "Representational state transfer," [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 20 05 2016].
- [38] "CamelCase," [Online]. Available: <https://en.wikipedia.org/wiki/CamelCase>. [Accessed 20 05 2016].
- [39] D. E. L. G. H. L. K. S. J. H. U. An, "Applicability Evaluation of Flood Inundation Analysis using Quadtree Grid-based Model," *Journal of the Korean Water Resources Association*, vol. 46 N°6, pp. 655 - 666, 2013.
- [40] "Voronoi Diagram," [Online]. Available: <http://mathworld.wolfram.com/VoronoiDiagram.html>. [Accessed 20 05 2016].
- [41] J. L. B. R. A. Finkel, "Quad Trees: A Data Structure for Retrieval on Composite Keys.," *Acta Informatica*, vol. 4, no. 1, pp. 1-9, 1974.

- [42] "Jasmine," [Online]. Available: <http://jasmine.github.io/2.4/introduction.html>. [Accessed 20 05 2016].
- [43] "RequireJS," [Online]. Available: <http://requirejs.org>. [Accessed 20 05 2016].
- [44] E. E. Koutsofios, F. P. N. U. AT&T Bell Labs., S. C. North and D. A. Keim, "Visualizing large telecommunication data sets," *EEE Computer Graphics and Applications*, vol. 19, no. 3.
- [45] N. G. M. |. L. Resolutions. [Online]. Available: <http://www.bgs.ac.uk/services/3dgeology/lithoframe.html>. [Accessed 23 06 2016].
- [46] "West Thurrock model information," [Online]. Available: <http://www.bgs.ac.uk/services/3Dgeology/modelInfo/thurrock.html>. [Accessed 23 06 2015].
- [47] "Skewness To Systematic review (Statistics)," [Online]. Available: <http://what-when-how.com/statistics/skewness-to-systematic-review-statistics>. [Accessed 11 05 2016].
- [48] M. Crawley. [Online]. Available: <https://fas-web.sunderland.ac.uk/~cs0her/Statistics/UsingLatticeGraphicsInR.htm>. [Accessed 15 05 2016].
- [49] M. O. WardIn, "Multivariate data glyphs: Principles and practice,," in *Handbook of data visualization*, 2008, p. 180.
- [50] J. K. H. H. Andreas E. Lie, *Proceedings of the Spring Conference on Computer Graphics*, 2009.
- [51] D. J. Fuchs, "Evaluation of Alternative Glyph Designs," [Online]. Available: <https://www.vis.uni-konstanz.de/en/members/fuchs/>. [Accessed 15 05 2016].
- [52] N. Elmqvist, P. INRIA, P. Dragicevic and J.-D. Fekete, "Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation," 2008.