Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 16, 2017

Smart–IVC

Enhanced Visualization of Cities Through Smart Visual Queries

Andrea Vicari

Abstract

Cities constantly evolve, with the appearance of new neighborhoods and the disappearance of old buildings. Information about cities are nowadays stored as static data inside huge storages where it is difficult and slow to retrieve particular information. Also different data about cities is not aggregated since they are provided by both public websites on the Internet and by public sectors of the city.

A visualization can help supporting any decision that involves city evolution, especially in the contest of what is called a Smart City. No such service exists that provides a unique environment in which the user can both visualize a city and interact with its elements. The only technologies available are either not exhaustive or too complex to use. For example, it is possible to find technologies online that are limited to provide a 3D–visualization without any kind of interaction between the map and the user.

Smart–IVC provides an interactive 3D–visualization model of cities, integrating heterogeneous data, and supporting complex visual queries. Smart–IVC is an application accessible by everyone and that aims to enhance the city visualisation, getting closer to the user needs.

Advisor
Prof. Michele Lanza
Assistant
Prof. Dr. Andrea Mocci

Advisor's approval (Prof. Michele Lanza): Date:

Contents

List of Figures

1 Introduction

Cities nowadays are in constant evolution. They change their structure over time through the appearance of new entities and the disappearance of old ones, thus they are very unsteady and prone to changes. Important changes might be considered with regard to the deployment and management of all types of infrastructures within cities. For example, the decision of which building to demolish in order to make space for the construction of a new mall in a city.

Moreover, cities have a very large impact on the economic and social development of nations: they represent the real foundation where people live, where companies have their business and in which numerous services are provided. Often, a city overview is not available, so decision may be taken without having a big picture of the surrounding environment. This could lead to choices that might have a negative impact on the current city, reducing the efficiency and the life quality of the citizens. Also, data is not very well aggregated since different information about the same city can be provided by different public and online services for example Google and OpenStreetMap or by public city entities like the cadastre office.

To control these changes and to aggregate this data, a visualization, which provides a city overview, is required. Such a visualization is considered to be the first step towards what it is called a Smart City. A Smart City can be defined as a city which uses information and communication technologies so that its critical infrastructure as well as its components and public services provided are more interactive, efficient and so that citizens can be made more aware of them.

1.1 Contributions

Smart–IVC aims to solve the problem generated by the static and non–aggregated nature of data regarding cities (as mentioned above) through an Interactive 3D–Visualization model. Therefore, the main contributions that Smart–IVC provides are:

- The visualization of a 3D city model. As a use case for this Bachelor Project, the city of Lugano has been taken into account.
- The interactivity that the user has with the entities inside the city: buildings can be clicked in order to receive information about them and in order to receive information between the selected building itself and the rest of the entities in the city.
- The possibility to have a graphical city overview using a very intuitive system of visual queries which produce results that are immediately visible through the highlighting or colouring of the buildings.

1.2 Structure of the report

- **Chapter 2** discusses the state of the art. It talks about already existing tools which allow city visualization in a virtual environment. In particular, it focuses on tools that use the Cesium framework comparing and contrasting the existing features with the ones proposed in Smart–IVC.
- **Chapter 3** is the main chapter of the entire report. It shows how Smart–IVC has been developed step by step. The decisions that have been taken and the technologies adopted will be explained and discussed in detail starting from the parsing of the unique .xml file provided by the Comune of Lugano and ending with the final result (i.e., the 3D–visualization of the entire city).
- **Chapter 4** illustrates showing some use cases of the application. Examples will be illustrated and explained in detail, in order to show the features proposed by Smart–IVC.
- **Chapter 5** is the conclusion of this report. Here, the current limitations of the application will be presented. Finally, we will discuss and analyze the future work and the different paths that Smart–IVC might take.

2 State of the Art

City visualization is a subject undergoing intense study, therefore it is easy to find new projects about it everyday, on the Internet. Still, the uniqueness of Smart–IVC resides on the fact that it provides not only a visualization of a city but, above all, it gives the user the possibility to interact with the city itself through a simple system of visual queries.

Before getting into the details with Smart–IVC, we will clarify the current state of city visualization and, since the projects about this topic are so various and so many, in this Chapter, we will present the ones that relate most with the final outcome of this Bachelor Project.

At first, we will present some 3D–city–model based on OpenStreetMap data, as **OSMBuildings**, **F4 Map** and **ViziCities**. Then, we will introduce the **Cesium framework**, since it has been used used as a basis for Smart–IVC. At the end, we will analyze the work done by the **Swiss Geospatial Portal** (still using Cesium), in order to compare it with the application described in this report.

2.1 OpenStreetMap–Based Frameworks

OpenStreetMap[1] is a project that creates and distributes free geographic data for the world. During the last decade, the third dimension has become a growing topic at OSM, so it is now possible to add detailed buildings and a lot of minor objects. Here, some of the frameworks that use this data, will be analyzed.

2.1.1 OSMBuildings

OSMBuildings[2] is an engine for displaying 3D buildings on a web map. It uses information on buildings provided by OpenStreetMap and it renders them on a map layer.
Altough in some cases, buildings are very detailed (e.g., in big cities like New York or Berlin), it does only provide a visualization without any kind of interaction.

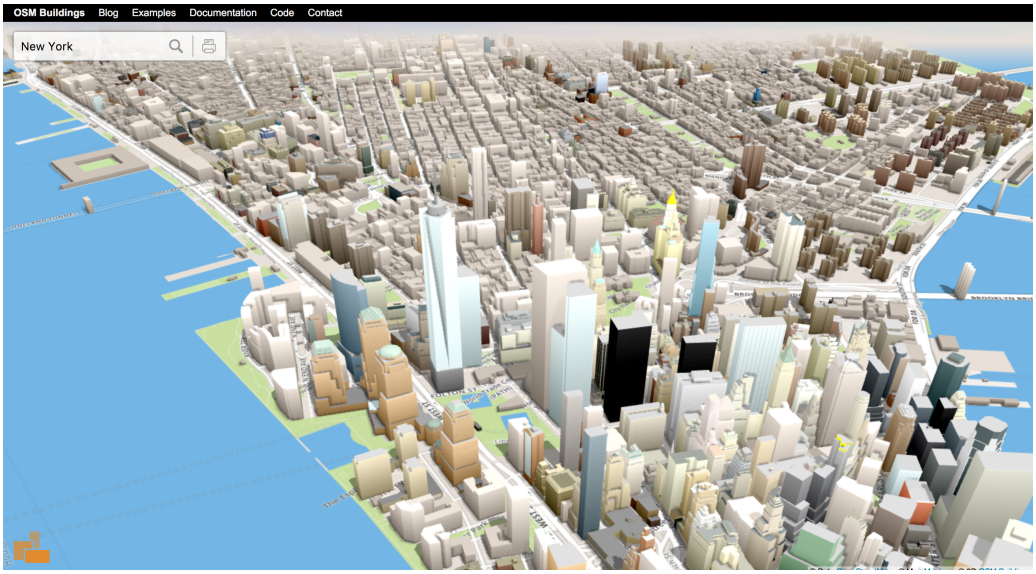


Figure 2.1. A visualization of New York city using OSMBuildings

2.1.2 F4 Map

The F4 Map[3] is an OSM-based 3D map using the WebGL technology. Also this map, uses Open Street Map's buildings but also adds some non–OSM–provided features like trees, cranes and other data.
Some 3D models are used (not based on buildings data in OpenStreetMap) for some specific buildings (e.g. the Eiffel Tower in the example below).

[1] OpenStreetMap: phttp://www.openstreetmap.org/
[2] OSMBuildings: https://osmbuildings.org/
[3] F4 Map: http://demo.f4map.com/

Figure 2.2. A visualization of Paris using F4 Map

Unfortunately, the F4 group does not provide any kind of documentation and this project is not open source.

2.1.3 ViziCities

The third and last map that uses OSM information about buildings is ViziCities[4] , a framework for 3D geospatial visualization in the browser. Its code is available on GitHub since it is an OpenSource project but they are still at version 0.3 of the application.
It lets the developer to add some useful information to the map like routes and GeoJSON (i.e., a format for encoding a variety of geographic data structures) but is still very limited and not very prone to interactivity.

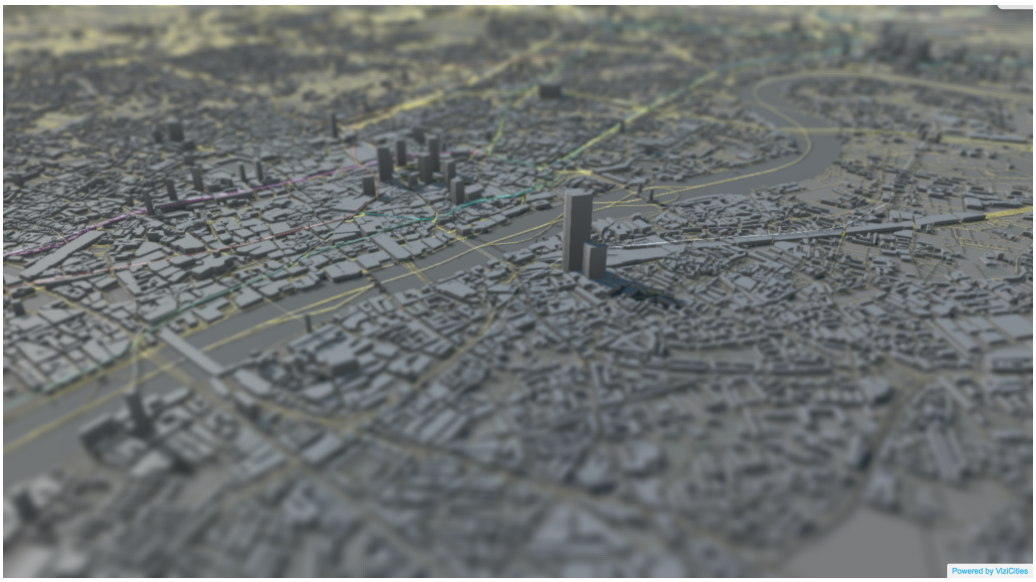


Figure 2.3. A visualization of New York city using ViziCities

[4]ViziCities: https://github.com/UDST/vizicities

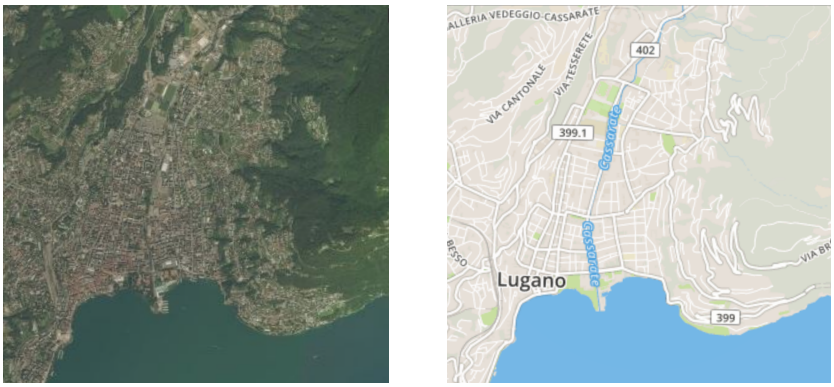
2.2 Cesium

Figure 2.4. Cesium logo and Cesium Virtual Globe

Cesium[5] is an open-source JavaScript library for world-class 3D globes and maps that is used to create a web-based globe and map for visualizing dynamic data. This framework runs using WebGL, a JavaScript API for rendering 3D graphics within any compatible web browser that allows GPU–accelerated usage of physics and image processing and effects as part of the web page canvas.
The features that Cesium provides are the following:

- **A virtual globe:** a three-dimensional software model or representation of the Earth that provides the user with the ability to freely move around in the virtual environment by changing the viewing angle and position.

- **Different imagery providers:** it allows to draw and layer on high-resolution imagery (maps) from several standard services directly on the virtual globe.



(a) High-resolution, mesh-based terrain provided by Bing

(b) Streets basic imagery provided by Mapbox

Figure 2.5. Example of two imagery providers available on Cesium

- **Different terrain providers:** it allows to visualize global high-resolution terrains and water effects for oceans, lakes, and rivers, but mainly the possibility to represent mountain peaks, valleys, and other terrain features.

[5]Cesium: https://cesiumjs.org/

(a) The standard WGS84 Ellipsoid **(b)** Terrain meshes provided by STK

Figure 2.6. Example of two terrain providers available on Cesium, this shows the benefits of a 3D globe compared to a 2D map.

- **Huge number of APIs provided:** since the uses of Cesium are really various, Cesium provides a very high number of APIs in order to control every aspect of the web application, i.e., draw every kind of geometry, handle animations, etc...
  The official website of Cesium also provides a complete documentation, various examples, and playgrounds.

2.2.1 Cesium: 3D-Tiles

Cesium's repository on GitHub[6] has an open fork called "3D-Tiles"[7]. As specified by the contributors of this fork: "In 3D Tiles, a tileset is a set of tiles organized in a spatial data structure, the tree. Each tile has a bounding volume completely enclosing its contents. The tree has spatial coherence; the content for child tiles are completely inside the parent's bounding volume. To allow flexibility, the tree can be any spatial data structure with spatial coherence, including k-d trees, quadtrees, octrees, and grids."

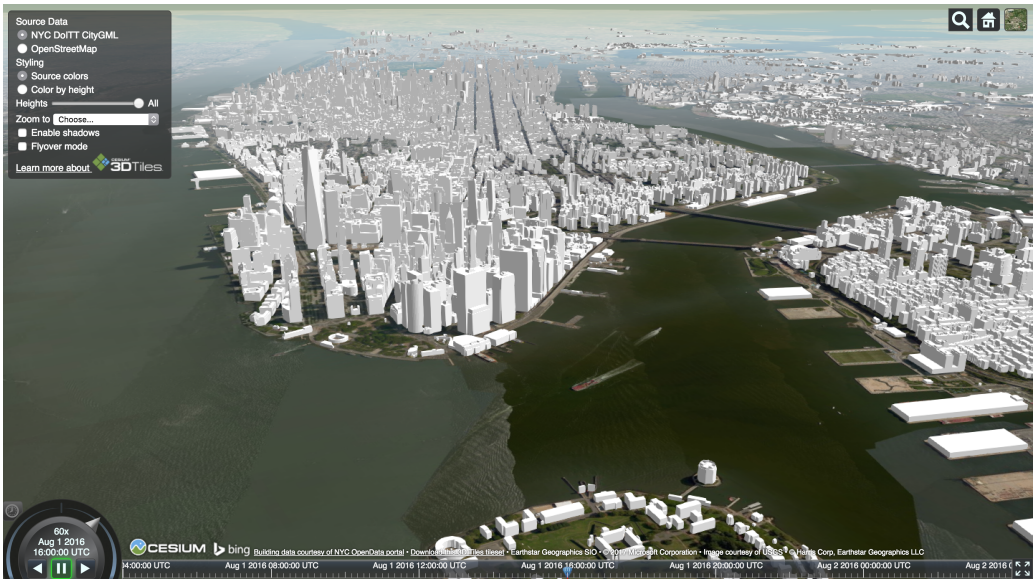


Figure 2.7. An example showing a 3D visualization of the city of New York. Using Cesium 3D Tiles Tecnhology

Unfortunately, being Cesium 3D–Tiles still under development, it is not provided any kind of documentation about it. Still, in a recent post on Cesium Forum, it is possible to understand that Cesium 3D–tiles fork will be soon merged to the main branch and therefore, also a complete documentation will be provided.
In the meantime, Cesium 3D–Tiles developers are available to help programmers interested in using this technology using their forum[8].

[6] Cesium GitHub Repository: https://github.com/AnalyticalGraphicsInc/cesium
[7] Cesium 3D-Tiles Fork: https://github.com/AnalyticalGraphicsInc/3d-tiles
[8] Developers' help forum: http://tiny.cc/884tly

2.2.2 Swiss Geospatial Portal Using 3D Tiles

Swisstopo, the Swiss Federal Geoportal is a federal government platform that facilitates public access to Swiss spatial data. This agency produces detailed maps of Switzerland and also documents geological, geodesic, and topographical changes in the landscape.
Swisstopo is one of the pioneers in implementing 3D Tiles to process its extensive data collection and Cesium to visualize it. The beta visualization of their 3D Geoportal is available, it makes their national geodata collection widely available[9].

Figure 2.8. A 3D visualization of the city of Bern in the Swiss Geospatial Portal. Using Cesium 3D Tiles Tecnhology

The work done by Swisstopo, so far, consists in placing the 3D–models of buildings in some of the cities in the norther part of Switzerland (e.g., Bern, Rapperswil-Jona, Winterthur, etc...). It is not provided any kind of interaction with the buildings.
Swisstopo provides a service in which it is possible to buy the 3D–models of the buildings, they are not highly detailed but they contain important information about the shape of the roof as can be seen in Figure2.9.

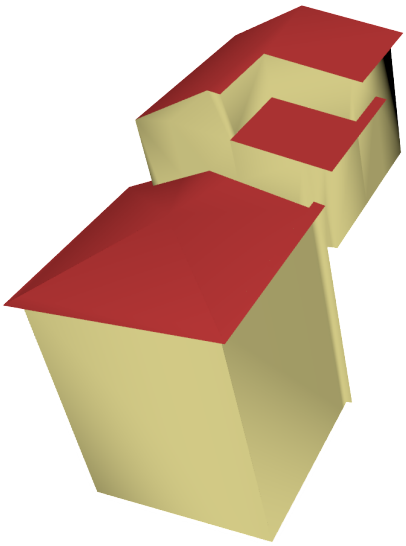


Figure 2.9. A 3D–Model of a building provided by the Swiss Geospatial Portal and used in 3D–Tiles

For this reason, during the last period of the development of Smart–IVC, models of the city of Lugano have been bought. The use of these models may be done as one of the future developments of this application. This will be discussed at the end of this report.

[9]Swisstopo implementing 3D–Tiles: http://tiny.cc/0s5tly

3 Smart–IVC

The upcoming sections will illustrate Smart–IVC features and how they were developed. The first part proposes the Smart–IVC modelling of data. Afterwards we present a section about the parser and the cron–jobs created. It follows a description of the API provided to the user. Finally this chapter concludes with some discussions about how we developed the structure and the logic behind the Web–Application.

3.1 Smart–IVC Architecture and Technologies

The environments selected to develop Smart–IVC are the following:

- **The Server–side** was written using the **Java** Programming Language. The **Spring Framework** was used, in particular using its most famous convention–over–configuration, called **Spring Boot**. The database used to store the information about the city is **MySQL**.
- **The Client–side** was written using the **Javascript** scripting language. The **jQuery** library was used and, of course, **HTML5** and **CSS3**. As mentioned above, the 3D visualization of the city was achieved using the **Cesium Framework**

Figure 3.1 shows the structure of the final Smart–IVC 's prototype.

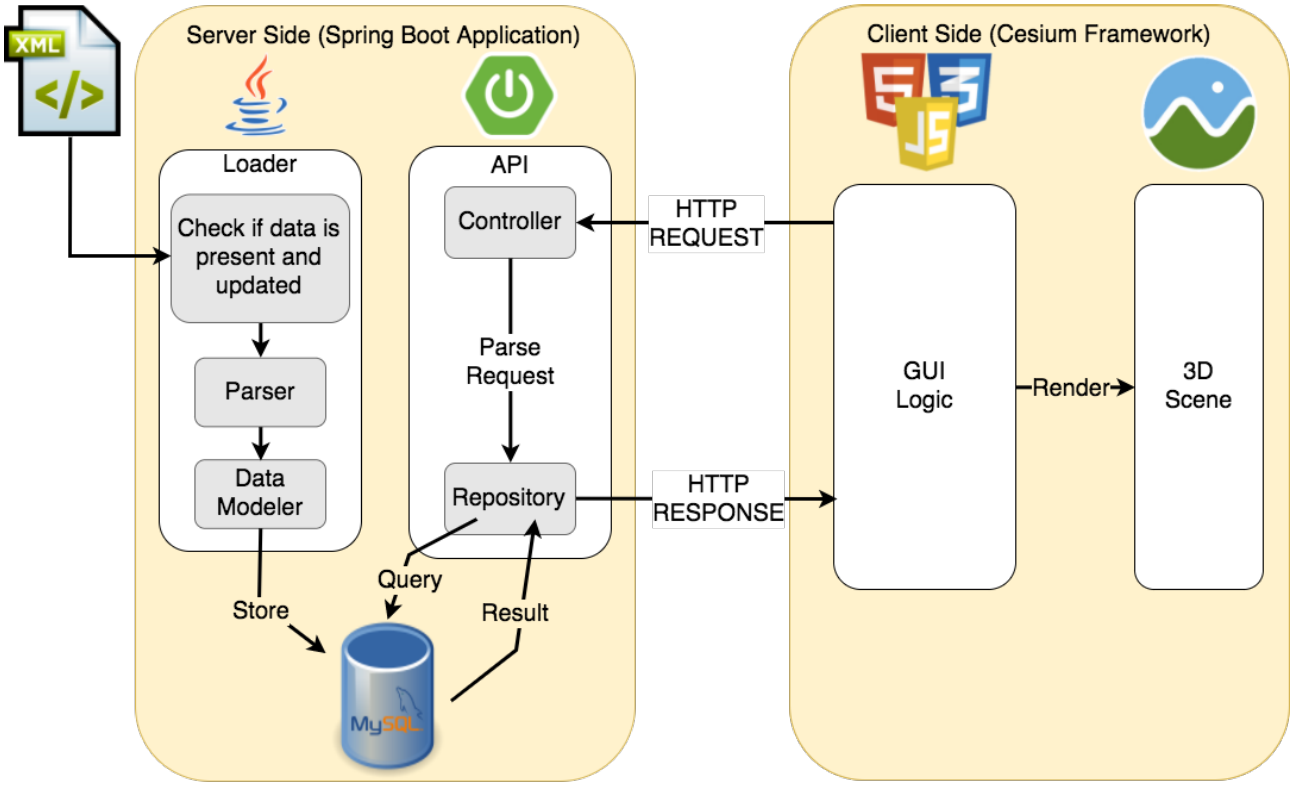


Figure 3.1. Smart–IVC architecture scheme

3.2 Server Side

3.2.1 Available Data and Modelling

The entire work starts from the "xml" file provided by the Comune of Lugano. The first choice to take was how to model the available data in the best way such that the way of retrieving information about any building was as fast and as consistent as possible.
The models created are the following: City, Suburb, Building, Address and Type. Figure 3.2 shows, in the form of an ER diagram, the final version of how the data was modelled into the database, specifying columns and relations between tables.

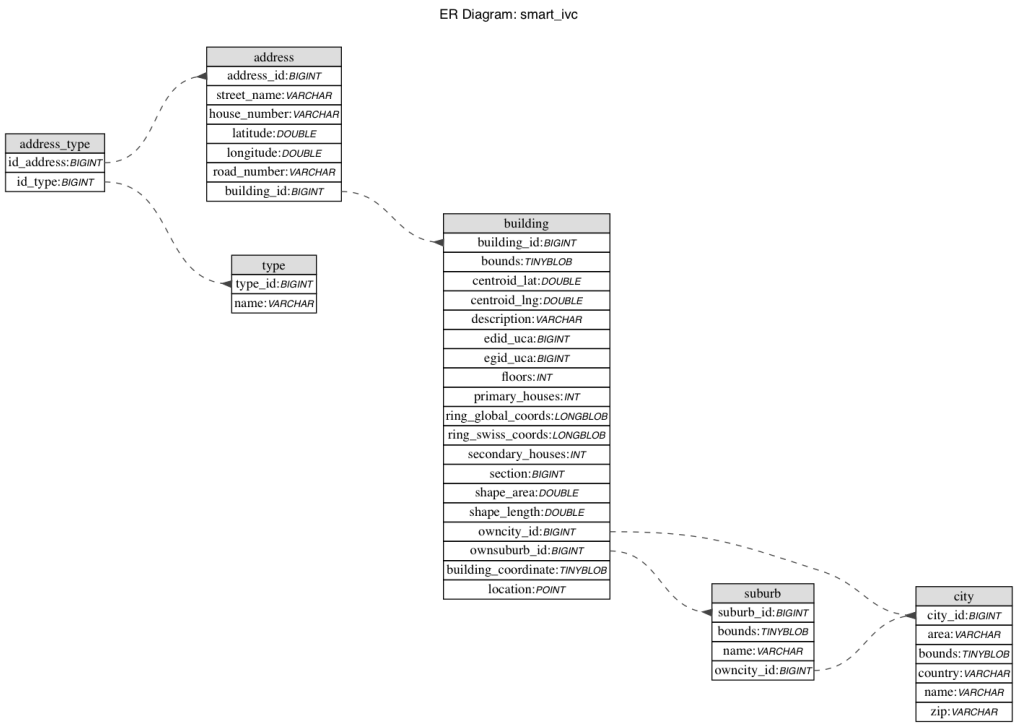


Figure 3.2. Smart–IVC database structure

3.2.2 Parser

The data type taken as input is of xml type. It is used in the process to load buildings data.
The beginning of the file contains an header which describes the schema of the content that follows. It regulates fields and types as well as values range, tolerances in the coordinates system and other set ups parameters. After the header, a long record of elements which represent buildings follows (exactly 18904). Structure and constraints, of record elements are defined in the header. The record presents several data, out of this data Smart–IVC finds use on almost all of them. Here there will be described the tags that can be found in the xml file and that were found useful for Smart–IVC

- **SHAPE:** it stores both the perimeter of the building and the max bounds that it occupies
- **Descrizione (Description):** it represents the type of the building. Between them only the ones of type “Edificio” (i.e., building in Italian), are stored
- **Sezione (Section):** it is a numeric value that represent the suburb in which the building is located
- **NUM_CIVICO_ID (CIVIC_NUMBER_ID):** it is a 8–character–long numeric value that contains: the number id representing the street name in the first four characters and the civic number in the remaining four.
- **EGID_UCA:** it uniquely identifies buildings on the entire Swiss ground on the basis of the “Registro Federale degli Edifici e delle abitazioni” (REA) from the “Ufficio federale di statistica”.
- **PIANI (FLOORS):** the number of floors per building
- **SHAPE_AREA:** the area of the plane described by the building
- **SHAPE_LENGTH:** the length of the perimeter of the building

The data is assimilated in Smart–IVC through the loader. A parser has been created in order to read and parse a file structured in the way described above. The parser creates a building model and an address model for every record in the xml file and sets their fields using the information stored in the xml tags. Once the data is created, the models gets stored into a database.

Coordinates of buildings are stored in a data type called CH1903. It represents the Swiss projection coordinates system. It uses an Oblique Mercator on a 1841 Bessel ellipsoid. An Oblique Mercator is an oblique conformal cylinder projection. Together with the 1841 Bessel ellipsoid, a reference ellipsoid of geodesy with base at the old observatory

in Bern, gives meaning to the projected coordinates. The transformation computations of the data from CH1903 to WGS84 is provided from the Swiss confederation. The precision with this transformation is of respectively 1 meter and 0.1". In order to derive the transformation, specific formulae has to be applied. The way these coordinates are converted can be found on the website of the Federal Office of Topography Swisstopo under the section NAVREF.

3.2.3 Commands

On the application start, the application checks if the data is already present in the database, if not it executes a command that reads the xml file and parses it using the parser described above and stores the models in the database. At this point, the data is not ready yet, since it does not contain all the important information required from Smart–IVC to work.
That is why some commands where create for different purposes. Again, on application start, it is checked if the data is updated properly, if not the following commands are executed:

- The **CityLoaderCommand** contains two commands to be executed: the first one, as explained above, parses the xml and stores the models in the database. The second one, adds some additional information about the buildings (i.e., number of apartments–per–building that are primary and secondary houses). This additional information was received by the Comune of Lugano in a second time in the format of a txt file. Therefore, the matching between buildings listed in the two files was done using the EGID value.
- The **ConverterCommand** is used to convert the coordinate system used for the perimeters of the buildings from CH1903 to WGS84. The conversion is done using the service provided by the Web APIs of the Federal Office of Topography Swisstopo where, given a pair of coordinates in the CH1903 system, their respective conversion in WGS84 is returned.
- The **CityInformationCommand** is used to add additional information to each building though three different services. The first one adds information about the membership suburb using an APIs service provided by Swisstopo that uses the EGID in order to get such an information. Another service provided by Swisstopo API's is used to get information about the address name and the civic number of the building, still using the EGID value. The last command uses the API service provided by OpenStreetMaps in order to get information about addresses, civic numbers (for the building that have not an EGID value stored) and type of the building (e.g., Hospital, School, University etc . . .).

Once these commands are executed, the data is ready to be used and retrieved in order to be shown on the web application.

3.2.4 Controllers and Queries

Once the data was retrieved from the xml, modelled, improved with additional details and correctly stored in the database, the only part missing on the server side was the creation of the APIs and the correlated work of querying the database to get the correct information.

We created controllers in order to make the communication between the server and the client easy and fast. Here will be presented some of the most important API created in the controllers:

The cityController:
It can be found under the section "/city". Here, we implemented the following APIs.

- "/{id}/": get information about a city using the city id.
- "/{name}/": get information about a city using the city name.
- "/allCityNames/": get a list of all the names of the cities stored in the database

These services may seem useless since in this report we always visualize the city of Lugano. Still, Smart–IVC has been created to support any city visualization. That occurs because, one of our future work idea, is consider the fact to add more cities to our database. In such a case, the APIs presented above will be more useful.

The suburbController:
It can be found under the section "/suburb". Here, we implemented the following APIs.

- "/{id}/": get information about a suburb using the suburb id.

- "/{name}/": get information about a suburb using the suburb.
- "/fromCityId={id}/": get the list of all suburb models belonging to the city that matches the id in the input

The typeController:
It can be found under the section "/type". Here, we implemented the following APIs.

- "/{id}/": get the name of a type using the type id.
- "/{name}/": get the id of a type using the type name.
- "/getall/": get the list of all the types id with their respective names.

The buildingController:
It can be found under the section "/building". This controller is the most developed since Smart–IVC bases its visual query system on the interactions between the user and the buildings shown in the city. Here, we implemented the following APIs.

- "/city={id}/": get the list of buildings models belonging to the city that matches the id in the input. It is the request used to render the entire city on start of the application
- "/max={maxLat},{maxLng}&min={minLat},{minLng}/": get the list of buildings models belonging to the bounding box defined by the two coordinates–points in the input. It was mainly used on our first approach on Babylon.js in order to render small portions of the city without drastically decreasing performances
- "/info/{id}/": get information about the building that matches the id in the input. The information received are the following
  - id value
  - Number of floors
  - Civic Numbers
  - Street Names
  - Types
  - Suburb
  - Primary and Secondary Houses percentage
  - EGID value
  - Shape Length
  - Shape Area
  - Coordinates of the Ring of the building (used to draw the model of the house in the Info–Box)
- "/query/{queryBody}/": get the id of the buildings that will be selected on the 3D–visualization, after the execution of the queryBody.
  Since we wanted to avoid creating a different query repository for any combination of query, we created a query–builder.
  The query builder works as follows: it parses the string "queryBody" in the request, it then builds a unique query and executes it. The use of a unique query was inevitable since we wanted to increase the speed of the response as much as possible.
- "/distanceQuery/{queryBody}/": get the a list of tuples as result containing: the id of the building and a value in the range $[0, 1]$ that represents the distance from the building specified in the request input. The largest the number, the farthest is the building is from the source building.
- "/distanceMap/byTypeId={typeId}/": this API is very similar to the previous one. It is based on the fact that there could be more than one building of the same type in the same city. Therefore a unique response is created aggregating the distance of the buildings from the various hotspots.

In the case of the last two APIs described, the value in the range $[0, 1]$ is used as a multiplier to get the correct shade from green to white in which the building will be coloured.

3.3 Client Side

3.3.1 The first Attempt: Babylon.js

Immediately after having gathered the essential data useful to draw the buildings of the city of Lugano, a first attempt of visualizing them was made using BabylonJS. It is a JavaScript framework for building 3D environments with HTML5 and WebGL. It allows the creation of a scene with customized lights, cameras, materials, meshes, animations, audio and actions. It also supports scene picking (i.e., an element on the scene is clickable and it is possible to interact with it).

In order to test the capabilities of BabylonJS, a small portion of Lugano (i.e., the part around the lake) was selected and rendered. The result can be seen in the following image:

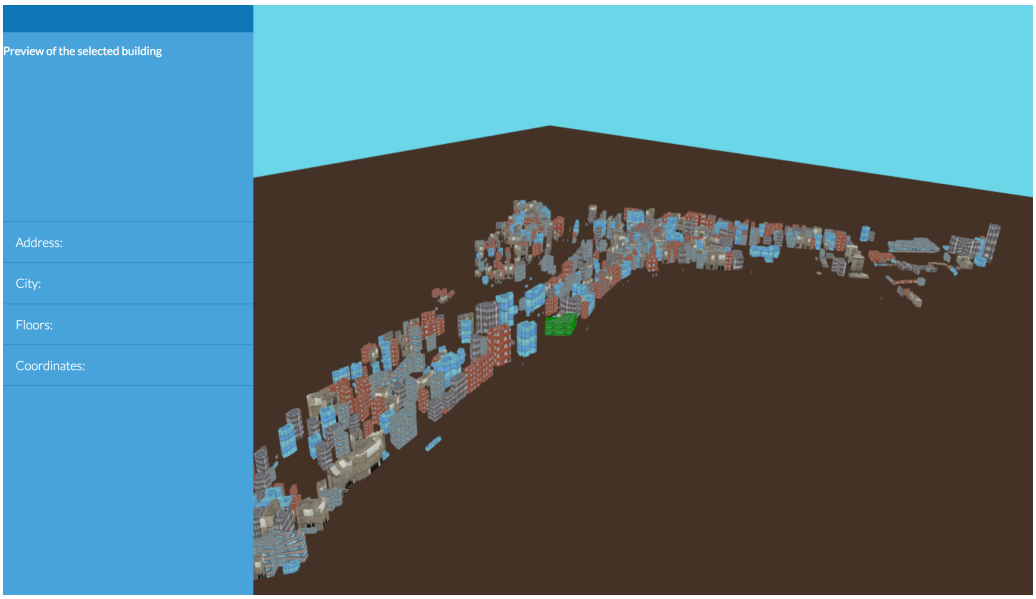


Figure 3.3. The first attempt of city visualization using BabilonJS

As long as the buildings showed in the scene where under ~ 2000, the browser was very fast in rendering during the loading of the page and the movement of the camera was smooth.
The main problems that make the idea to use BabylonJS discarded, were has been basically two:

- The buildings to be rendered were far above the mentioned threshold, the rendering would take more than 30 seconds.

- BabylonJS provides no basis to start from: the initial scene is completely empty and, as it is possible to see in the image above, buildings lay on a plane. That represented a serious problem since the visualization of the city was planned to be shown in a way that would be as realistic as possible.

A solution to the last problem would have been using the Google Elevation API. This service, given a pair of coordinates (i.e., latitude and longitude), returns the exact altitude of that point.

Unfortunately, the API system of Google provides just 2,500 free requests per day. In the case of Lugano, that extends its territory for more that $25km^2$, considering one request for meter it would have been taken more than 10 days to get the entire terrain structure (for the entire city of Rome, that spans almost $46km^2$, the days taken would have been more than 20).

Nonetheless, this would have made the rendering slower since, in addition to the visualization of the buildings, also a rendering of the terrain (i.e., lakes, mountains and rivers) would have taken place.
Therefore, this lack of both Google–API–requests and good performances, lead the idea to use BabylonJS to be discarded. Later, during the development of Smart–IVC, Babilon.js was reintroduced just to render the building model in the Info–Box about the selected building.

3.3.2 The final version: Cesium.js

As stated above, the Cesium Framework was finally used to build the client–side of the application.
Cesium, on the contrary, provides a ready–to–use virtual globe in which it is possible to directly create shapes and polygons using coordinates in WGS as points. As stated before, Cesium is provided with a full documentation about its APIs in which it is possible to control and modify every aspect of this framework: from the provided menu to the drawing of complex polygons of the surface.

In addition to this, on Cesium it is also possible to select a particular terrain provider (in the case of Smart–IVC, we used the STK), which allows to visualize global high–resolution terrains and represent mountains, valleys and other terrain features. In addition, an API on Cesium is provided in which, given a terrain provider and a point on the map, it is possible to know the elevation of that point without a maximum limit of requests. That solved the problem presented above that was the limited number of requests provided by Google in order to get the elevation of a coordinate.

In order to render the buildings on the map, the polygonGeometry API has been used: among all the attributes that this object takes (i.e., type of shadow, color, definition, etc...), it also takes an array of coordinates. These are the coordinates of the point useful to draw the geometry. Therefore, in order to draw the entire city of Lugano on the startup of the application, it makes a request to the server in order to get the coordinates (in WGS) of the buildings in the desired city, then they are rendered one at a time on the map. This operation done in Cesium is quite performing since it takes $\sim$ 5 seconds to render the entire city of Lugano and then, the operation of rotating and moving the camera is very smooth (on the contrary of what was achieved using Babylon.js).

All the issues encountered with the use of Babylon.js were solved by using the Cesium Framework. That is why we decided that Smart–IVC has to rest on this techonlogy.

3.3.3 Provided Interface

Smart–IVC aims to create interactive 3D cities representations accessible by everyone. In order to have interactions that may suggest interest in users, it is required to have an user–friendly system which allows interactions with the entities in the city. In order to achieve this feature in the most flexible way, a tab based interface is at disposal of the user. The proposed interface is composed as follows:

- An **Info–Box** is shown whenever a building is selected. All the available information about it are displayed such as: street name, civic number, membership suburb, number of floors, purpose of the building, EGID value, perimeter length, area of the building and Percentace of Primary and Secondary Houses. On top of the Info–Box is also possible to interact with a stand–alone model of the building rendered using Babylon.js.

- A **sidebar** is at user's disposal and it is divided into the following subsections:

  - The **Visualize** Tab let the user: **add elements** to the city like the geolocalization and the position of the cameras distributed around the city, **color** the city by height or by suburb and decide to **show some suburbs** rather than others.

  - The **Query City** Tab is the core of the interaction between the user and the city visualization. It is possible to create a completely personalized query, selecting which field to search and which value should be searched. This part will be better explained later in this report.

  - The **Query History** Tab contains the list of all the queries executed during the session, showing the number of results for every different query.

  - The **Credits** Tab just contains information about who worked on this project and what technologies were used for the 3D–visualization.

3.3.4 Query Builder

The client side, takes advantage of the great flexibility given by the APIs created on the server–side and explained above. They are very useful in order to visualize on the city, results of the query executed visually on the sidebar.
Once the user has clicked on the **Query City** tab, the interface shown will be the one on Figure 3.4.

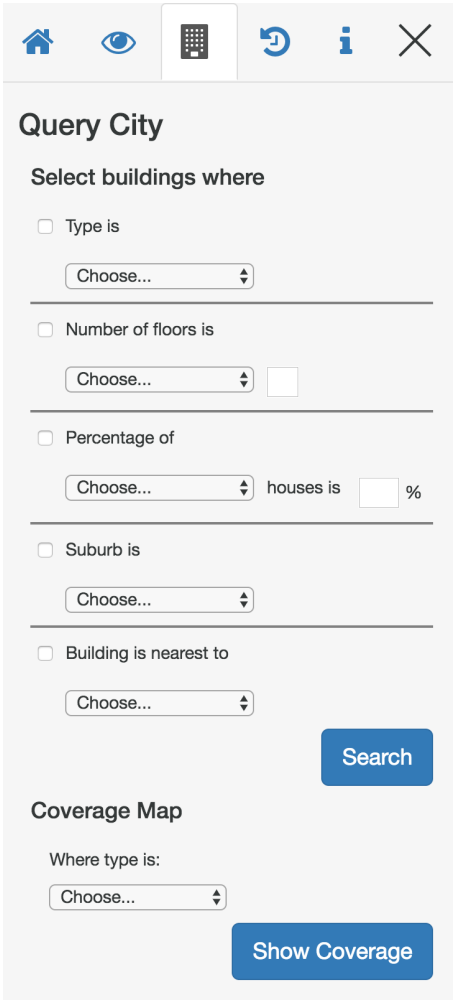


Figure 3.4. The Query City Tab

On the **Select Buildings** subsection, it is possible to select, row by row, which constraint to add to the final query. The user would only have to check the checkbox and set a value for that specific field. Of course, it is also possible to select multiple rows in order to do more restrictive queries. The buildings that match the executed query, will be highlighted with a different color.

On the **Coverage Map** subsection, it is instead possible to visualize on the map the coverage of certain buildings considered hotspots of the city. The result of this kind of query will occur colouring the hotspot (or hotspots if many of the same type) in red, the neighbour buildings will be coloured with an interpolation of colours from lime green to with depending on their distance from the hotspot.

Both kind of queries, will be better explained and enhanced with visual examples in the chapter called “Use cases”, later in this report.

4 Use Cases

In this chapter some of the possible actions that can be performed into Smart–IVC are presented. The aim of this chapter is to provide a practical introduction to Smart–IVC, through different use cases.

4.1 Access information

The first use case presented introduces the available option to access building models data.
An InfoBox is shown, whenever a building is selected through a click of the mouse or a tap on the screen of a smartphone. Figure 4.2 and Figure 4.1 shows how these information are given to the user.
The first field displayed is a 3D representation of the model of the building clicked without its surrounding environ-

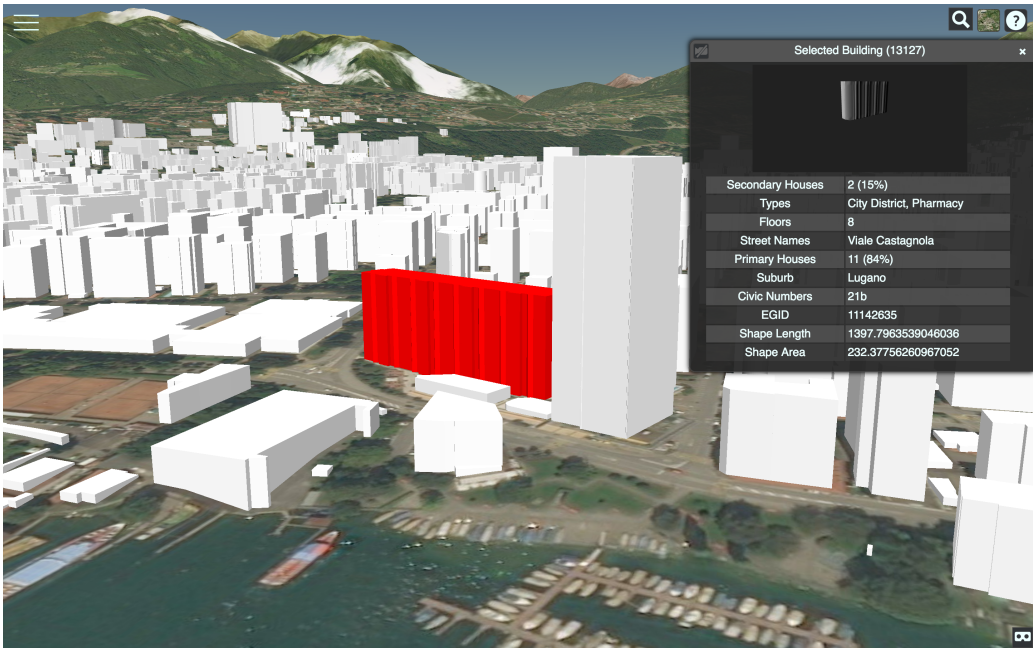


Figure 4.1. Selecting a building will make the InfoBox appear automatically

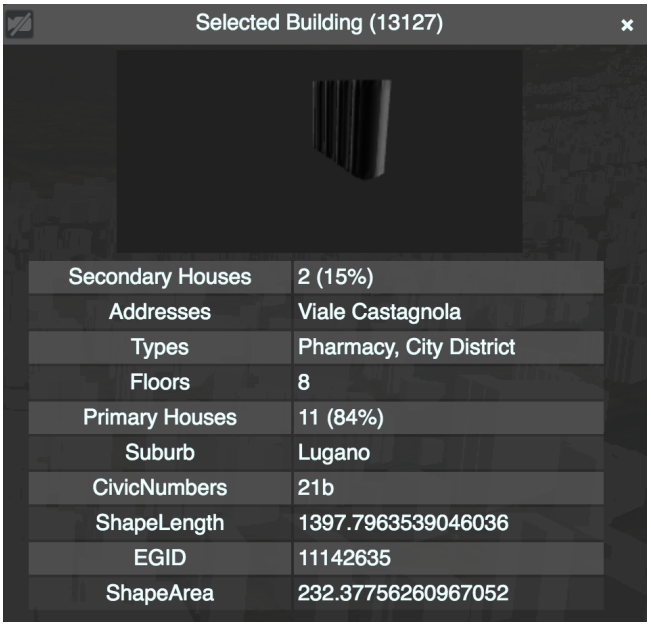


Figure 4.2. A detailed look at the InfoBox

ment. It is possible to interact with this canvas in order to rotate and zoom the building model freely.
After, a list of useful information about the building follows: the length of this list changes from building to building depending on the availability of the information.

The building clicked in the example of figure **??** presents all the possible available information that an user can get about a building.
The fields of the displayed data are denoted as follows:

- **Types** provides the list of groups in which the building is present. Groups are defined by the OpenStreetMap APIs and are used to represent the usage of the buildings which, conceptually, have characteristics in common. Examples of types are: Post Office, Pharmacy, Hospital, University etc ....
- **Floors** is the number of floors that the building possesses
- **Shape Length & Shape Area** represent the size of buildings available for consultancy, they are measured in length and area both expressed in meters.
- **Street Name & Civic Numbers** are the data concerning the addresses of the street where the building is located. There could be more than one street name and civic numbers in case the building has many of them (e.g., different entrances in the same building that are on different streets). Addresses data are taken from Swisstopo APIs and, if not available from this service, OpenStreetMap APIs are used.
- **EGID** the unique identifier of the building given by the Swiss REA
- **Suburb** represents the suburb in which the building is located
- **Primary & Secondary houses** represents both the number of primary and secondary houses in the selected building and the percentage that each value represents over the total number of houses in that building

4.2 Visualize the City

The first tab available to the user when the side–bar is opened, is the Visualize Tab. It is divided into three main subsections: “Show on the map”, “Color city” and “Show Suburbs”.

The first subsection contains two checkboxes: the first one, let the user show the geolocalization, i.e., the position of the device from which the user is using Smart–IVC. The second checkbox, if checked, shows the position of webcams located around Lugano; this webcams continuously stream videos of the city. In Figure 4.3 it is possible to see the result on the map, when both checkboxes are checked.

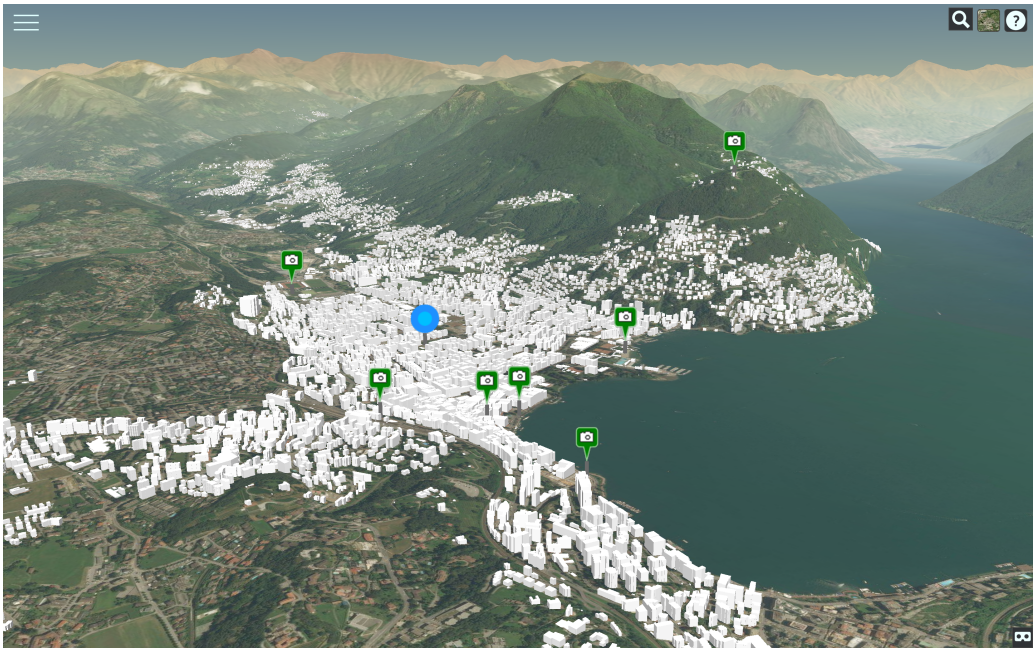


Figure 4.3. When both checkboxes are checked, geolocalization and webCams positions are shown on the map

Since the 3D visualization had still to allow the user to exactly locate the position of these pins, whenever the visualization is more zoomed it it, a “stick” is created starting from the ground and ending at the bottom of the pin, as it is shown in Figure 4.4.

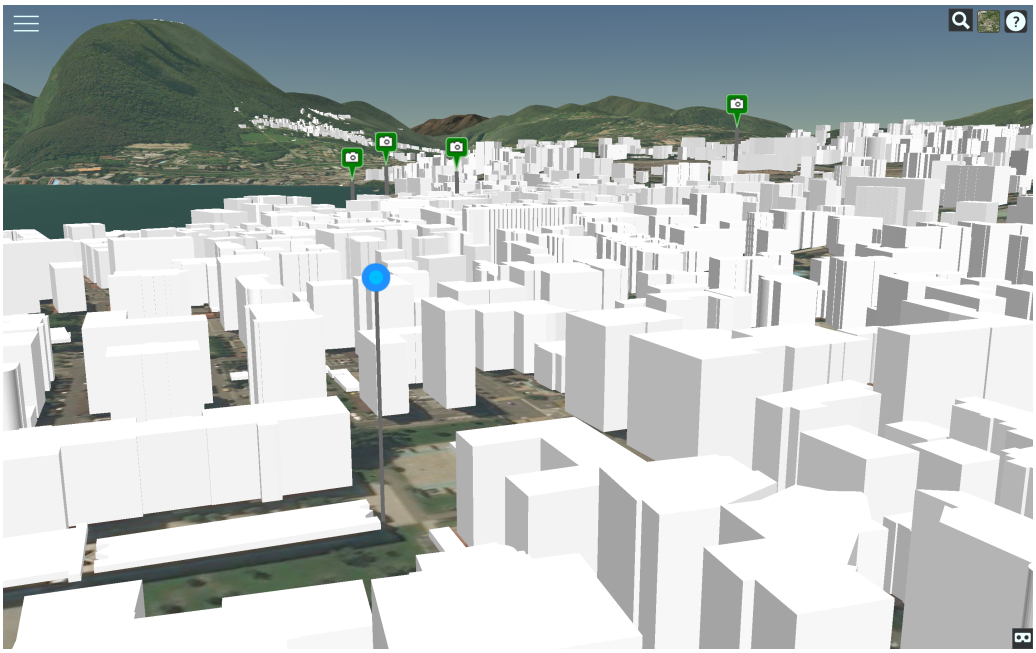


Figure 4.4. The “Stick” that connects the ground to the pins

In the second subsection three radio buttons can be found: the first one is used to reset the color of the entire city to the default color. The second colours the buildings in the city based on their height (Figure4.6). The third one is used to colour the entire city based on its suburbs and so every suburb has a different color(Figure 4.7).

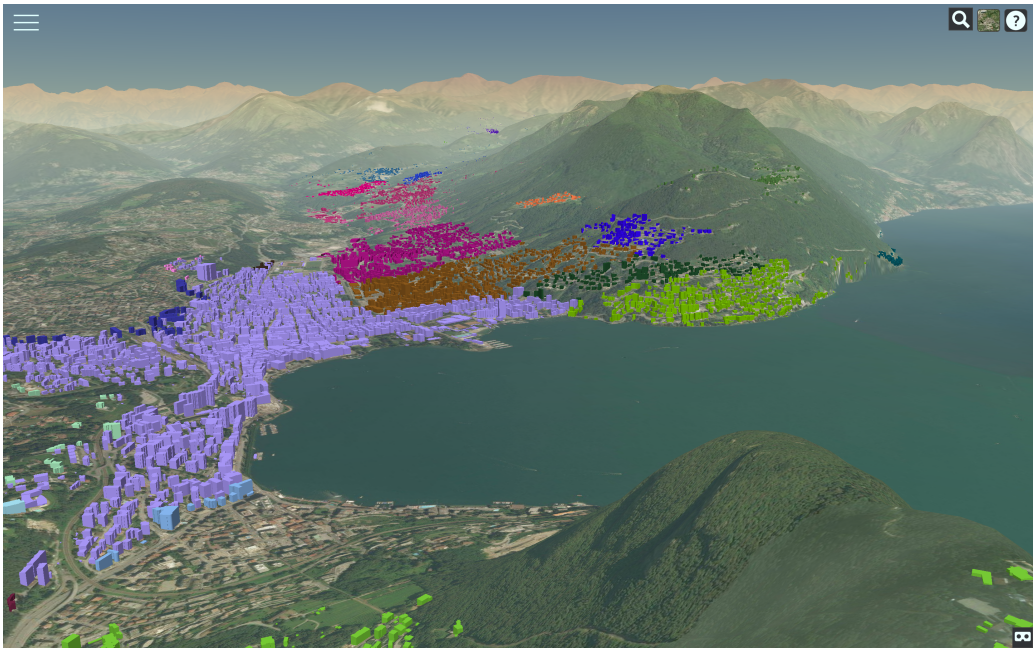


Figure 4.5. A Visualization of the city of Lugano where every suburb is coloured differentrly

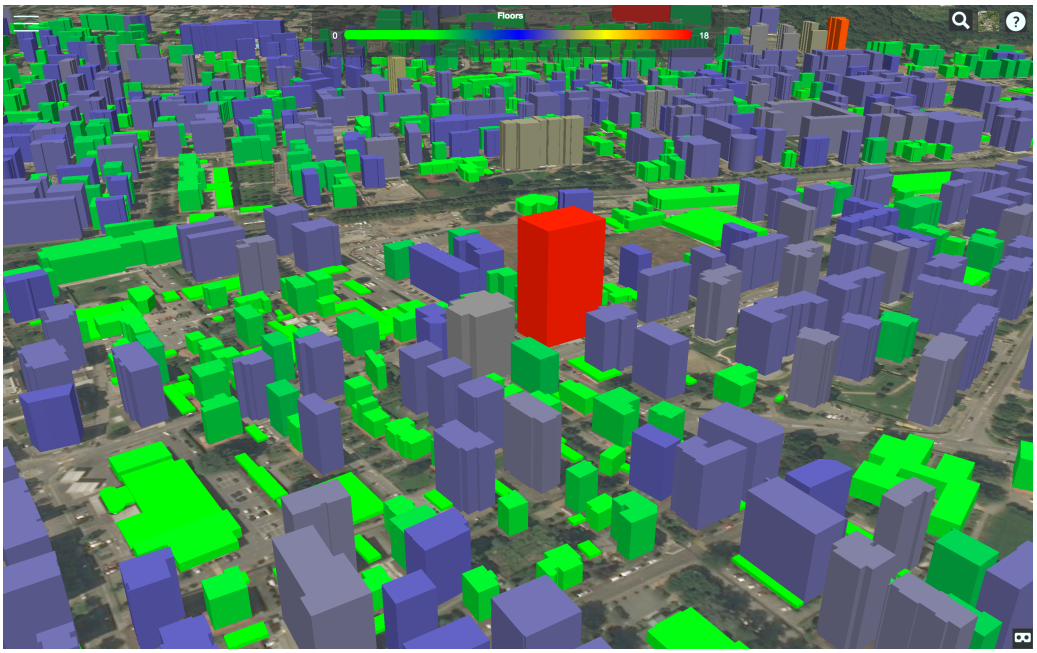


Figure 4.6. A Visualization of the city of Lugano where every building is colorued based on its height

The third and last subsection, contains the list of checkboxes representing all the suburbs in the city and their respective colours. By default all checkboxes are checked but uncheck them will result in hiding the desired suburb as show in Figure 4.7.

Figure 4.7. A Visualization of the city of Lugano where the suburb Viganello is hidden

4.3 The Query System

4.3.1 Building Selection

The following use cases show what actions can be performed using the visual query system. The interactions proposed are accessible through the apposite “Query City” selection tab in the provided sidebar.
As it is possible to see on Figure 3.4 in the previous chapter, Smart–IVC allows the user to perform from very single to more complex query on the entire city. Some examples are shown in the following images:

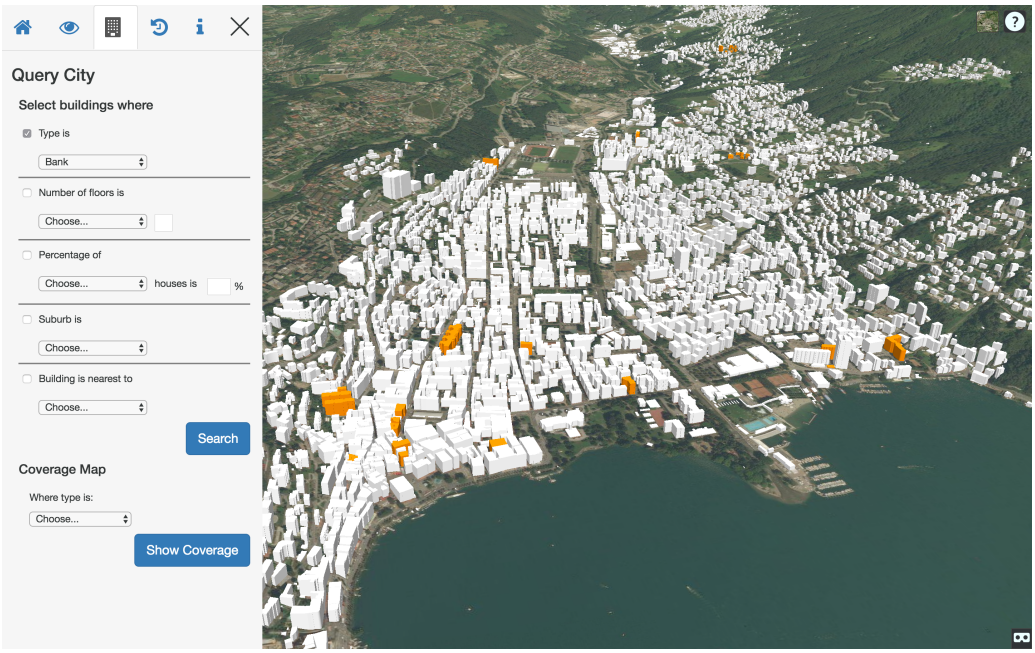


Figure 4.8. Result of the query: "Get all the banks in Lugano"

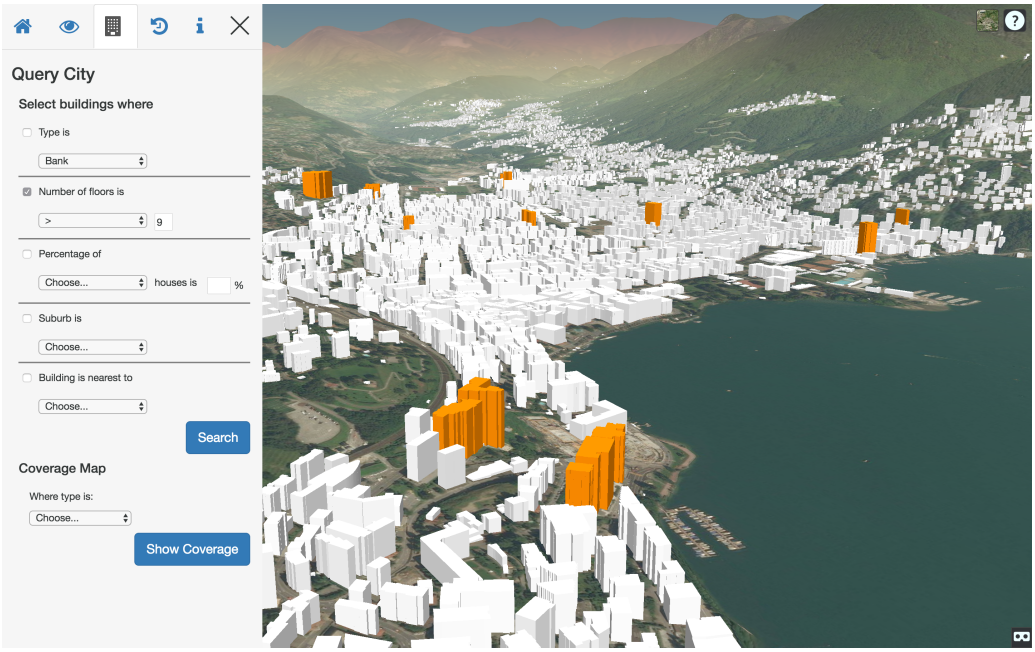


Figure 4.9. Result of the query: "Get all the buildings with more than 9 floors in Lugano"

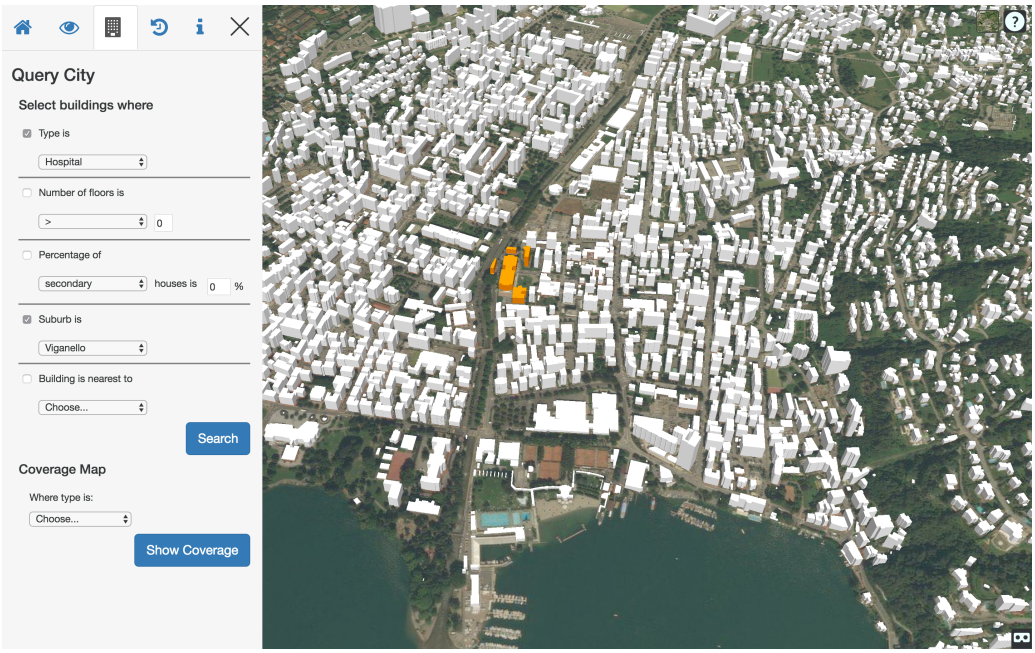


Figure 4.10. Result of the query: "Get all the hospitals in Viganello (Suburb of Lugano)"

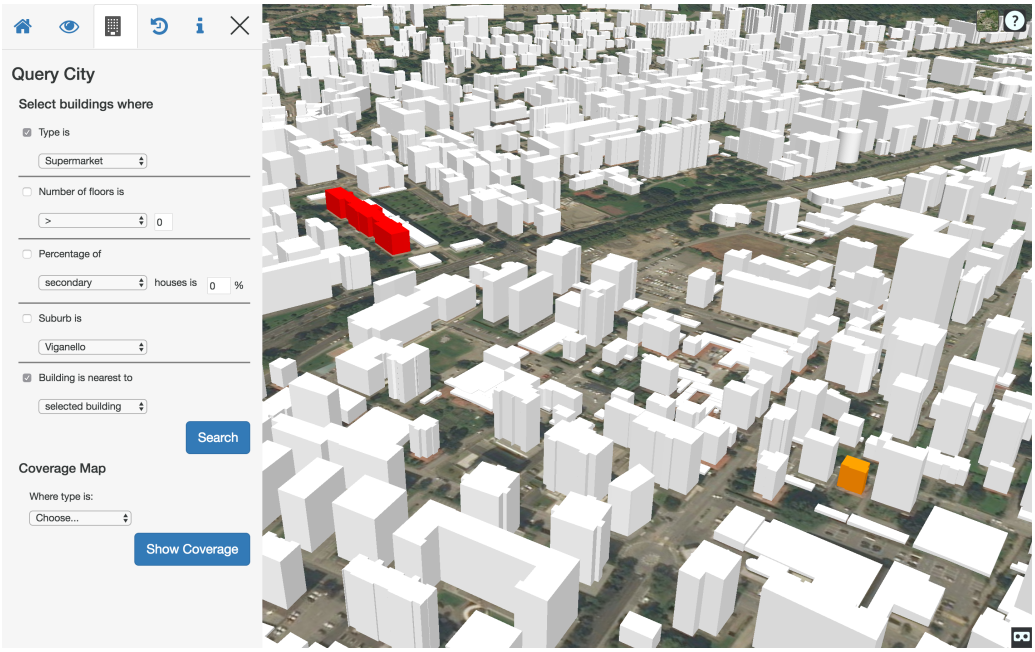


Figure 4.11. Result of the query: "Get nearest Supermarket near the selected building"

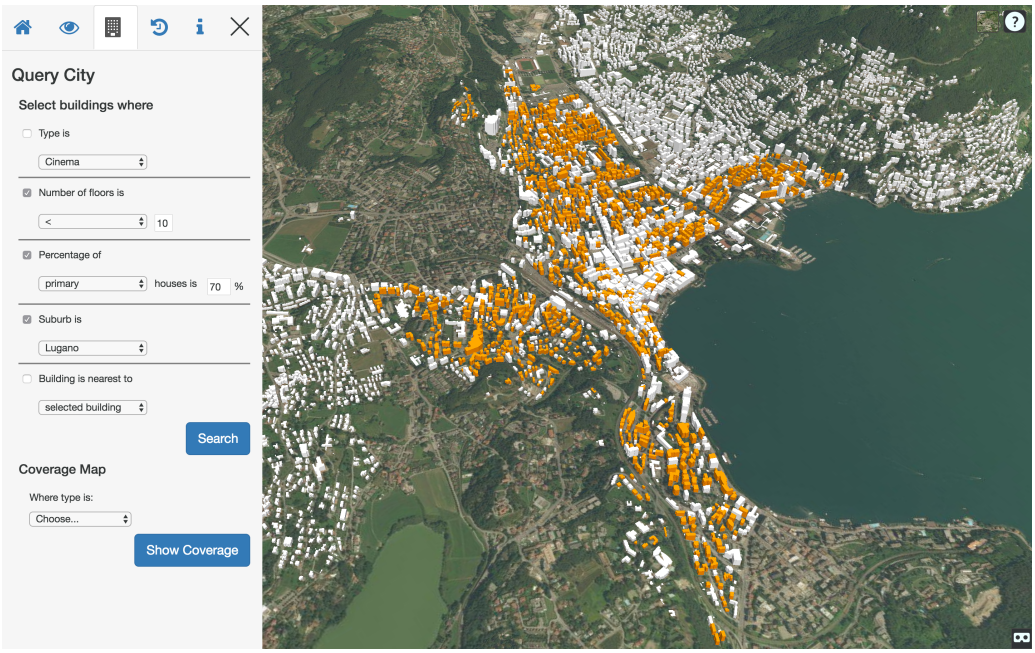


Figure 4.12. Result of the query: "Get all the buildings with less than 10 floors and where the percentage of primary houses is up to 70% in the Suburb of Lugano"

4.3.2 City Gradient Map

Smart–IVC also allows to visualize the coverage of a specific type–building with respect to the entire city. The coverage is grafically shown coloring the buildings differently depending on their distance from a hotspot.

In particular, whenever a user does a coverage query, the hotspot or hotspots (in the case of multiple buildings of the same type) will be colored in red. The other buildings will be colored using an interpolation from "Lime green" (i.e., nearest buildings to the hotspost) to white (i.e., farthest buildings to the hotspot). If a building is inside the coverage area of two hotspot, let us say hotspot_1 and hotspot_2, its color will be determined by the mean between the distance of the building from hotspot_1 and the distance of the building from hotspot_2.

Figures 4.13 and 4.14 show two examples of coverage of particular hotspot in the city of Lugano. The former represents the coverage of banks, the latter the coverage of hospitals in the city.

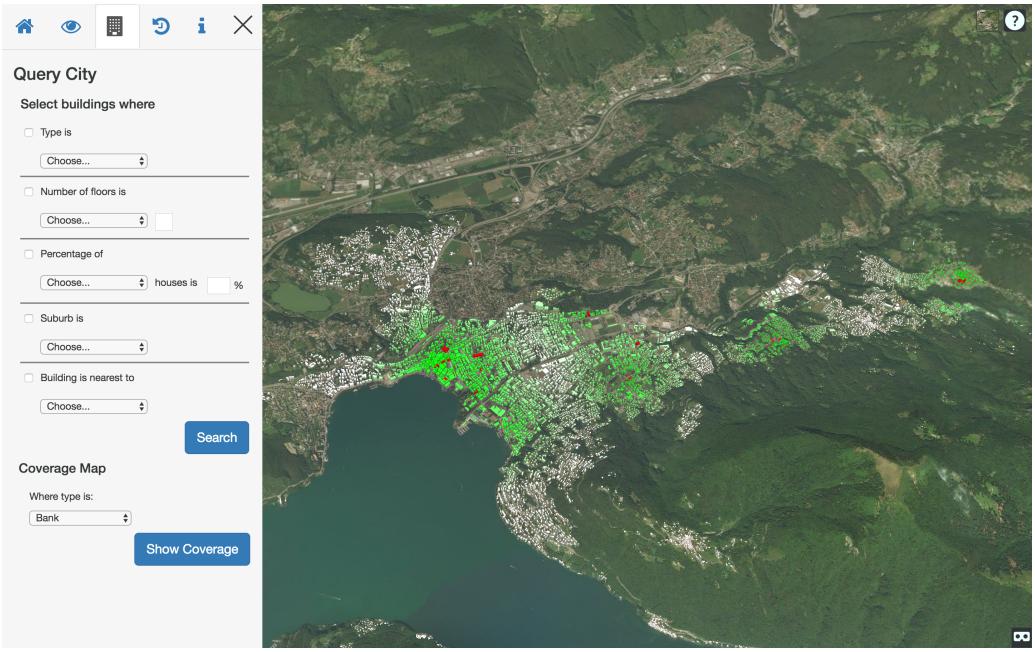


Figure 4.13. Coverage map of banks in the city of Lugano

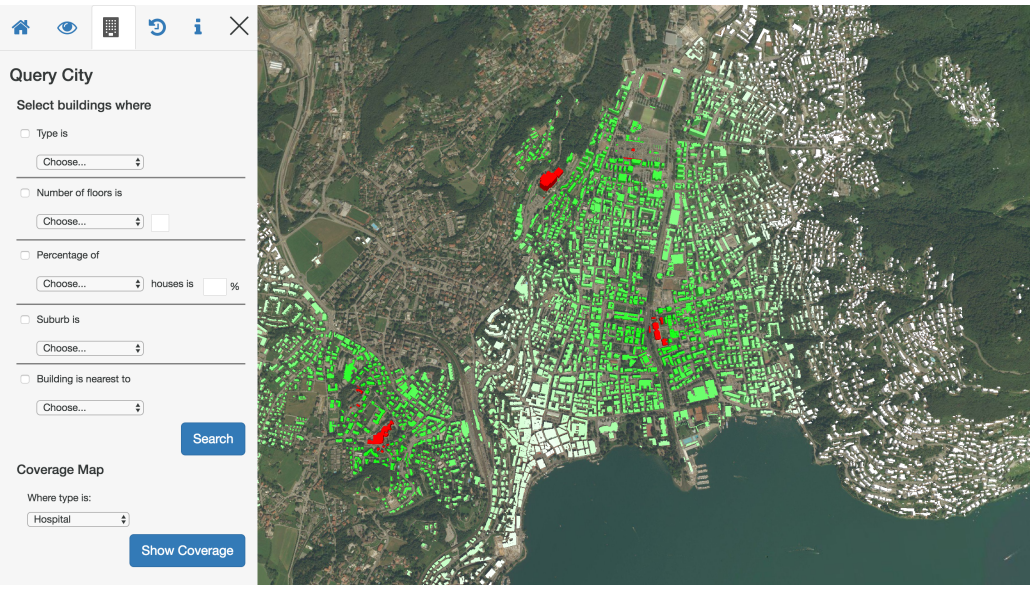


Figure 4.14. Coverage map of hospitals in the city of Lugano

5 Conclusion

5.1 Potential and Limits

In our opinion the tool has great potential. As explained above, even if there are many tools which represent cities in 3D, very few of them are able to allow interactions and almost none of them allow the possibility to do query like presented in Smart–IVC. We think that the possibility to take plain alphanumeric data and gives it a virtual representation which provides visual feedback is the success key of this application.

The main limits of Smart–IVC are the lack of additional data about the urban environment (e.g., traffic data) and the quality of data provided by third party entities (e.g., OpenStreetMap). We did not manage to always find accurate data to inject in the example of Lugano, therefore we were limited to make interactions with the available data. Since not all data was available for each building, where data was missing we had to fill in with OpenStreetMap's information that were not always accurate.

5.2 Future Work and Possible Developments

Right now Smart–IVCis a prototype which could be further developed, since the ways out of this application are really limitless. With some more data available like a more accurate representation of the buildings, it could be possible to result in an even nicer and more precise 3D–visualization. In addition, more information related to buildings, bus stops, actual streets, rivers, bridges, green zones and tunnels could be included. Once other elements are available in the representation further interactions could be implemented raising the level of visual feedback provided by the tool.

Smart–IVC, with its graphical rapresentation and interactivity, has raised interest in both private citizens and the public sector. The City of Lugano stated interest on the potential application of Smart–IVC, and suggested a novel visualization on data that involves primary and secondary residence.

Our future work involves integration with better data sources, like Google Maps. We also plan to integrate more refined 3D models of buildings as provided by Swisstopo.

References