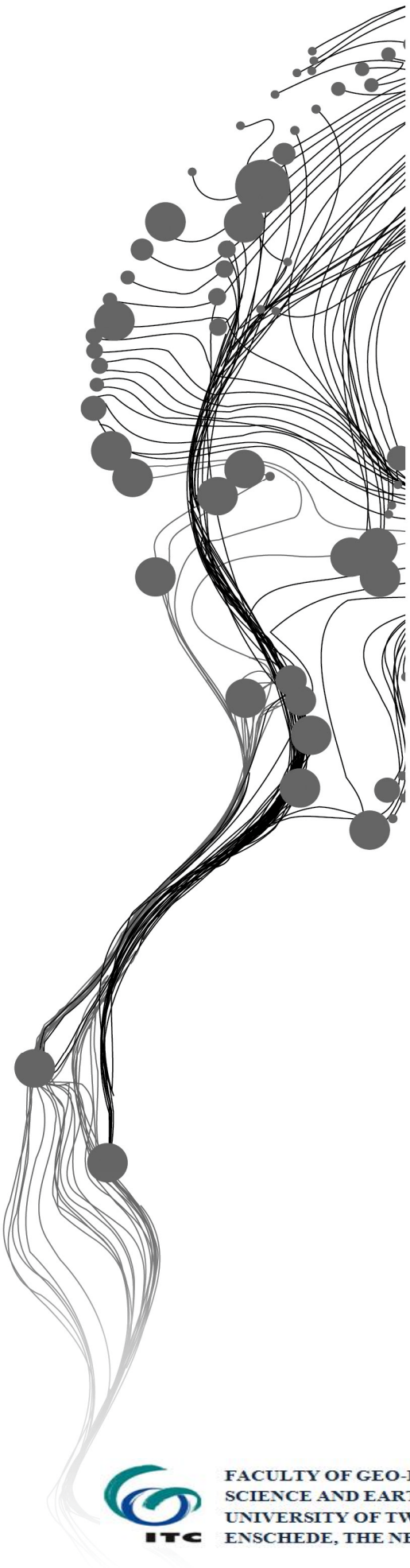
Web based 3D analysis and visualization using HTML5 and WebGL

KANISHK CHATURVEDI
March, 2014

ITC SUPERVISORS
Dr. Javier Morales
Mr. Claudio Piccinini

IIRS SUPERVISOR
Mr. Ashutosh Kumar Jha



Web based 3D analysis and visualization using HTML5 and WebGL

KANISHK CHATURVEDI
Enschede, the Netherlands [March, 2014]

Thesis submitted to the Faculty of Geo-information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

THESIS ASSESSMENT BOARD:

Chair	:	Prof. Dr. Ir. A. Stein
ITC Professor	:	Prof. Dr. M. J. Kraak
External Examiner	:	Mr. Vinod Bothale (NRSC, Hyderabad)
IIRS Supervisor	:	Mr. Ashutosh K. Jha
ITC Supervisors	:	Dr. J. Morales
		Mr. C. Piccinini

OBSERVERS:

IIRS Observer	:	Dr. S. K. Srivastav
ITC Observer	:	Dr. N. A. S. Hamm

FACULTY OF GEO-INFORMATION SCIENCE AND EARTH OBSERVATION, UNIVERSITY OF TWENTE, ENSCHEDE, THE NETHERLANDS

INDIAN INSTITUTE OF REMOTE SENSING
Indian Space Research Organisation
Department of Space, Government of India

DISCLAIMER

This document describes work undertaken as part of a programme of study at the Faculty of Geo-information Science and Earth Observation (ITC), University of Twente, The Netherlands. All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the institute.

Dedicated to my parents…

ABSTRACT

CityGML is a recent standard developed to describe, store and exchange virtual 3D city models. This standard not only allows to understand the spatial properties of urban objects, but also provides a common platform to integrate city level information from different resources and make them accessible to the concerned people. A plethora of software have been developed for processing and visualizing CityGML data, but its visualization on the web is still a challenging area. Although many APIs have been developed to display 3D graphic contents on the web, they work only with certain browsers or with additional browser plug-ins installed. To overcome this limitation, this research focuses on utilizing HTML5 and WebGL. Applying such approach, 3D capabilities can be realized directly in the browser without any need for an additional plug-in or extension. Another benefit is, WebGL provides hardware accelerated 3D functionality on the web, resulting significant performance improvement. This research deals with visualization and analysis of 3D objects of CityGML on a WebGL based virtual globe running on an HTML5 enabled web browser. The primary focus is towards developing a framework to visualize geometry and semantics of 3D city objects on the web based virtual globe with the help of WebGL. Cesium virtual globe, which is an open source JavaScript API based on WebGL, has been considered in this study. The research also includes the development of on-the-fly 3D analysis, performed directly by the client. The emphasis is given to 3D buffer analysis techniques helpful in understanding and preparing for evacuation planning or other emergency scenarios in an urban area. Additionally, the functionality also includes the implementation of 3D operations such as 3D intersection and 3D inside on top of the city model. The results of this research study show that such functionality provides maximum power in the hands of the user. Without depending on the server, such computationally intensive analysis can be performed directly on the client.

Keywords: Virtual city models, 3D visualization, CityGML, KML, HTML5, WebGL, 3D spatial analysis, virtual globes, 3D spatial database

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my IIRS supervisor, Mr. Ashutosh Kumar Jha, for the continuous support and motivation throughout the research phase. His guidance helped me in all the time of research and thesis writing.

I sincerely thank my ITC supervisors Dr. Javier Morales and Mr. Claudio Piccinini for their encouragement and insightful comments, which helped me to focus on the right things from the very beginning of the research.

I would also like to take this opportunity to thank Dr. Y.V.N Krishna Murthy, Director, Indian Institute of Remote Sensing (IIRS), Dehradun for giving me this opportunity to pursue IIRS-ITC Joint education M.Sc. Programme 2012-2014. I am grateful to Mr. P.L.N Raju, Group Head, RS and GID, IIRS for offering enrichment of the project work at all stages. I especially thank all of the faculty and staff at IIRS for their kind support. I would also like to thank Dr. S.K. Srivastava, Head, GID, for his constant support.

Special thanks to the Cesium developer community, who actively responded to my technical queries. Without their help, it would not have been possible to learn the Cesium in such a short span of time.

My heart filled gratitude to all my classmates at IIRS and ITC for making this experience such a valuable and pleasant one. The time I spent with them was special and will always be cherished.

Last but most importantly, I owe this achievement to my family and my best friend Isha, who have been the greatest source of inspiration and hope in my life. They have always been my strength and their love always empower me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Background

The advancement in computer graphics, high computational capacities and web technologies have entirely revolutionized visualization techniques. The involvement of latest trends such as 3D visualization and animation have completely changed the way the information is revealed. With the help of third dimension and dynamic contents, the objects can be visualized and interpreted in a much better way than before. As a result, 3D visualization has become an important part in many sectors, such as engineering, interactive multimedia, medicine and many more. Not surprisingly, the benefits of 3D visualization can also be observed in applications of geographic information science (GIS). Due to its success, numerous applications and advanced tools are being developed for representing and analysing 3D world.

One of the most advanced technological progress in 3D applications of GIS are virtual 3D city models. They allow to understand the spatial properties of urban objects in a more meaningful way. They also provide a common platform to integrate city level information from different resources and make them accessible to the concerned people. Due to 3D representation, the city objects look realistic and hence it is easier for the human to interpret and analyse them (Mao, 2011). The success of virtual 3D city models have led people from both academia and industry to explore and analyse them for real life problems. Consequently, they are widely used in almost every field, varying from urban development, natural resources to disaster management.

1.1.1. CityGML

City Geography Mark-up Language (CityGML) is a recent standard developed to describe, store and exchange virtual 3D city models. It is based on eXtensible Mark-up Language (XML) and represents geometric as well as semantic attributes of a virtual 3D city model. Additionally, as shown in Figure 1-1, it provides functionality to represent the scale of specific object with the help of five level of details (LOD) (Gröger, Kolbe, Nagel, & Häfele, 2012).

- LOD0 represents the digital terrain model in 2.5D.
- LOD1 represents the building models as flat-roofed blocks.
- LOD2 represents the building models with distinct roof structures and thematically differentiated surfaces.
- LOD3 represents the architectural models with more details including walls, roofs, balconies and bays.
- LOD4 represents architectural models with interior structures along with LOD3.

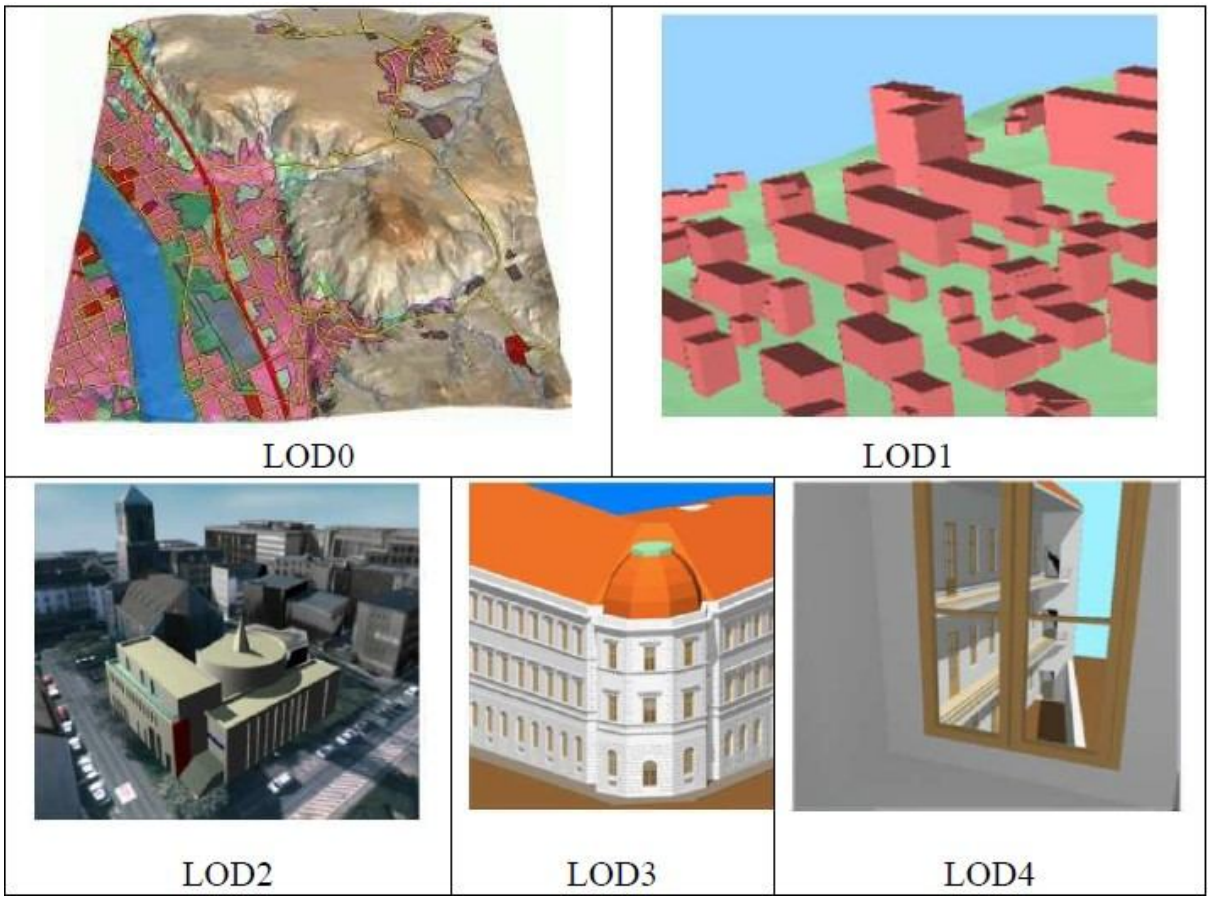


Figure 1-1 : The five LODs defined by CityGML (Source: Gröger et al., 2012)

Owing to a large size and complexity of CityGML files, handling, analysis and visualization of such data have always posed a challenge to people working with city models. Consequently, a plethora of software have been developed for processing and visualizing CityGML files, for example, Safe Software's FME, Bentley Map or Autodesk's LandXplorer (Gesquière & Manin, 2012). These tools have extensive analytical capabilities along with a good performance speed. This is due to the fact that these tools require to be installed on user's hardware machine and thus, they utilize hardware's high processing speed and graphics card memory.

1.1.2. Web based 3D visualization formats

The use of CityGML files on the web has opened doors for a large number of useful applications providing better information sharing and dissemination. It has also facilitated users to perform various analyses, queries or navigation on virtual 3D city models. However, due to large size and complex structures of CityGML files, their visualization on the web browser can be time-consuming and cause inefficiency related to geometry and semantics. As a result, visualization of CityGML files on the web has become an essential area of research today (Prieto, Izkara, & del Hoyo, 2012).

Prior to CityGML, the major developments to display 3D graphics content on the web include various application program interfaces (APIs) such as Flash, O3D, VRML, KML and X3D. Flash

plug-in enables web browsers to display multimedia contents such as images, 3D models and animations. O3D is an open source JavaScript API developed by Google to display 3D contents on web browser. Virtual Reality Mark-up Language (VRML) is a major development towards interoperability of 3D data on the web environment. Keyhole Mark-up Language (KML), adopted by Google Earth, allows to represent geo-location along with 3D COLLADA models and other multimedia formats. X3D is the successor of VRML with the addition of XML based file format. (Ortiz, 2010).

1.1.3. HTML5 and WebGL

With the rapid transition of desktop based GIS applications to the web, the urgent need has arisen to develop platform-independent web applications. The fifth revision of HTML, HTML5, has turned out to be an important milestone in the same direction. HTML5 (Hickson & Hyatt, 2010, p. 5) is an open standard format and provides common platform for applications to be developed and used on the web. It involves several useful elements such as canvas, scalable vector graphics (SVG), geo-location, web workers and web sockets. These elements can be used independently to develop robust GIS applications on the web browser.

WebGL (Marrin, 2011) is an extension of HTML5 canvas element, which is now widely used for developing web applications requiring 3D visualization. It is a 3D graphics API, written in low level language and is based on OpenGL ES 2.0. To avoid complex low level programming, several WebGL based frameworks have been developed, providing ease of development. Some of the important frameworks worth mentioning are:

- Three.js: Three.js is a JavaScript based library, which creates 3D contents on the web browser with a very low level of complexity. It is lightweight in nature and can perform rendering with the help of HTML5 canvas, SVG and WebGL (Mrdoob, 2013).
- Scene.js: Scene.js is a WebGL based library, which uses JavaScript for 3D visualization on web browsers. It differs from Three.js as it is intended towards fast rendering of large number of individually pickable objects. This features makes this library useful for engineering and data visualization applications (Kay, 2013).
- GLGE: GLGE is a JavaScript based library, which abstracts the low level instructions from the developer and wraps them into meaningful instructions. It is also intended to provide ease of development of 3D content (Brunt, 2013).

1.2. Motivation and Problem Statement

Existing 3D visualization solutions work only with certain browsers or with additional browser plug-in installed. This limitation can be overcome by assembling the visualization using WebGL and HTML5. Applying such approach, 3D capabilities can now be realized directly in the browser without any need for an additional plug-in or extension (Iglesias, 2012). Another benefit of

WebGL over other technologies is that it utilizes hardware's graphics card memory for displaying and performing operations on 3D contents and hence, it provides hardware accelerated 3D functionality on the web. (Taraldsvik, 2011).

Consequently, the combination of HTML5 and WebGL is being widely used by researchers to solve complex problems. Many virtual globes have been developed utilizing this combination, for example Cesium Virtual Globe (Analytics Graphics, Inc., 2011), WebGL Earth (Klokan Technologies, 2011) and Open Web Globe (Christen & Nebiker, 2011). Even the most recent development MapsGL by Google Maps utilizes WebGL technology to add 3D graphics to the browser without the need to install additional plug-in (Google, 2013). It has also been proposed that with the help of WebGL and 3D web services, computationally intensive algorithms such as 3D view shed analysis, 3D density, spread analysis and 3D buffer can be implemented on virtual 3D city models with better performance on web browsers (Moser, Albrecht, & Kosar, 2010). These algorithms are extremely helpful in understanding and preparing for emergency scenarios in case of a disaster in urban area. 3D queries can also be performed on a web browser with the help of spatial databases (Koussa & Koehl, 2010).

However, there are a few limitations in working with WebGL. Even though it is supported by major browsers such as Google Chrome, Mozilla Firefox, Safari and Opera, there are browsers such as Internet Explorer 11 which do not support WebGL completely. Moreover, WebGL is a low level API as it is designed to interact directly with graphics card. Hence, devices with low graphics card memory may pose significant performance issues. Additionally, to avoid complex low level programming, use of pre-defined WebGL libraries such as Three.js, Scene.js and GLGE is mandatory (Parisi, 2012).

Considering the challenges in working with complex 3D city models on the web environment and taking into account the advantages of WebGL technology, this study is motivated towards developing a web based tool to visualize as well as analyse complex 3D city models. The primary focus is on creating on-the-fly 3D buffer zones on a virtual 3D city model on a web browser. Further, the 3D analysis can be performed on the objects to determine the buildings which either intersect or are completely inside the buffer zones. Such implementation can be very helpful for field workers and decision makers to understand and be well prepared for evacuation planning or other emergency scenarios in an urban area.

1.3. Research Identification

1.3.1. Research objectives

The main objective of the proposed research is to develop functionality to implement computationally intensive algorithms, particularly, 3D buffer analysis with the help of WebGL API on HTML5 enabled web browser.

Sub-objectives:

(1) To identify suitable method to visualize CityGML file on a web browser with the help of WebGL API.
(2) To develop client-server architecture to achieve progressive visualization of a CityGML file on the client.
(3) To study, understand and develop functionality to implement 3D buffer zone creation algorithm on the client.
(4) To assess the performance and visual quality of mentioned algorithm implemented on the web environment.

However, owing to time constraints, this research shall be limited in following areas:

(1) It does not focus on developing a complete GIS system, but an implementation of 3D analysis and visualization with the help of WebGL API.
(2) Out of many 3D algorithms such as 3D proximity analysis and 3D viewshed analysis, it focuses only on 3D buffer zone creation and 3D operations on virtual 3D city model.

1.3.2. Research questions

(1) What are the different techniques proposed in literature to visualize geometric and semantic attributes of CityGML file on a web browser efficiently?
(2) Which is the best suitable WebGL based library to perform analysis and visualization of CityGML file on a web browser?
(3) How does the involvement of WebGL based virtual globe improve performance of computationally intensive algorithms?
(4) Which part of the functionality to be implemented on client side or server side in order to achieve best results?

1.4. Innovation aimed at

The innovation expected from this research is to perform 3D analysis on an HTML5 enabled web browser with the help of WebGL API. Based on the above research objectives and questions, the proposed web based framework allows the user to visualize the geometric and semantic attributes of the 3D city objects of CityGML. Also, it allows to create on-the-fly 3D buffer zones on top of city objects and performing 3D operations to determine the city objects which are either completely inside or partially intersect the buffer zones. As per literature review, there is no implementation of such algorithms on a web based environment without the need of a browser plug-in.

1.5. Thesis Structure

The research work is organized as follows:

Chapter 1: Introduction, this section presents general overview about the research work. It describes the basic idea of topic, motivation, problem statement, research objectives, and research questions.

Chapter 2: Literature Review, this chapter deals with background of the study and literature review. It briefly explains important components of the research: virtual 3D city models, virtual globes, 3D analysis techniques and 3D geometry on the web.

Chapter 3: Data and Usability, this chapter describes the data, hardware and software tools, used during the research. Considering the research objectives, it also highlights the usability of the application by explaining the use case diagram for interaction between the user and the application.

Chapter 4: Design and Implementation, this chapter details about the high level architecture of the functionality and detailed implementation of each component of the architecture.

Chapter 5: Results and Discussions, this chapter discusses the results obtained at each implementation step, its discussion and analysis.

Chapter 6: Conclusion and Recommendations, this section provides the answer of the research questions in concluded form and highlights recommendations for further study.

2. LITERATURE REVIEW

2.1. Virtual 3D city models

CityGML is the most suitable standard to represent virtual 3D city models. It not only represents geometry, but also the semantics of the city objects. Considering the benefits in working with CityGML, several studies have contributed in visualization of virtual 3D city models using CityGML data. Due to its extremely large size and complexity, visualization of CityGML on the web environment is not as mature as on desktop based applications. In order to visualize CityGML data on the web, several 3D standards such as X3D, JSON and KML/COLLADA have been used.

2.1.1. X3D

X3D is an open standard format based on XML syntax. It is the successor of VRML format and is used to represent 3D scenes on the web environment. Several studies have been conducted to visualize CityGML data on the web browser with the help of X3D. (Mao, 2011) developed a framework to provide online visualization of 3D city models on the web environment. According to the research, the CityGML is chosen as the appropriate city model source and is stored on the server for analysis using Java classes. Further, 3D visualization is achieved on WebGL based browser with the help of 3D scenes in X3D. It also helps in providing real time user interaction on the browser as X3D nodes are directly integrated into HTML5 DOM content with the help of X3DOM. Figure 2-1 shows the framework, developed in the study:

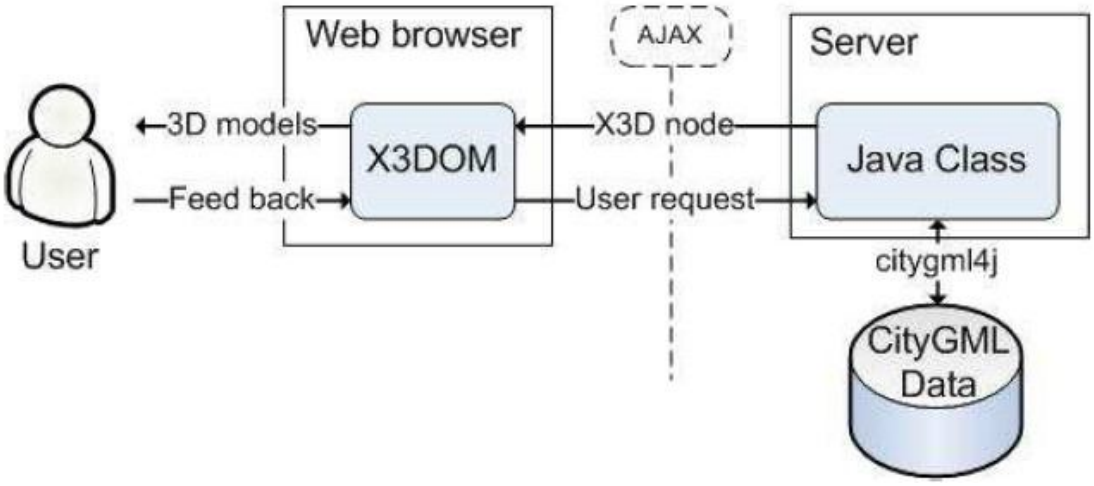


Figure 2-1: Framework for online visualization of CityGML (Source: Mao, 2011)

Supporting the importance of X3D, (Prieto et al., 2012) also developed a complete prototype to visualize CityGML files without plugins. Unlike previous work, this research incorporates the usage of 3D web services such as W3DS instead of Java classes. It allows retrieval of required

parts of the whole city and sharing the information in real time. Consequently, this approach provides better interoperability. The methodology is shown in Figure 2-2:

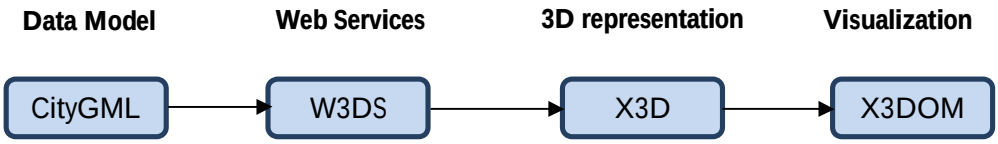


Figure 2-2: Methodology for visualization of CityGML (Source: Prieto et al., 2012)

2.1.2. JSON

JavaScript Object Notation (JSON) is an open standard format to be used for data interoperability. With the evolution of JSON as a lightweight data exchange format, the X3D has been replaced by JSON in many studies. 3DPIE (OGC, 2012) is a large project proposed by Open Geospatial Consortium (OGC) to carry out processing of 3D information in the web environment. During the project, in order to portray a CityGML file, three tests have been performed and compared.

- In the first test, the entire CityGML file is fetched from a WFS server on a thick client based on C++.
- In the second test, the CityGML file is first processed on a server using JAXB parser, so that only the required part can be fetched on the client.
- In the third test, the CityGML stream is replaced with a JSON stream as it is well understood on the client with the help of three.js library and is easy to use with WebGL.

Extending the third test, (Gesquière & Manin, 2012) adopts the classic tile-based approach to work on CityGML files. The CityGML file is broken into several tiles and each tile is transformed into JSON, which is stored on the server. The client can make request to the server on the basis of specific tile and consequently, the server responds with JSON file for that tile. The functionality can be described in Figure 2-3:

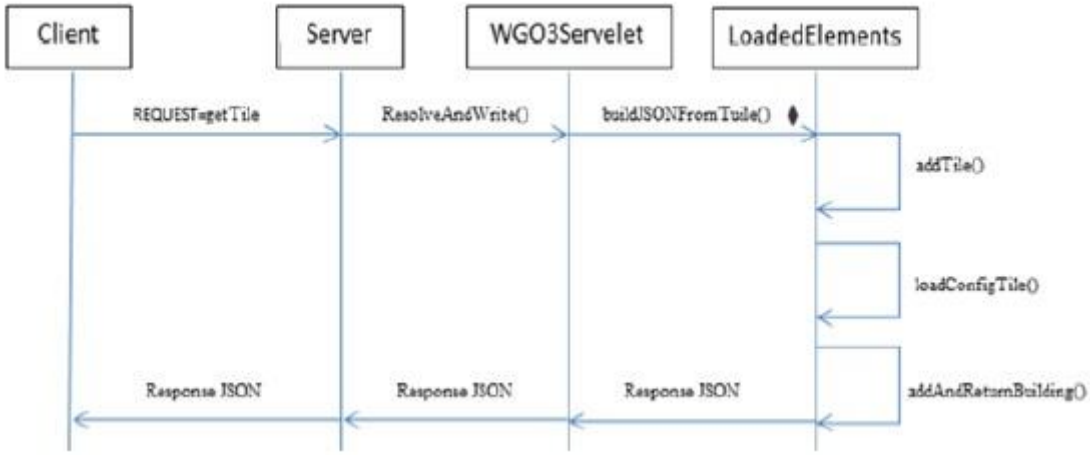


Figure 2-3: Tile based communication between client and server (Source: Gesquière & Manin, 2012)

The major advantage of tile-based approach is that it allows progressive visualization, which means, only the area required to be visualized will be fetched from the server. Moreover, JSON is better than X3D because it allows the semantic and attributes based query directly on the client side.

Similarly, (Federico Prandi, 2013) explains the working of a project named iSCOPE (interoperable Smart City services through an Open Platform for urban Ecosystems). Through this project, various 'smart city' services have been developed based on interoperable 3D CityGML files. The architecture for this project also contains tiles of CityGML files, instead of a spatial database, on server side. The client side again makes request on the basis of specific tile and achieves progressive visualization.

2.1.3. KML/COLLADA

Keyhole Mark-up Language (KML) is an XML based file format to represent geo-location on web applications such as Google Earth and Google Maps. Collaborative Design Activity (COLLADA) is also an open standard XML scheme to represent 3D models. With COLLADA integration, KML is being successfully used for 3D visualization on the web browsers.

KML/COLLADA integration makes it possible to visualize CityGML data with the help of a software 3DCityDB (Kolbe, König, Claus, & Stadler, 2013). It is an open source software, which allows to import CityGML dataset to a 3D geo database such as PostGIS2.0 and Oracle Spatial. The major features of this software are:

- It supports all kind of level of details and appearances.
- It also supports complex terrain and building models.
- With the import of CityGML to a geo-database, it allows direct analysis on a city model.
- It also allows to export CityGML from the geo-database.

With the help of a plug-in 3DCityDB Importer/Exporter, the CityGML data can be easily exported to KML/COLLADA. With the help of this conversion, the geometric and semantic attributes of CityGML data can be visualized on applications such as Google Earth and Google Map. 3DCityDB Importer/Exporter also allows to generate tiles for the CityGML file and store the information of tiles in the form of JSON.

2.2. Virtual Globe

With the advancement in internet based applications, virtual globe has become the new medium to visualize and interact with global geospatial data. It not only reduces the effort of manually accessing archives of satellite imageries, but also allows users to interact and extract content from the globe in real time (Elvidge & Tuttle, 2008). To develop cross-platform, cross-browser applications, several WebGL based virtual globes have been developed. Some of the globes, worth mentioning, are:

2.2.1. WebGL Earth

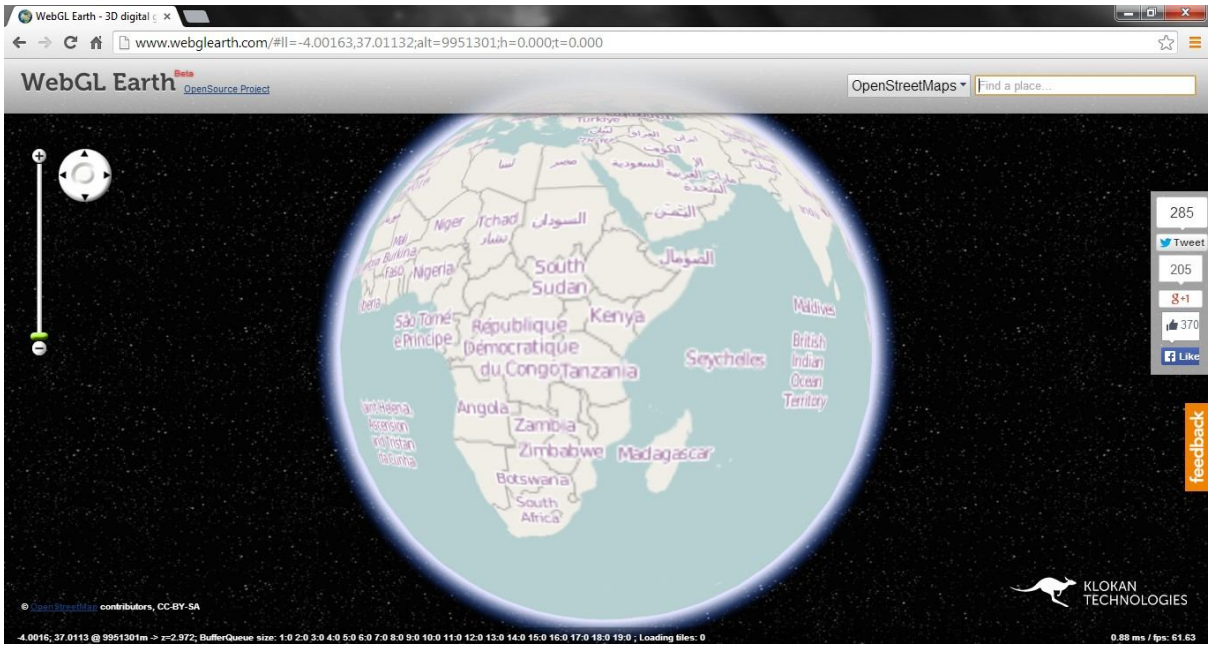


Figure 2-4: WebGL Earth appearance

WebGL Earth (Klokan Technologies, 2011) is an open source software developed for visualization of maps, satellite imagery and aerial photography on top of a virtual terrain. It is based on WebGL standard specifications, which allows to build customized applications without the need of plugins.

Features

- It allows camera-dependent functionalities such as rotation, zoom and tilt.
- It supports existing maps from multiple sources such as OpenStreetMap and Bing.
- It also supports custom map tiles for the earth or other planets.

2.2.2. OpenWebGlobe

Figure 2-5: OpenWebGlobe appearance

OpenWebGlobe (Christen & Nebiker, 2011) is a WebGL based 3D geobrowser, which allows to process and visualize very large volumes of geospatial data. It consists of a complete SDK to develop web based applications without the need of plugin.

It supports various forms of image data, elevation data, point of interest and 3D models. It is mainly useful for processing very large amount of data, upto terabytes, in highly parallel and scalable computing environments. But it is different from other globes in the way that the data must be pre-processed before loading on the internet.

2.2.3. Cesium

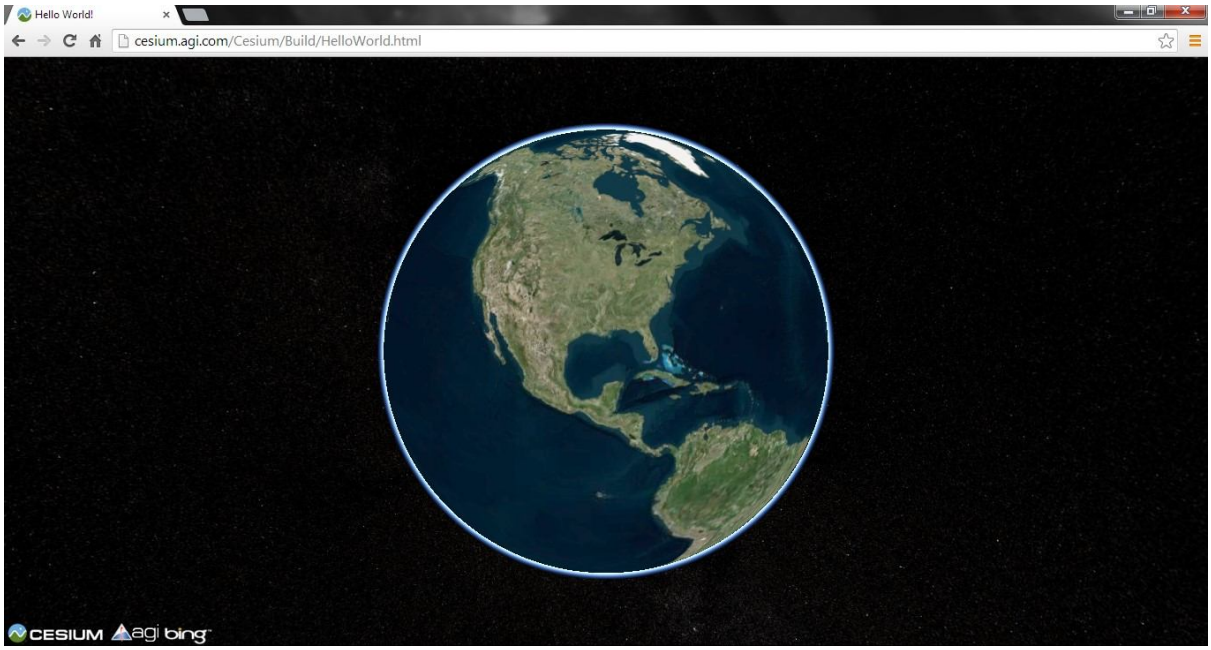


Figure 2-6: Cesium Virtual globe's appearance

Cesium (Analytics Graphics, Inc., 2011) is an open-source JavaScript library to create 3D virtual globes as well as 2D maps on a web browser. It utilizes WebGL to provide hardware acceleration and plugin independence and provides cross-platform and cross-browser functionality.

Features

- It is most suitable for dynamic geospatial data visualization with the help of Cesium Language (CZML). CZML is a JSON based schema, which describes geospatial data along with their properties that vary over time.
- It can integrate layer imageries from different sources, such as OpenStreetMap, Bing Maps, ArcGIS MapServer and standard image files. Even, the external WMS and TMS can be integrated. Each layer, then, can be visualized according to specific brightness, contrast or saturation.
- It includes extensive libraries which support 2D as well as 3D geometries. The user can draw polyline, polygon, ellipsoid, sphere, labels, billboards and sensors.
- It supports data imports from KML, ESRI Shapefiles and JSON.
- It includes handlers to control mouse/keyboard events, camera movements and zoom and pan the virtual globe.
- It supports extensive materials to describe the surface appearance of the objects. It also supports custom materials for the objects.
- It supports math libraries to support major reference frames such as World Geodetic System (WGS84) and International Celestial Reference Frame (ICRF). The libraries also support conversions of coordinates and Cartesians.

Cesium provides a higher level of abstraction, making it a highly easy-to-use tool as compared to its counterparts. Its architecture includes four main layers as shown in Figure 2-7:

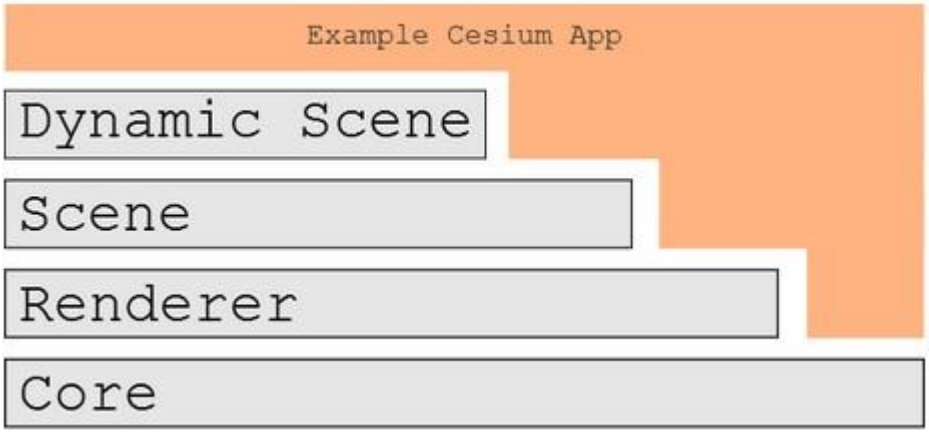


Figure 2-7: Cesium architecture. (Source: Analytics Graphics, Inc., 2011)

The different layers of the architecture are responsible to add specific functionality and to raise the level of abstraction. The layer is usually dependent on the layers underneath it. The details are:

- Core – This is the lowest layer in Cesium and includes mainly low-level functions. These functions majorly include computations and calculations such as mathematical conversions, transformations and projections.
- Renderer – This layer is a thin abstraction over WebGL. It includes already available GLSL functions to provide shader programs, textures and buffers.
- Scene – This layer is mainly responsible to provide overall functionality of the globe. It includes high-level globe and map constructs such as 3D globe or map, handling layer imageries from multiple sources, creation of geometries and materials, camera control and animation.
- Dynamic Scene – This is the top-most layer of Cesium, which provides dynamic visualization of the data with the help of CZML. It allows to store the data in dynamic objects, loads and renders the dynamic objects altogether instead of rendering every frame.

Cesium virtual globe is based on Scene.js JavaScript 3D render engine and utilizes several third party scripts such as require.js, dojo.js and almond.js, which helps in developing highly interactive and user-friendly applications.

2.2.4. Reason for selecting Cesium

Since all the proposed solutions are still in early development stage, it is quite difficult to select any one of the solution to provide entire functionality. Nevertheless, considering the complexity of CityGML datasets and their visualization and analysis, Cesium virtual globe has been chosen due to following reasons:

- Extensive collection of libraries: Cesium includes an extensive collection of libraries, related to import data from multiple sources, mathematical computations, creation of 3D

geometries, camera and flight control. Overall, it makes easier to work with complex 3D city models.

- Improved performance: Cesium works directly over WebGL, providing hardware-accelerated graphics. Due to this reason, performance can be significantly improved while handling large complex 3D city models.
- Ease of use: The development on Cesium is comparatively easy due to well-structured codebase and documentation. Each library is very well explained with examples and tutorials. Cesium also includes an interesting application called 'Sandcastle', which provides live coding on the browser.

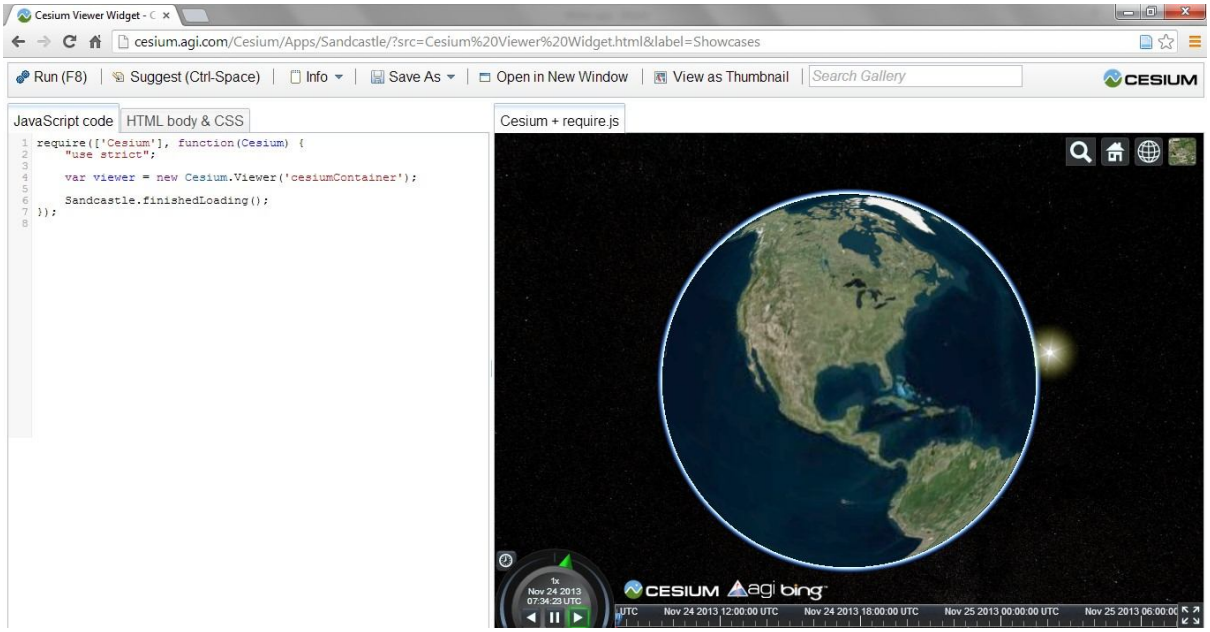


Figure 2-8: Cesium sandcastle appearance

- Open source: Cesium is an open-source JavaScript library under the Apache 2.0 license. As it is free for commercial and non-commercial use, it is possible to add new features modify the existing features as per research's requirements.
- Active community: Cesium involves an active discussion forum, discussing and working on several issues and improvement suggestions. It also ensures a release every month, incorporating the issues and suggestions put in discussion forum, making it an up-to-date tool.
- 3D GIS capabilities: Although the primary intention to develop Cesium was the visualization of geo-spatial data, its wide collection of libraries enables to perform various GIS analyses on the globe. Additionally, with the help of WebGL, it is also possible to perform computationally intensive algorithms on the globe.

2.3. 3D Analysis

With the advancement in 3D visualization techniques, it has been possible to develop 3D GIS analysis capabilities, which are quite helpful in urban planning, disaster management and environmental planning. In the context of applications related to 3D city models, (Moser et al., 2010) have categorized 3D GIS analysis in four broad aspects:

2.3.1. Spread analysis

The paper mentions that with the help of spread analysis in 3D city models, the emission of traffic pollutants from various sources can be identified and the dynamic air flow of the pollutants in the city model can be analysed. Such analysis can be helpful in identifying heavy pollution or no pollution zones and in turn, helpful in decision making process in urban planning scenarios.

2.3.2. 3D density

3D density mainly deals with disaggregation of the data. According to the paper, such analysis can be very helpful in emergency management scenarios. In order to find the hazard zones in a building complex, population can be grouped according to different factors such as day and night population distribution and the impacting factors like weekend and rush hour traffic. Further, 3D disaggregation of such statistical 2D population can be done and compared.

2.3.3. Visibility analysis

Visibility analysis deals with identification of points or areas that are visible to an observer standing at a specific point in the city. Due to the involvement of tall buildings and several objects in an urban model, this is the most common analysis in urban planning processes. The paper briefly explains that visibility analysis can be helpful to provide security to the airport in an urban area. By combining viewshed analysis applied from different observer points to the airport apron and runway, the suitable locations for antenna positions can be identified.

2.3.4. Proximity or buffer analysis

Proximity or buffer analysis deals with identifying surrounding area of a specific measurement around a point or line. The paper briefly explains that such analysis can help urban planners to set up a new station on the route of a subway in an urban area. As buffer analysis is the part of research objective, in-depth understanding of the analysis is of special interest to the author of this research.

Furthermore, (Walenciak, Stollberg, Neubauer, & Zipf, 2009) explains the implementation of 3D buffer analysis on a web based environment. The research identifies a 3D-WPS "BombThreatScenario3D", which allows an end user to input the location of a bomb in 3D coordinates and the explosive force of bomb. The WPS performs the calculation and generates two buffer zones, the security and the danger zone, around the point specified by the user. These buffers are generated in the form of transparent spheres and are stored as Virtual Reality Modelling Language (VRML), which sre sent to the client end. The buffer zones can be visualized on client end as shown in Figure 2-9:

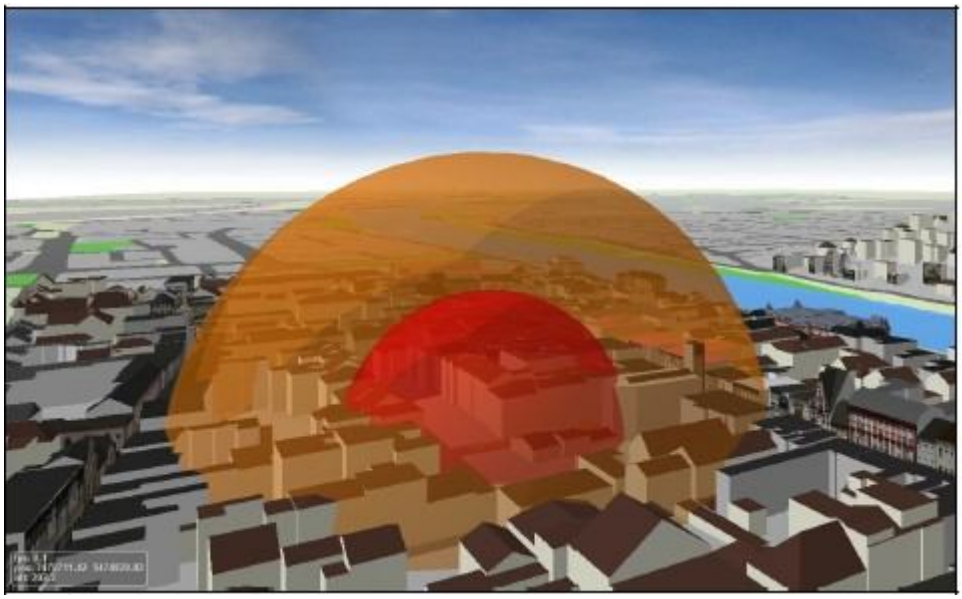


Figure 2-9: Visualization of 3D buffer zones (Source: Walenciak et al., 2009)

According to the research, there are a few limitations with this approach. The overall processing is done on the server side and the role of the client side is just to visualize the buffer zones. In order to provide more interaction with 3D objects, it is beneficial to perform on-the-fly processing at the client side. Additionally, VRML does not allow the integration of semantic attributes of the city models. Therefore, it is better to use different data exchange formats based on GML.

In the context of 3D city models, sometimes buffer analysis, alone, is not sufficient and is supported by various 3D operations such as 3D intersection, 3D union, 3D distance and many more. (Moser et al., 2010) describes the application of 3D buffer for selection of a route of subway to a new station. First, the route of the subway to the new station is identified and the objects in the vicinity of the route, such as basement, wastewater pipes or power lines are selected with the help of 3D distance operation. Then, the 3D buffer operation is applied to the distance in order to find the interfering volume. Lastly, the interfering volume is subtracted from potential volume with the help of 3D intersection, which leaves the appropriate space for the placement of the subway tunnel. In this way, the complete analysis can be done with the help of various operations.

(Shephard, 2010) has described various 3D analyses operations in accordance with ArcGIS 10. Some of the important analyses in relevance to this research are:

2.3.4.1 3D Buffer

This analysis is performed by 'Buffer 3D' tool in ArcGIS 10. It is useful for volumetric computations and generates spheres for point inputs and cylindrical features for line inputs.

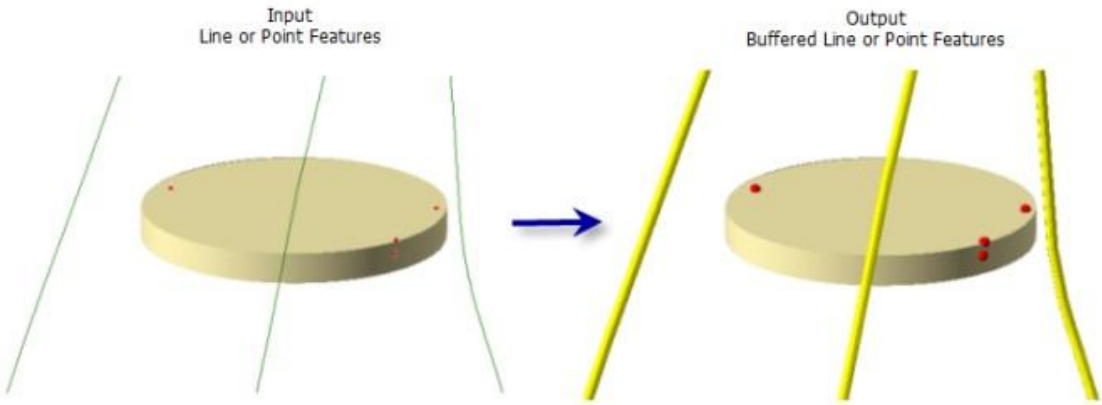


Figure 2-10: 3D Buffer

2.3.4.2 3D Intersection

This analysis is performed with the help of the tool 'Intersect 3D' in ArcGIS 10. It computes the geometric intersection of the two volumes that overlap either completely or partially.

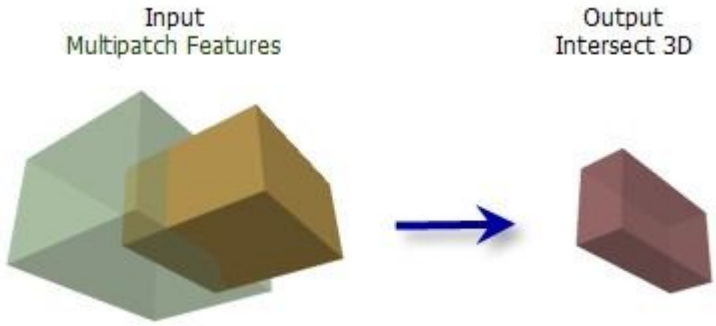


Figure 2-11: 3D intersection

2.3.4.3 3D Difference

This analysis is performed by 'Difference 3D' tool in ArcGIS 10. It computes the geometric intersection of the two volumes that overlap completely or partially, then subtracts the resulting volume from the volume to be considered for the difference.

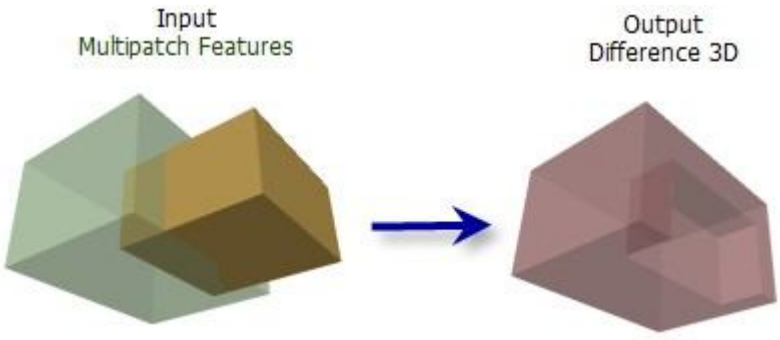


Figure 2-12: 3D Difference

2.3.4.4 3D Union

This analysis is performed by 'Union 3D' tool in ArcGIS 10. It computes the geometric intersection of the two volumes that overlap completely or partially, then aggregates the volumes together and create a single volume.

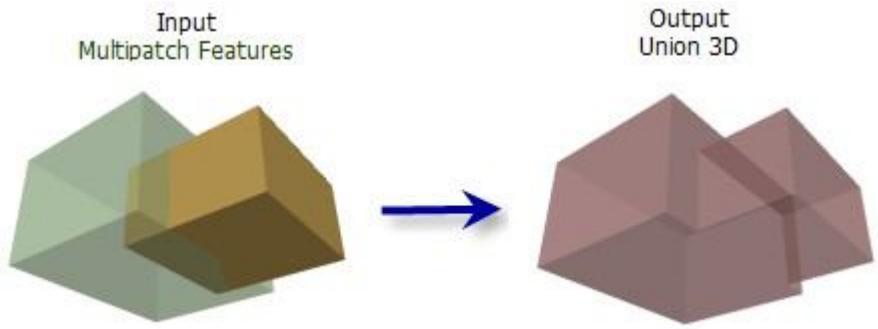


Figure 2-13: 3D Union

2.4. 3D Geometry on the web

This section is dedicated to the techniques for visualization of 3D geometries on the web. The creation of 3D geometries on the web is still at an early stage. The following sub-sections explain theoretical concepts of the 3D geometries and their implementation with JavaScript in Cesium API. Although it is possible to create any type of geometries, but considering the scope of this research, this section focuses only on 3D polygon, sphere and axis aligned bounding box.

2.4.1. 3D polygon

The 3D building in LOD1 is generally represented as 3D polygon, which is also called polyhedron. A polyhedron is a closed region, containing multiple faces, made up by edges in such a way that each edge joins only two faces (Ericson, 2005). The representation of 3D polygon is shown in Figure 2-14:

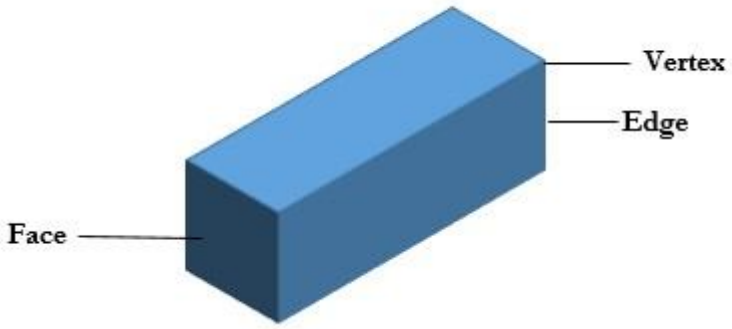


Figure 2-14: 3D polygon

Cesium provides in-built functions to draw such 3D polygons. It is a 3-step process. First, the specification of the polygon geometry is provided. The list of parameters required is as mentioned in Table 2-1:

Table 2-1: Parameters of 3D polygon geometry (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Polygon coordinates	It is an array of coordinates (longitude, latitude and height) of building vertices
2	Extruded Height	It allows to extrude the polygon geometry by providing the height value.
3	Vertex Format	It allows to set specific format to the vertex, for example, 3D position or specific colour
4	Per position Height	It considers the z coordinate of each vertex of the building

Afterwards, the geometry instance is created, positioning the geometry on the world coordinates and providing specific attributes. The list of parameters required is listed in Table 2-2:

Table 2-2: Parameters of 3D polygon geometry Instance (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Polygon Geometry	It is the specific polygon geometry.
2	Model Matrix	It allows to position the polygon on the world coordinates by transformation.
3	ID	It is the unique ID for the geometry instance which distinguishes it from other geometries.
4	Attributes	It allows to provide specific colour attribute to individual geometry.

Once the geometry instance is defined, it can be stored as a primitive. Cesium allows to store multiple geometries in a single primitive, which appear as a complete scene. To create a primitive, the list of parameters required is mentioned in Table 2-3:

Table 2-3: Parameters of 3D polygon primitive (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Geometry Instance	It is the instance of the geometry with a unique ID
2	Appearance	It is responsible for rendering the geometry using low level GLSL vertex and fragment shaders. In other words, the geometry is rendered utilizing WebGL.

In this way, multiple polygon geometries can be created and rendered at the same time, which can further be visualized on the virtual globe.

2.4.2. Sphere

Sphere is a bounding volume which is perfectly symmetrical. It requires centre and radius to be created.

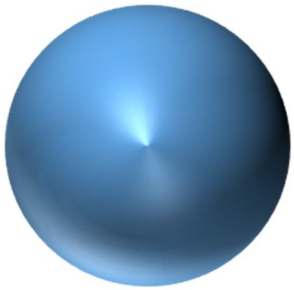


Figure 2-15: Sphere

It can also be created using JavaScript through Cesium. Table 2-4 lists the parameters required to provide specification of sphere geometry:

Table 2-4: Parameters of sphere geometry (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Radius	It is the radius of the sphere
2	Vertex Format	It allows to set specific format to the vertex, for example, 3D position or specific colour

Then, similar to polygon geometry, geometry instance is created. The list of parameters required are mentioned in Table 2-5:

Table 2-5: Parameters of sphere geometry instance (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Sphere Geometry	It is the specific polygon geometry.
2	Model Matrix	It allows to position centre of the sphere on the world coordinates by transformation.
3	ID	It is the unique ID for the geometry instance which distinguishes it from other geometries.
4	Attributes	It allows to provide specific colour attribute to individual geometry.

In the end, in the same way, the primitive is created for the geometry instance, which renders the sphere on the globe.

2.4.3. Axis Aligned Bounding Box

Axis Aligned Bounding Box (AABB) is a type of bounding volume in the form of a six sided rectangular block, which tightly bounds the geometry. The face normal of all the faces of the box

are always parallel with the axes of the coordinate system. In this way, in case of geometries on the same plane, this approach provides most optimum fit and fastest overlap check in comparison to its counterpart techniques. It generally includes a centre point and minimum and maximum points with x, y and z coordinates (Ericson, 2005).

Cesium contains an in-built JavaScript library to create the AABB for the geometry. By providing just the coordinates of the vertices of the geometry, the AABB can be created which contains following three parameters:

Table 2-6: Parameters of object oriented bounding box (Source: Analytics Graphics, Inc., 2011)

S. N.	Parameter	Purpose
1	Minimum	The minimum point of the box with x, y and z axes
2	Maximum	The maximum point of the box with x, y and z axes
3	Centre	The centre point of the box

Once the AABB of the geometry is created, its intersection with different geometry such as sphere, can be performed, which is explained in more detail in section 4.4.3.4.

3. DATA AND USABILITY

This chapter is divided into three sections; the first section describes the study area and the datasets used in the research. The second section describes the hardware and software tools required to successfully carry out research activities. The third section explains application requirements in terms of usability. It includes the required components of the interface of the web application and a use case diagram explaining how a user would interact with the application.

3.1. Study area and data

The data required for this research is a well formed CityGML file. The CityGML file (Potsdam.gml) of area Potsdam, Germany has been considered for this research. The file has been provided by Technische Universität München, Germany along with the installation set-up of 3DCityDB. It comprises, in total, 97 3D building objects of LoD1 specification. The complete detail of the dataset has been provided in Table 3-1:

Table 3-1: Details of the data set used

Name	Potsdam.gml
Region	Potsdam, Germany
File Type	CityGML
Level of Detail (LoD)	1
Projection	Universal Transverse Mercator (UTM)
Zone	33 Northern Hemisphere
Datum	European Terrestrial Reference System 1989 (ETRS89)
Spheroid	GRS 1980
Units	Meters
EPSG Code	25833
Total buildings	97

3.2. Tools used

3.2.1. Hardware Tools

The hardware used for this research has been listed in Table 3-2:

Table 3-2: Details of the hardware used

Processor	Intel Core i5-2450M 2.40GHz
RAM	4 GB
Graphics Card	AMD Radeon HD 7650M GPU
Video RAM	2 GB

3.2.2. Software Tools

The software/packages used for this research have been listed in Table 3-3:

Table 3-3: Details of the software used

S. N.	Software/Packages	Purpose
1	PostGIS version 2.0	To be used as 3D city database to store 3D city objects
2	3DCityDB Importer/Exporter version 1.6	Import of CityGML files to the 3D city database and export as KML/KMZ
3	Apache HTTP Server 2.2	Set up a local web server for client-server communication
4	Google Earth	To test the generated KML/KMZ files
5	Google Chrome v32.0.1700.76 m	To test the web application on web browser
6	Mozilla Firefox v26	To test the web application on web browser
7	Opera 19.0	To test the web application on web browser
8	Internet Explorer 11.0	To test the web application on web browser
9	Notepad++ v6.5.1	To develop and manage code base
10	Enterprise Architect	To create UML diagrams

3.3. Usability

The intention of this research is to develop a cross browser web based application allowing the user to perform the research objectives. As per the objectives, the application should allow the user to visualize the geometry and semantics of 3D city objects from a source CityGML file. Further, to perform 3D analysis, the research focuses on 3D buffer analysis. As mentioned in section 2.3.4, the previous research studies demonstrate the use of 3D buffer zones for visualizing a bomb threat scenario. Likewise, this research also focuses on creation of such 3D buffer zones in the form of spheres. But unlike previous studies, this study intends to create the buffer zones on-the-fly on the client end. With the help of front-end programming using JavaScript and the powerful combination of HTML5 and WebGL, the focus is on developing such computational intensive analysis on the front-end. It also aims to perform 3D operations such as 3D intersection and 3D inside to determine the 3D building objects which are either completely inside or partially intersecting the buffer zone. This implementation is very helpful for field workers and decision makers to understand and be well prepared for evacuation planning or other emergency scenarios in an urban area.

Further, to develop a robust and clean application, it is very important to gather the requirements of the application, keeping better user experience principles in mind. It is necessary to design a clear and easy-to-use interface to provide better usability to the user. Considering all the requirements of the research, the user interface of the application must meet following criteria:

- ✓ The interface must include Cesium virtual globe, in 3D mode, rendered within HTML5 canvas element.
- ✓ The globe's interface must provide the functionalities of pan, rotation, zoom, tilt and fly for a better scene exploration.
- ✓ The application should be able to visualize 3D contents from the source files and provide real-time dynamism to it.

- ✓ The application must provide the functionalities to perform analysis on 3D objects by the user as per the research objectives.
- ✓ The interface should include buttons/tools to provide assistance to the user.
- ✓ The web application should be as light as possible.

3.3.1. Use case diagram

To further explain the user's interaction with the web application, the use case model has been developed. The use case model is the convenient way to describe the user's interaction with the system with the help of involved actors and use cases (Pender, 2002).

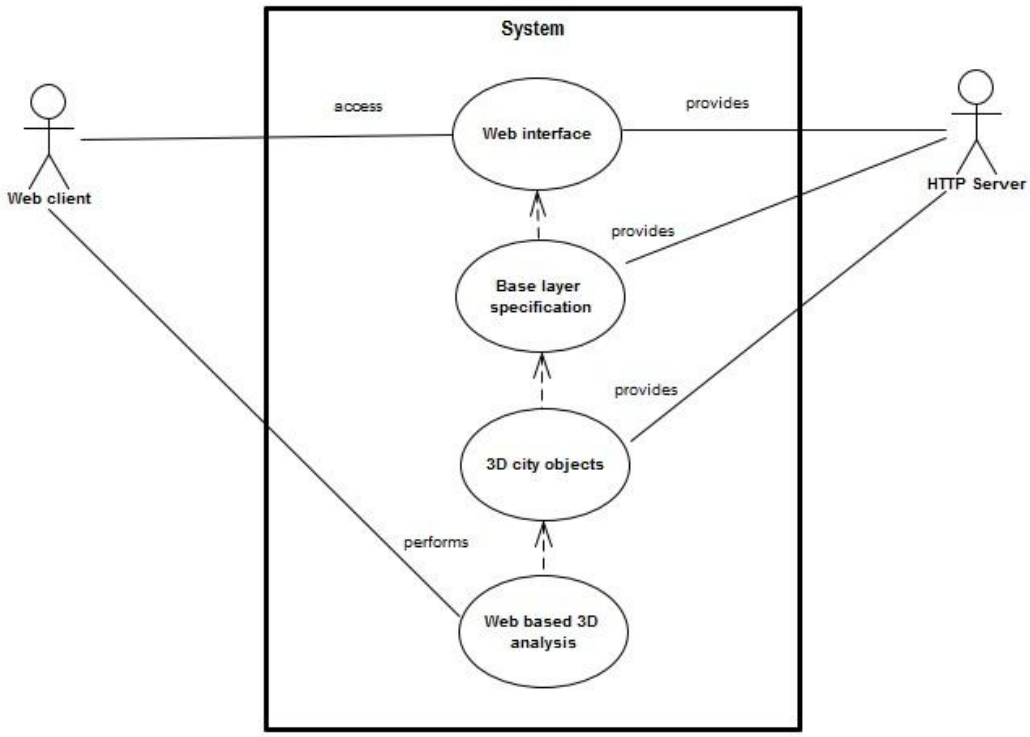


Figure 3-1: Use case diagram

As shown in Figure 3-1, the use case model consists of two actors and four use cases. The explanation of actors has been provided as below:

- Web client – Web client is the end-user to use the web application and perform the desired operations. The client is responsible to interact with the web interface of the application, visualize the 3D contents of CityGML file and further perform analysis as per the requirements.
- HTTP Server – This actor is responsible to provide actual data to the web client in the form of HTML file, consisting CSS and the JavaScript files, for visualizing Cesium virtual globe on the client side. Further, it is responsible for sending across the data for 3D city objects from CityGML. During the research, Apache HTTP local server has been used for developing the application. However, in future, the scope can be broadened by using

a remote server architecture in order to develop a complete web application to be used on internet.

The details for use cases have been explained as below:

- Web interface – The client would access the web interface by sending a request to the HTTP server. In turn, the server would respond with a single HTML file, comprises of corresponding CSS and JavaScript for Cesium API. As a result, the client would be able to visualize the virtual globe, in 3D mode, rendered perfectly within HTML5 canvas element. The interface also contains the buttons/tools, created using JavaScript, to assist the end-user to perform specific actions.
- Base layer specification – The Cesium API contains layer imageries from different sources such as Bing Maps, ESRI Maps and OpenStreetMap. These layers can be obtained from the Cesium API JavaScript on HTTP server and can be embedded in the Web interface. With its help, the end-user can simply select the layer imagery as per the requirement. In future, the facility of using external WMS and TMS can be added to the application.
- 3D city objects – 3D city objects are stored under a processed file, which is retrieved from CityGML. This file is stored on HTTP server and the objects can be parsed by the JavaScript of the web application. As part of requirement, the geometry and semantics of these city objects can be embedded to web interface for them to be visualized by the end-user.
- Web based 3D analysis – As per the requirement, the end-user is capable of performing 3D analysis, in particular, buffer zone analysis on top of 3D city objects within web interface. The end-user can create buffer zones and perform analysis directly on the client, without interaction with the data stored on the server.

4. DESIGN AND IMPLEMENTATION

This chapter describes the high level architecture of the application, followed by detailed explanation of implementation of each component.

4.1. High Level Architecture

This section describes the high level architecture diagram of the application for the reader to have a better understanding of different components and their interaction with each other.

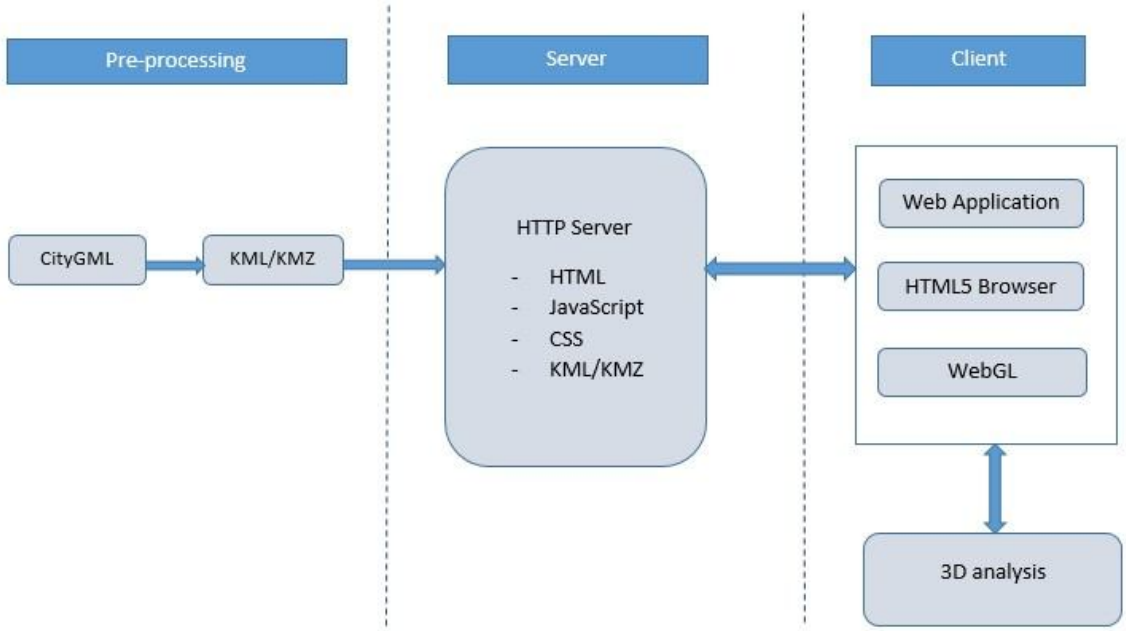


Figure 4-1: High level architecture of the application

As shown in Figure 4-1, the architecture comprises of three main blocks:

- Pre-processing – It involves the conversion of CityGML files to KML/KMZ files using the tool 3DCityDB Importer Exporter. The reason for such conversion is that CityGML files are usually written in local spatial reference systems and are not suitable to be directly visualized on virtual globes. Virtual globes support global coordinates such as WGS84.
- Server – The server includes running Apache HTTP server, comprising a single HTML file and files related to Cesium API. The HTML file references CSS and JavaScript for Cesium API. It enables the user to visualize and interact with the Cesium virtual globe as part of the web interface. The server also includes the KML/KMZ file obtained from CityGML. This file is parsed by the JavaScript for the end-user to visualize and interact with 3D city objects.
- Client – The client side involves the web application, which includes the interface for the user to visualize and interact dynamically with 3D city objects. The interface also includes functionality to create 3D buffer zones and analyse the objects with the buffer. The web application runs on an HTML5 enabled browser, which is WebGL compatible.

The design and implementation for each of the blocks have been explained in subsequent sections:

4.2. Pre-processing

As per research requirements, the input should be a well formed CityGML file and the objective is to visualize the contents of CityGML on the WebGL based Cesium virtual globe. CityGML files are usually written in a local spatial reference system. Due to the same reason, the input file Potsdam.gml, being used in this research, is written in local datum ETRS89 (EPSG: 25833). As this is a local reference system for Europe zone, it is not suitable to be visualized on global scale on a virtual globe. In order to achieve the same, the pre-processing of the data is mandatory, which projects the local reference system to a global reference system such as WGS84.

The tool 3DCityDB Importer/Exporter v1.6 has been used for the same purpose. It is an open source tool, which allows to import CityGML file to a 3D city database and export the contents from the database in the form of KML/KMZ. The functionality can be described with the help of Figure 4-2:

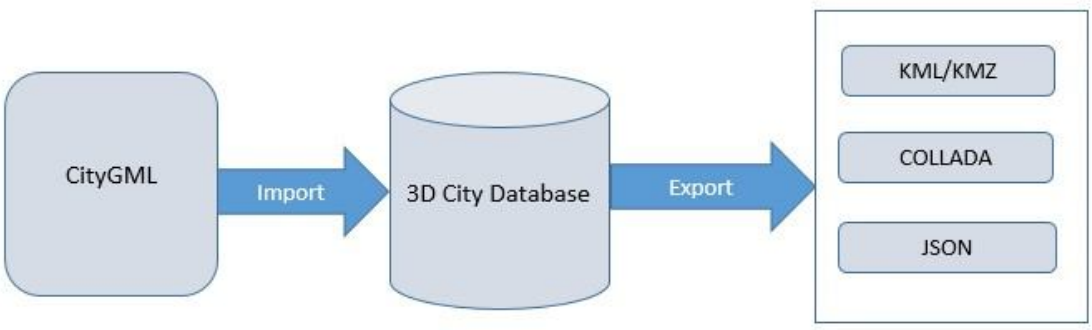


Figure 4-2: Conversion process of CityGML to KML

As shown, the tool allows us to import CityGML file Potsdam.gml to the 3D city database. It provides us the options to import contents for specific object id, bounding box or even the city objects such as building, water body, land use, vegetation and transportation. In this research, as the bounding box (envelope) is already defined in Potsdam.gml, the complete file has been imported to the database.

The 3D city database is an open source 3D geo database, which provides the functionality to store, manage and represent virtual 3D city models on top of a standard relational database. PostGIS v2.0 has been used as a spatial relational database in this research. The reason is, PostGIS v2.0 is capable of handling and storing 3D data types. The installation of 3D city database on top of PostGIS v2.0 provides the tables to store each class object of a city model such as building, water body, land use, vegetation and transportation. By importing the CityGML to the 3D city database, all the city objects are stored in their respective tables. Since the file Potsdam.gml contains only 3D building objects, they are imported successfully to building table in 3D city database.

Once the data is successfully imported to the 3D city database, the data is ready to be exported. The tool provides us an option of 'KML/COLLADA Export', with the help of which, the city

objects from the database can be exported as KML files. The tool also provides us the option to specify number of tiles and their length. The information of tiles and corresponding city objects can be exported in the form of an external JSON file. The 3D model inside virtual city model can also be exported as separate COLLADA file and can be embedded in KML file, which is out of scope of this research.

If there exists tiles and COLLADA models, each of the tile and corresponding objects are stored in a KMZ file. Further, the reference of the KMZ files, in accordance with the tiles, is given in a master KML file using KMLNetworkLink (KML Tutorial, 2013). In other words, a single KML file is created, which contains the references of KMZ files for each of the tile and the corresponding detail for each tile is stored in the KMZ file. If there are no tiles and COLLADA models, there will be only one KML file containing geometry and semantic information of the CityGML file. The geometry contains x, y and z-coordinates for each of the building vertex and they are stored as polygon under a linear ring. With the help of z-coordinate value (height), the buildings appear as extruded. The semantic information is stored as CDATA (W3C Recommendation, 2013) inside the KML. The semantic information includes name of the building, building ID, height value, address and envelope.

4.3. Server side implementation

The Apache HTTP server has been used in this research, which acts as a local server and is responsible for providing the data to the client side. For server side implementation, the necessary files have been stored as shown by hierarchy diagram in Figure 4-3:

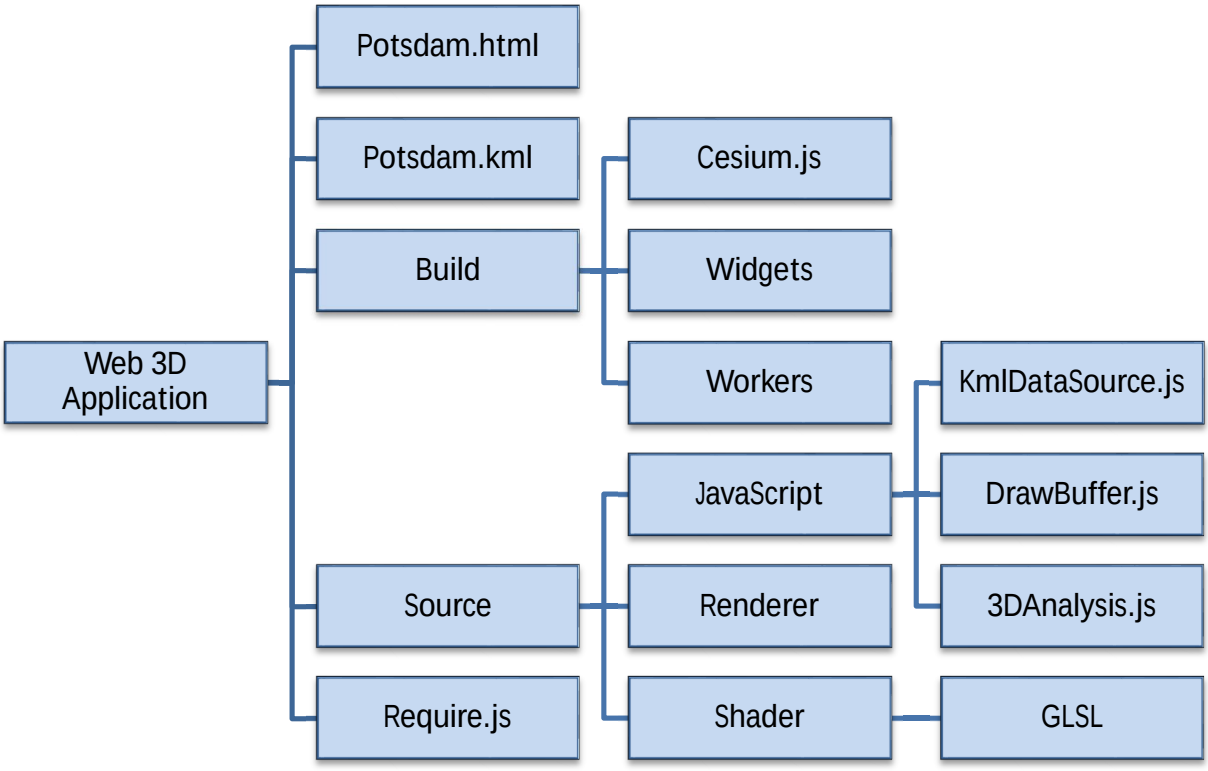


Figure 4-3: Application's structure diagram

The files stored on the server are:

(i) **Potsdam.html:** This is the main HTML file, which is responsible for visualizing the contents on the web browser. In order to use the web application, the client requests for this file and it performs the functioning and sends back results to the client. This HTML file is solely responsible for communication between the client and the server.

(ii) **Potsdam.kml:** This is the KML file derived from Potsdam.gml. This KML file is parsed by a user-defined JavaScript within Potsdam.html and is responsible for visualizing the geometry and semantics of city objects on the virtual globe on the client.

(iii) **Build:** This folder contains the files related to Cesium API. Cesium version b24 has been used for this research. The files are:
- Cesium.js: This is the JavaScript file, which is responsible for creation of 3D virtual globe and related activities. This file is referred by Potsdam.html and performs the overall activity by calling other JavaScript and CSS files as part of Cesium API.
- Widgets: This folder contains related CSS files for styling and structuring of the contents on the web browser. It positions virtual globe within HTML5 canvas element and also contains different layer imageries from Bing Map, ESRI Maps and OpenStreetMap.
- Worker: This folder contains the JavaScript files for web workers for different operations of Cesium virtual globe. Web workers (W3C Recommendation, 2012), is a new feature introduced with HTML5. It allows the scripts to run in background along with the processing at main page. This functionality improves the processing speed at client side.

(iv) **Source:** This folder contains the JavaScript files to develop required functionalities.
- JavaScript: This folder contains in-built JavaScript files, which are part of Cesium API. The files are responsible for creation of geometries, performing mathematical operations and even loading different layer imageries on the globe. The worth mentioning are three JavaScript files, which have been developed during this research.
  - KmlDataSource.js: This file is not officially part of Cesium release yet and is still in development phase. It has been used and modified in the research to parse Potsdam.kml for visualization of 3D buildings on the Cesium globe.
  - DrawBuffer.js: This file has been developed to create dynamic 3D buffer zones as part of the research objectives.
  - 3DAnalysis.js: This JavaScript contains the algorithm to determine 3D building primitives, which either intersect or are completely inside the buffer zone.
- Renderer: This folder includes JavaScript files, which directly utilize WebGL without including much details of low level WebGL API. It includes scripts for mainly buffers, textures, vertex arrays and cube maps.
- Shader: This folder includes shader programs and built-in GLSL (OpenGL Shading Language) uniforms. GLSL (GLSL Specification, 2014) is a high level language, using the syntax of C/C++, which allows to write shader programs executable on GPU.

(v) **Require.js**: This is an external JavaScript file, which provides us the functionality to develop scripts in a modular way. Modular scripts help in managing large application into smaller chunks of code. Furthermore, Require.js helps in the management of dependencies between various module, and thus, improves the overall execution speed and quality of the code (RequireJS, 2013).

4.4. Client side implementation

This section explains about the implementation on the client, which includes detailed explanation of the web based application and its working which has been developed using JavaScript. It also includes some explanation of the web browsers and low level rendering WebGL API.

4.4.1. Web browser

The first part of the client side implementation is the selection of web browser to use the developed web application. As per the primary requirements of this research, the web browser must be HTML5 enabled in order to utilize the canvas element for visualizing 3D contents. Further, the web browser must support WebGL specification in order to develop the cross-browser, cross-platform application, which can be run without any need to install a plugin. Also, the WebGL specification would provide hardware acceleration to the application while working with 3D objects. Considering the mentioned requirements, a few web browsers have been selected in order to test the results. Although comparison of web browsers is not a part of scope of this research, it has been done to test the results on all possible browsers to develop a cross browser application. The mentioned browsers are:

- Mozilla Firefox: Mozilla Firefox v26 has been used for this research. This browser has been used primarily in the research during complete development of the application. The add-on 'Firebug' is the most exciting part of the browser, which helps in detecting errors related to JavaScript and HTML in the best possible structured manner and eases the development process of the application.
- Google Chrome: Google Chrome v32.0.1700.76 m has been used to test the results. The browser is very lightweight as compared to its counterparts and thus, improves the user interaction with the application.
- Internet Explorer 11: The latest version of Internet Explorer (version 11) includes HTML5 and WebGL capabilities. As Internet Explorer still dominates the market capture in terms of number of users, it has been decided to test the results on this browser in order to reach to the maximum number of users.
- Opera: Opera v19 has also been used in this research to test the results. Opera is also one of the most successful lightweight browser and supports WebGL completely.

4.4.2. WebGL

WebGL API plays a very important role in this research to use the application without installing any third-party plugin. Also, it provides hardware acceleration, which helps in improving the performance of the application while working with 3D contents. Cesium creates the virtual globe on the web utilizing WebGL. In this section, the focus is on understanding how Cesium utilizes WebGL for the developed web application.

As mentioned in Cesium Architecture (Figure 2-7), the two components of the architecture, Renderer and Scene, are responsible for utilizing WebGL.

- Renderer – As mentioned in section 4.3, renderer helps the user to develop code in an abstracted way such that the code does not include complex low level WebGL API. This layer mainly includes following elements written in easy to understand JavaScript
    - Built-in GLSL uniform and functions
    - Abstractions for shader programs
    - Textures and cube maps

- Buffers and vertex arrays
- Render states
- Frame buffers

Although the renderer is not directly included in the application code, it is utilized by the higher level constructs such as scene or dynamic scene.

- Scene – Scene is responsible for providing a higher level construct, such as 3D globe or map, layer imageries from different sources or even custom geometries. The application development includes defining the scene, which contains several 3D geometries. The 3D geometries are rendered the help of renderer component utilizing low level WebGL API.

In that way, Cesium API helps in the development of web application utilizing WebGL and without writing the code in a low level language.

4.4.3. Web based application

The web based application is the main component of the research, and has been developed to be run on top of an HTML5 browser utilizing WebGL. This is an interface for the end-user to perform the functions as part of the research objective. Considering the research objectives, the processing of application can be described with the help of following process flow diagram in Figure 4-4:

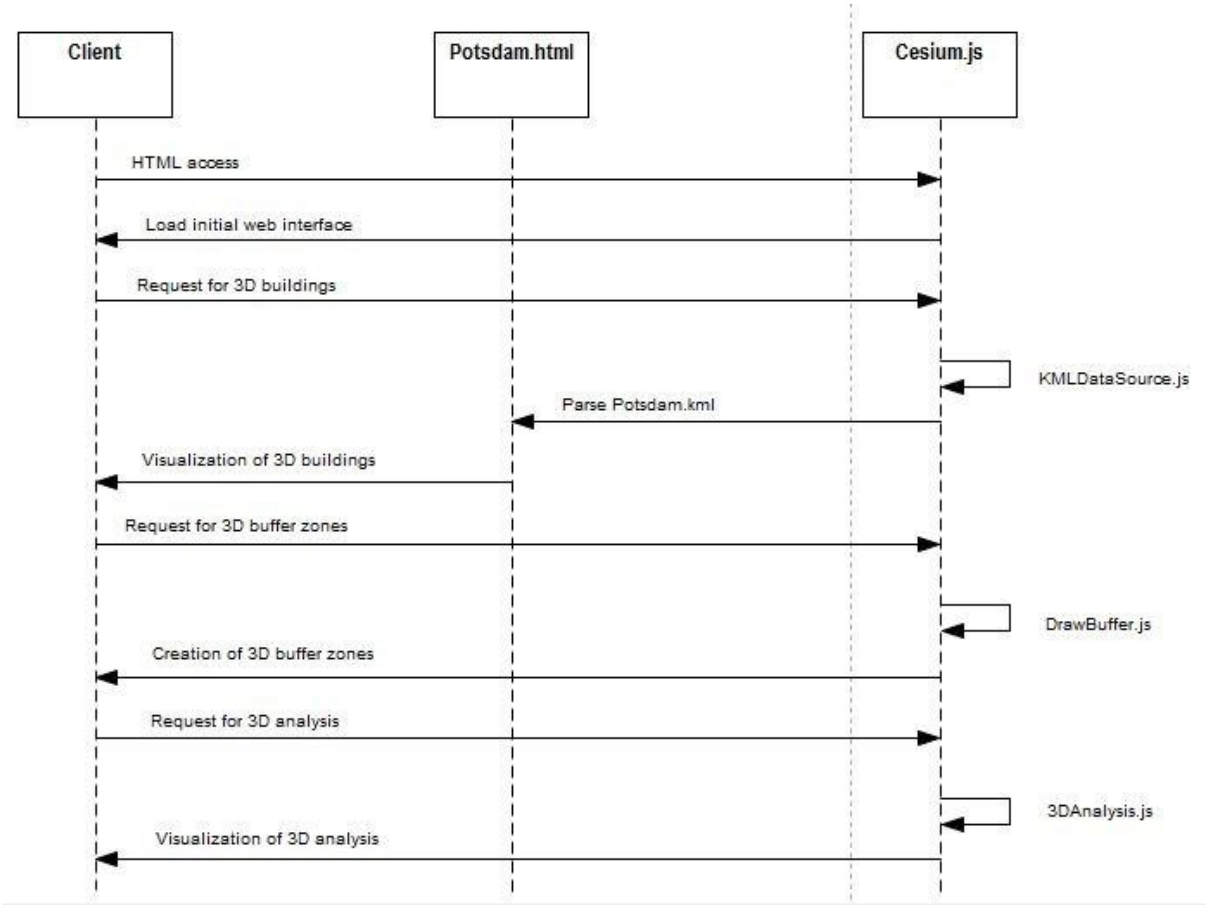


Figure 4-4: Application's process flow diagram

The processing can be explained in detail in following subsections:

4.4.3.1 Initial web interface

The following steps take place to load initial web interface

- The client sends a request to the server for Potsdam.html
- Potsdam.html calls Cesium.js, which accesses related JavaScript and CSS files. These files are responsible for loading initial virtual globe on the web browser, positioned perfectly within HTML5 canvas element. The user can properly interact with the globe by zoom, pan, tilt and fly functions. These files also provide the buttons to geo-locate and load layer imageries from different sources.
- With the help of Require.js, the separate modules have been added to the interface, which help users to perform the required actions, such as, to visualize geometry and semantics of the city model and perform 3D analysis.
- Hence, by requesting the Potsdam.html, the user can visualize the initial interface of application on the web browser.

4.4.3.2 KML parsing

This is the next step of the application to load the contents of the city model. It involves following steps:

- Potsdam.html accesses KMLDataSource.js with the help of Cesium.js.
- This JavaScript parses the contents of Potsdam.kml in order to visualize them on Cesium virtual globe.
- The parser first identifies and stores the bounding box of the city area from the KML.
- The parser then identifies the vertex points (x, y and z-coordinates) of each building object in the KML.
- It creates polygon geometry for each building using x and y coordinates and use z coordinate as height parameter. It allows 3D visualization of the building on the virtual globe. Each polygon geometry can then be made part of the primitive, which allows each building object to be treated as an individual object.
- Additionally, the semantic information of city model is stored in KML as CDATA. This information is stored with corresponding building object's geometry.
- The functionality has been developed in such a way that the camera first flies to the bounding box of the city fetched from the KML, and visualizes the 3D geometry of the objects. Each building can be picked as an individual object, thus, the individual building can be highlighted on mouse over and the user can retrieve its semantic details on mouse click.

4.4.3.3 Dynamic buffer zone creation

This functionality has been developed for the user to create on-the-fly 3D buffer zones. It involves following steps:

- With the help of DrawBuffer.js called by Potsdam.html, the prompt appears for the user to provide input value for the radius of buffer, which can be stored in a variable.
- Using JavaScript event handler, the mouse click retrieves the longitude and latitude for the desired location and, stored in a variable.

- Using longitude and latitude as centre and radius provided by the user, the buffer zone can be created as hemisphere dynamically. The hemisphere is created as a sphere geometry and stored as a separate primitive.

4.4.3.4 3D analysis

For further analysis, the functionality has been developed to perform 3D operations by focusing on 3D intersection and 3D inside. The functionality allows the user to check how many buildings are completely inside or partially intersecting the buffer zone. This is useful mainly for identifying buildings under threat and starting evacuation plans for them. The algorithm developed for this purpose is inspired from (Ericson, 2005). The objects in consideration are hemisphere and 3D polygon. Accordingly, the intersection or collision algorithm has been developed, which contains following steps:

1. Create axis-aligned bounding box (AABB), as mentioned in section 2.4.3, for the 3D building objects.
2. Determine the center C and radius R of the hemisphere.
3. Find a closest point P on the AABB from the center C of the hemisphere.
4. If the squared distance of PC is less than the squared distance of R, then AABB of the 3D building either intersect or is completely inside the sphere.
5. If this is the case, highlight the building to different colour for the user to identify them.

Explaining the algorithm in more detail, the AABB of the building contains the minimum, maximum and centre points of the bounding box that closely fits the building object. In order to find the closest point on AABB from centre of the hemisphere, first the distance of the centre of hemisphere and centre of AABB is computed and the iteration is performed on this distance with respect to x, y and z- axes of the AABB, until the closest point is found. Once the closest point is found, its distance is calculated from the centre of the hemisphere. If the square of this distance is less than the squared distance of the radius of the hemisphere, then AABB and the hemisphere intersect. Graphically, in 2D, these results can be shown as:

Case I: $PC^2 > R^2$ (Do not intersect)

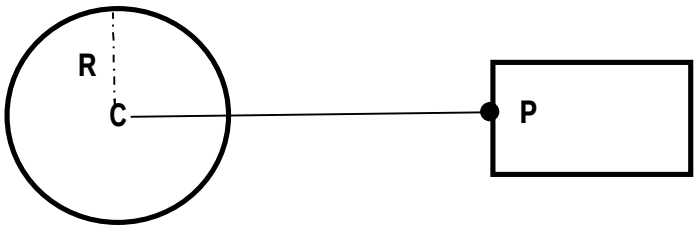


Figure 4-5: Representation of case I of sphere-AABB intersection

Case II: $PC^2 < R^2$ (Intersects)

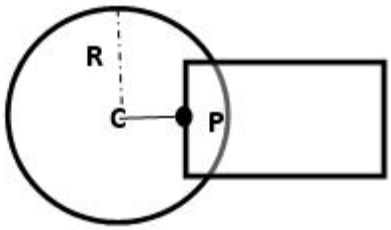


Figure 4-6: Representation of case II of sphere-AABB intersection

Case III: $PC^2 < R^2$ (Completely inside the sphere)

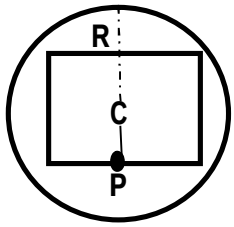


Figure 4-7: Representation of case III of sphere-AABB intersection

Hence, by demonstrating three cases, it can be shown that if the squared distance of centre of sphere and AABB (PC) is greater than squared distance of radius of the sphere, then the two objects are disjoint. Otherwise, if the same squared distance PC is less than the squared distance of radius of the sphere, then the AABB is either completely inside or partially intersects with the sphere. Moreover, it can also be found, if the squared distance PC is equal to the squared distance of radius of the sphere, then both the objects are adjacent. In this way, this algorithm determines the number of 3D buildings intersecting or completely inside the buffer zone in the web application.

5. RESULTS AND DISCUSSIONS

This chapter presents the results of the implementation of each step during the research. It also covers the discussions of highlights of each implementation step during the execution of the research.

5.1. Web based interface

The web base interface is the initial page of the application, which is accessed by the client through a call of HTML page on local server. The screenshot of the initial web interface can be displayed in Figure 5-1:

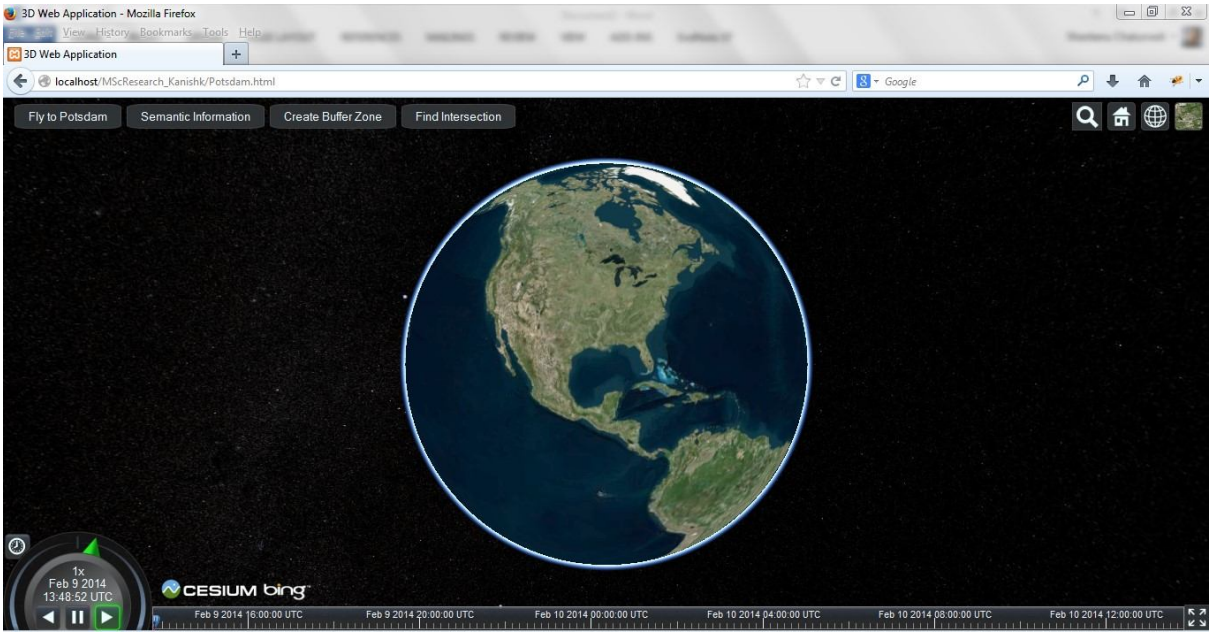


Figure 5-1: Initial web based interface

As shown, the interface includes the Cesium WebGL virtual globe, which is capable of performing the rotation, zoom, pan and fly operations. On the top right, the in-built buttons are available to geo-locate, refresh, changing the 3D/2D mode and selecting the base layer. On the top left, the buttons have been created to assist the end user to perform specific operations. The details of the buttons are:

- Fly to Potsdam: This button parses the KML file and zoom-in the camera to the bounding box of the study area.
- Semantic information: This button is responsible to provide the semantic information of the specific building.
- Create Buffer Zone: This button allows the end-user to create the 3D buffer zones dynamically at specific location.
- Find intersection: This button allows the user to determine the buildings, completely inside or intersecting the buffer zone.

5.2. Visualization of geometry

As soon as the end-user clicks on the button 'Fly to Potsdam', the application parses the Potsdam.kml with the help of KMLDataSource.js and takes the camera to the bounding box of the location. The geometry of the parsed buildings can be visualized in Figure 5-2:

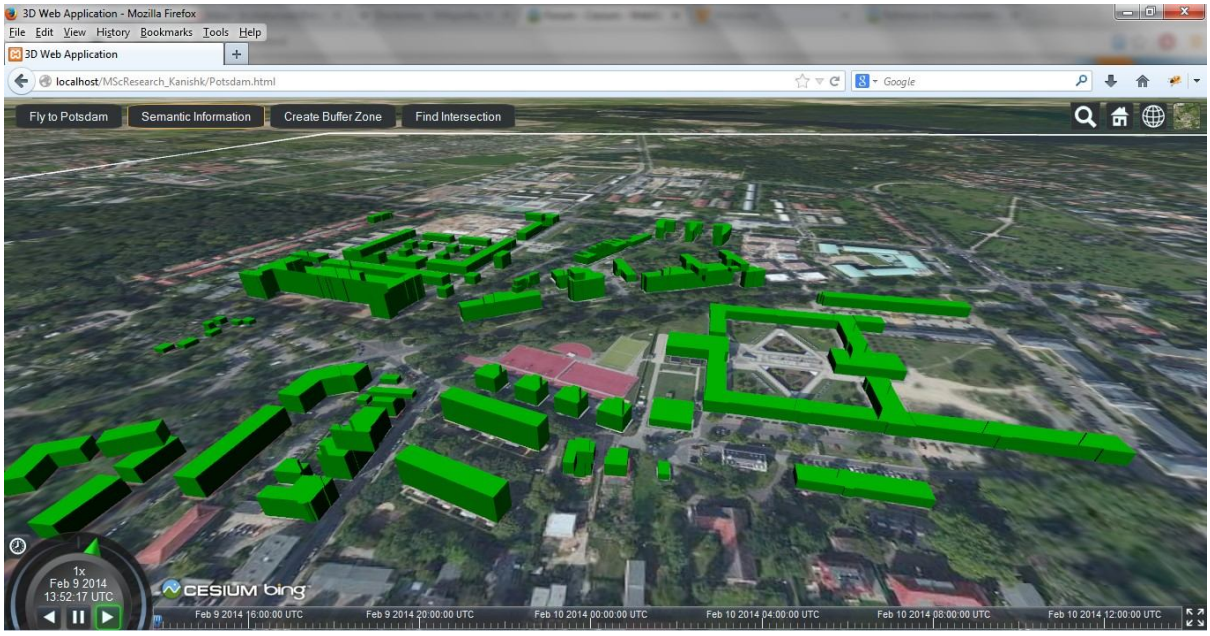


Figure 5-2: Geometry visualization. Base layer: Bing Maps Aerial

As shown, the geometry of the 3D buildings in LOD1 can be visualized in the application. The user can further zoom, pan or rotate the camera in order to visualize the buildings from different angles. In the figure, the base layer used is Bing Maps Aerial. The end-user can select different base layers, which can be shown by subsequent figures:

Figure 5-3: Geometry visualization. Base layer: Bing Maps Roads

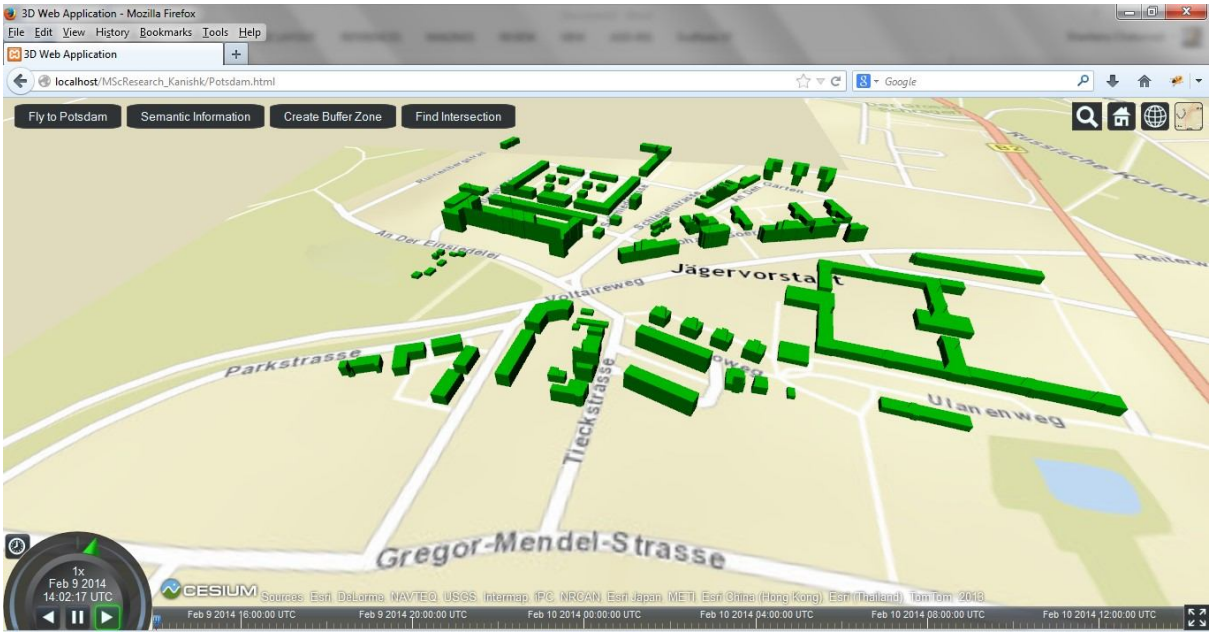


Figure 5-4: Geometry visualization. Base layer: ESRI World Street Maps

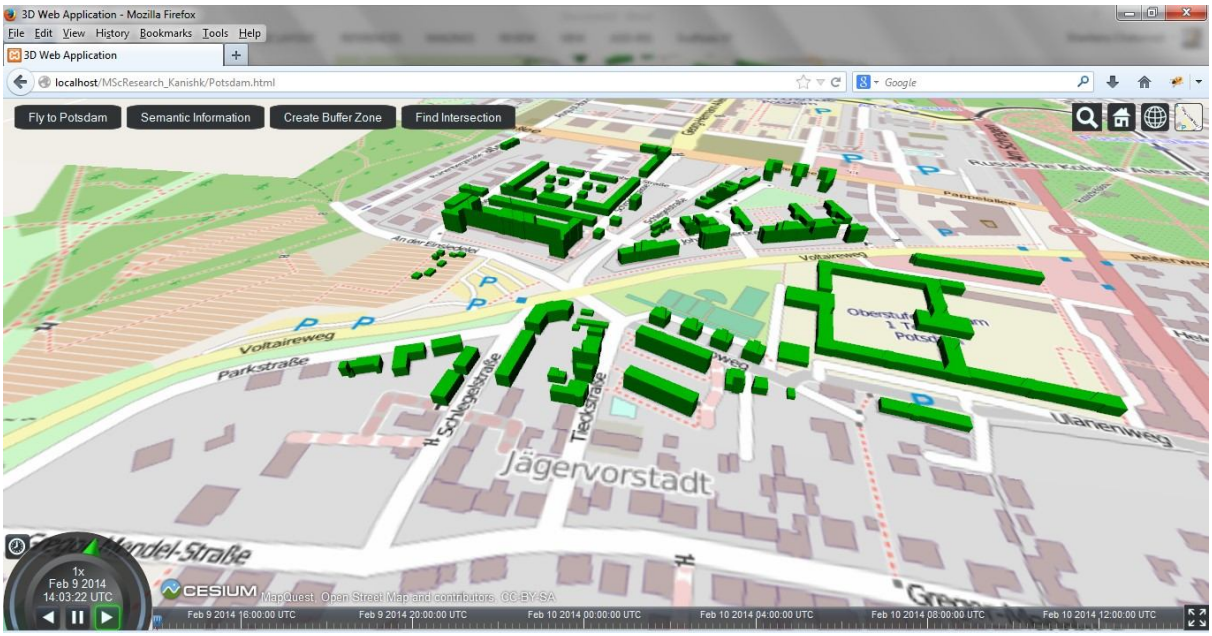


Figure 5-5: Geometry visualization. Base layer: OpenStreetMap

5.3. Visualization of semantics

With the button 'Semantic information', the user can pick any of the buildings, which get highlighted on mouse over and displays the semantic information on mouse click on the building. The functionality can be demonstrated in Figure 5-6:

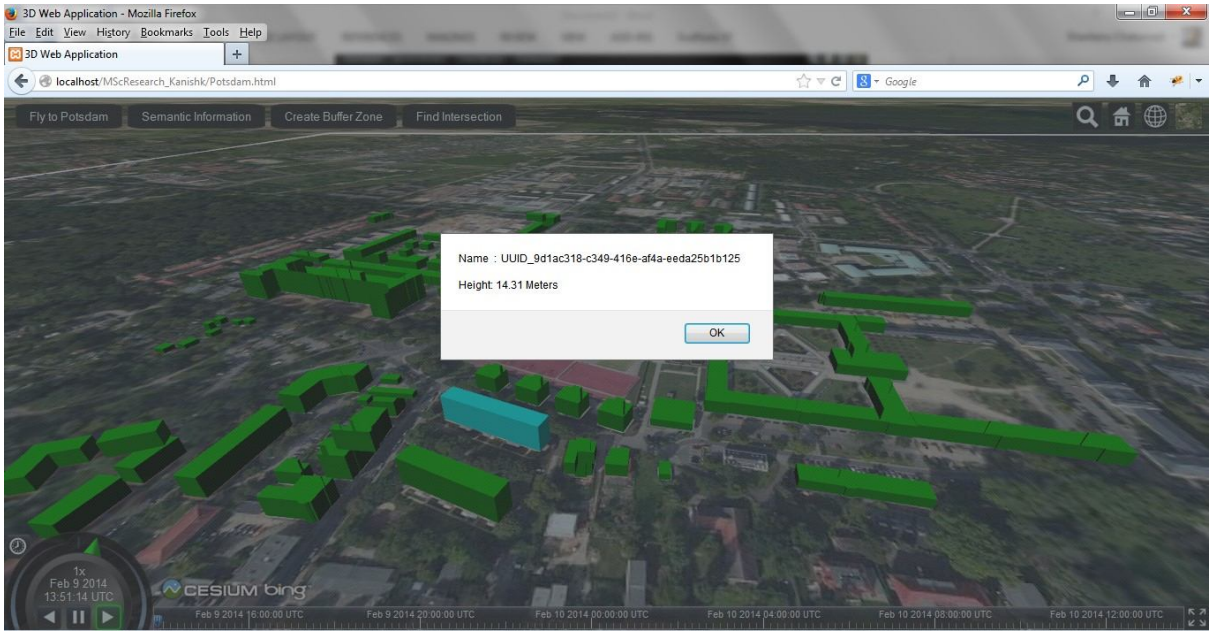


Figure 5-6: Visualization of semantic information

As shown, the current functionality visualizes the building ID and height of the building (in meters). As per specific requirement, more details can be added to be displayed. It is also possible to display the textures of the building in case of higher LODs, which is out of scope of the research.

5.4. Creation of 3D Buffer Zones

When the end-user clicks on the button 'Create Buffer Zone', the prompt appears for the user to provide the radius of the buffer as input. The prompt is shown in Figure 5-7:

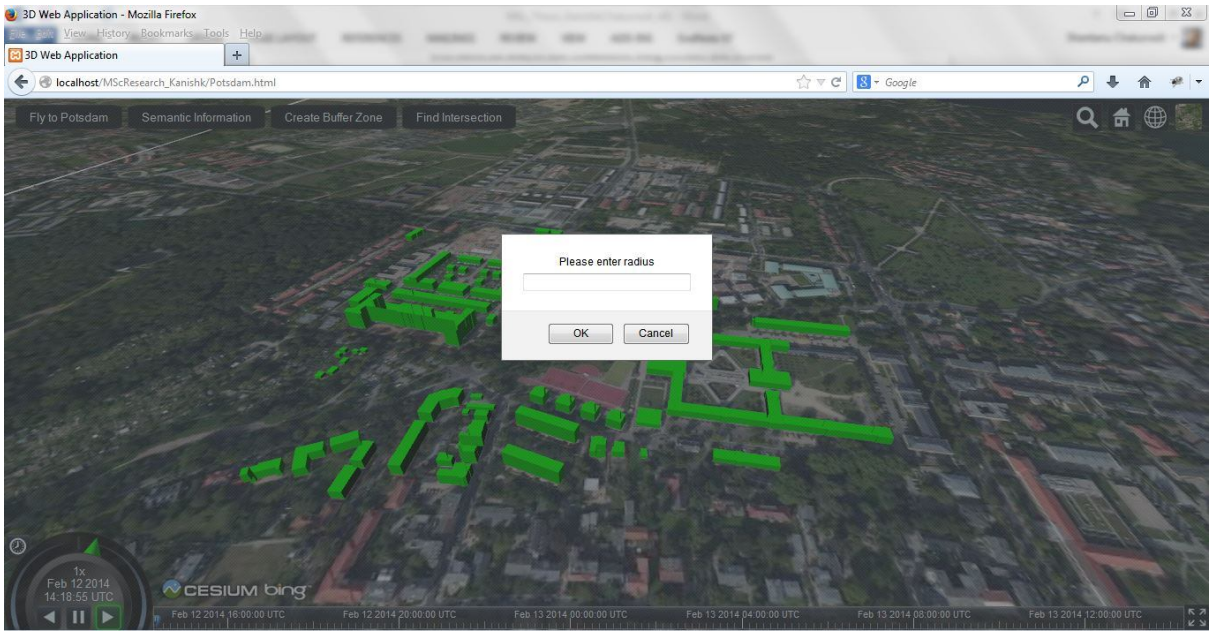


Figure 5-7: Input prompt for the radius of the buffer

Once the user provides the radius, the buffers can be created dynamically by just clicking on the required position.

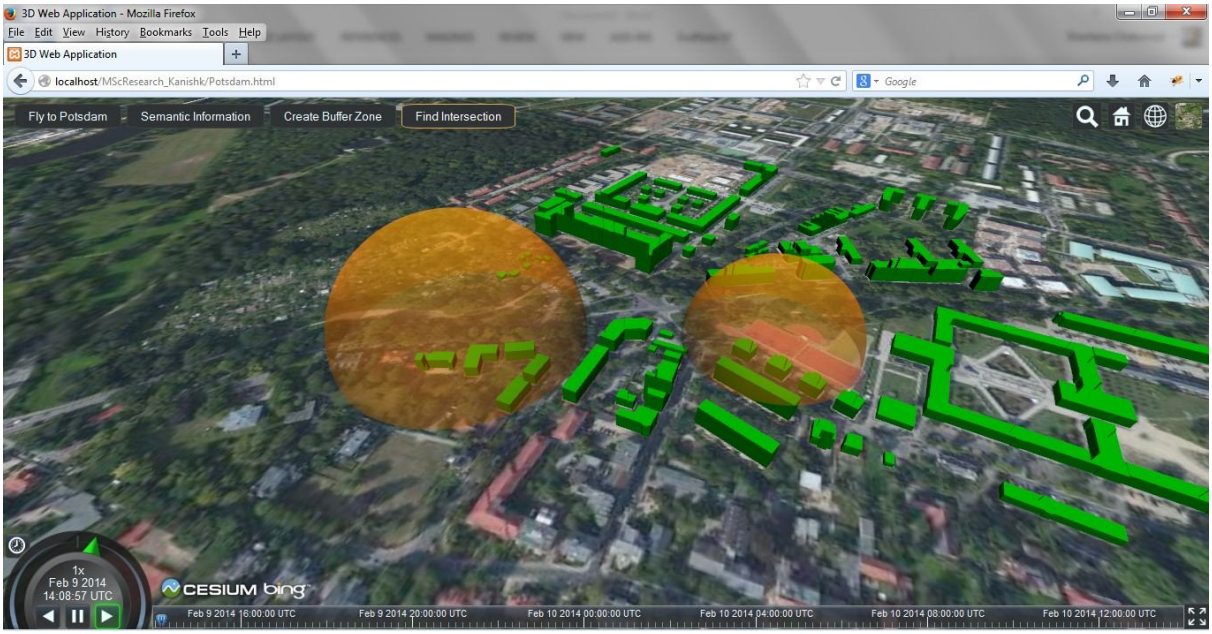


Figure 5-8: Creation of dynamic buffer zones

As shown in Figure 5-8, the buffer zones of different dimensions can be created by the user by just a single mouse click.

5.5. 3D Analysis

Further analysis can be performed by the user by clicking on the button 'Find intersection'. It determines the buildings which are either completely inside or partially intersecting the buffer zones. The result can be demonstrated in Figure 5-9:

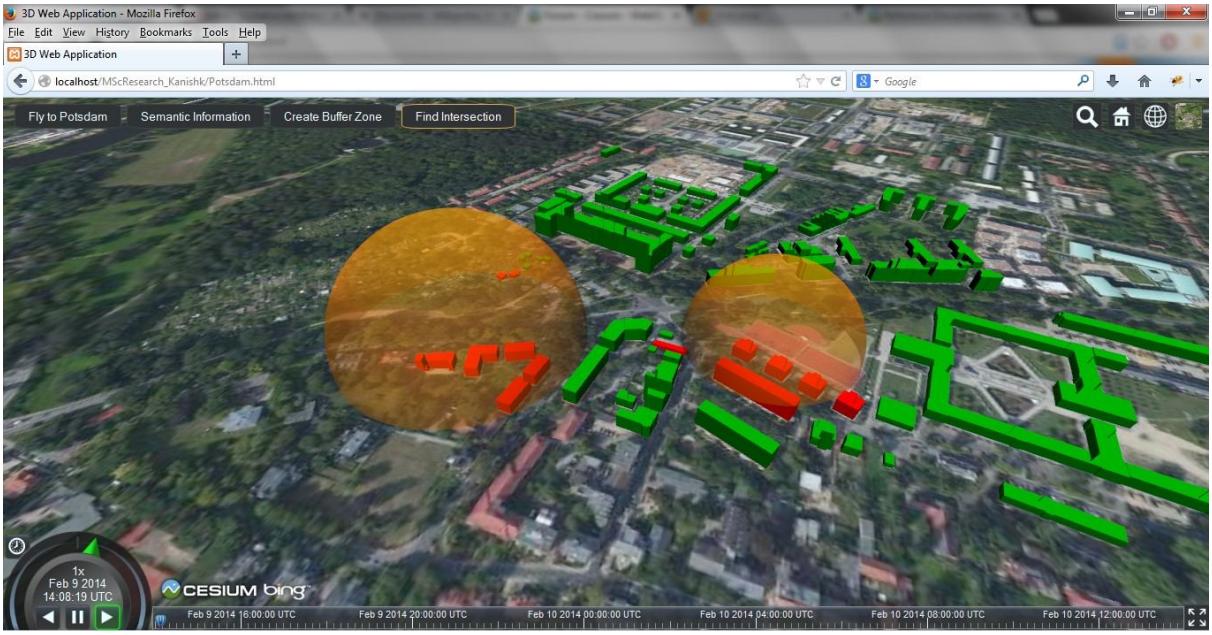


Figure 5-9: Finding geometry intersections

5.6. Tests on web browsers

The results and performance of the web application have been tested on different WebGL compatible web browsers. The web application has been developed on Mozilla Firefox and all the screenshots in previous sections have been taken from the same browser. Although the comparison of the web browsers is not the intention of this research, the results have been tested on other browsers to check the cross-browser functionality of the application. The other browsers considered are Google Chrome, Opera and Internet Explorer11.

Test 1: Google Chrome

The web application has been tested successfully on Google Chrome. All the functionalities of the application performed as per the requirement. Google Chrome, being very lightweight, provided better user experience in comparison to all other browsers. The screenshot of final result on Google Chrome has been displayed in Figure 5-10:

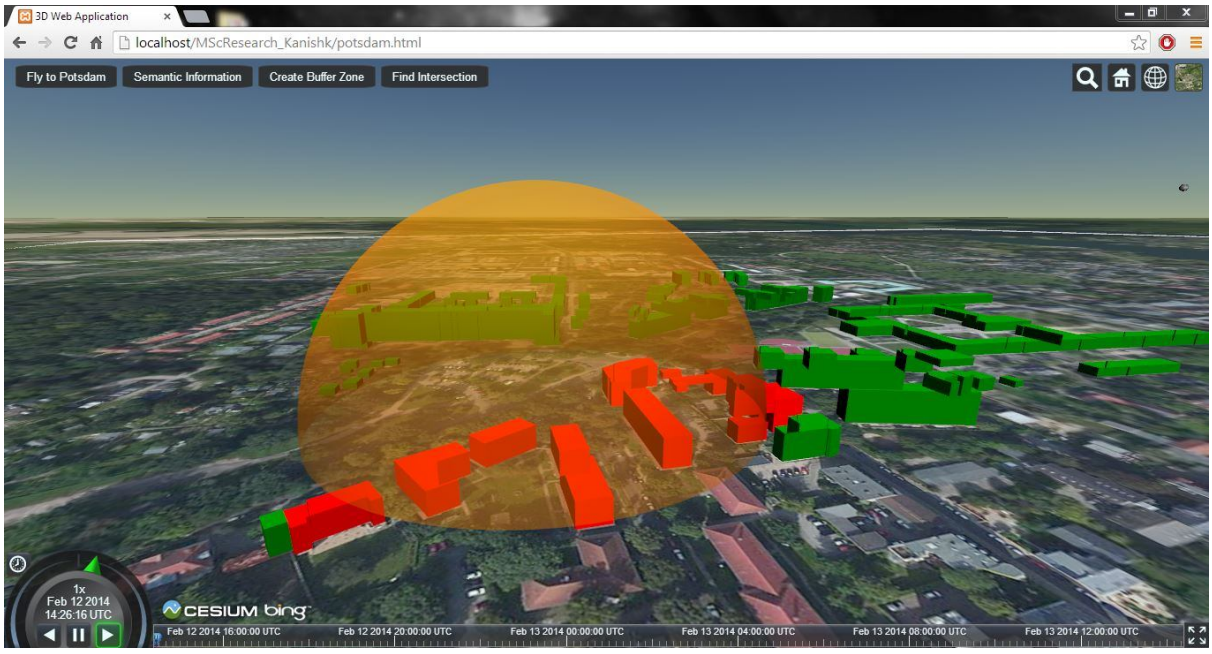


Figure 5-10: Results on Google Chrome

Test 2: Opera

The web application has also been tested successfully on Opera. Again, all the functionalities of the application performed as per the requirement. The screenshot of final result on Opera is shown in Figure 5-11:

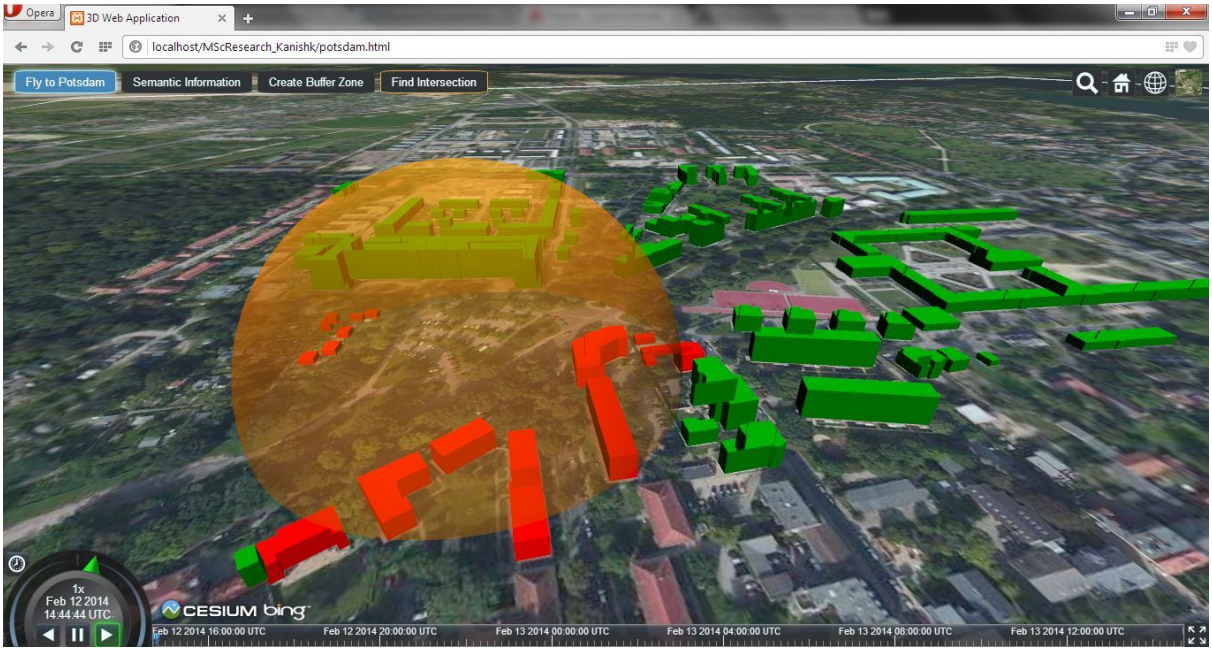


Figure 5-11: Results on Opera

Test 3: Internet Explorer 11

The web application failed to run on Internet Explorer (IE) 11. As per the specification, IE11 still partially supports WebGL. Not all the functionality of WebGL is supported by IE11. During the testing of the application, some of fragment shaders, used by Cesium, failed to load on the browser. The screenshot of the result is shown in Figure 5-12:

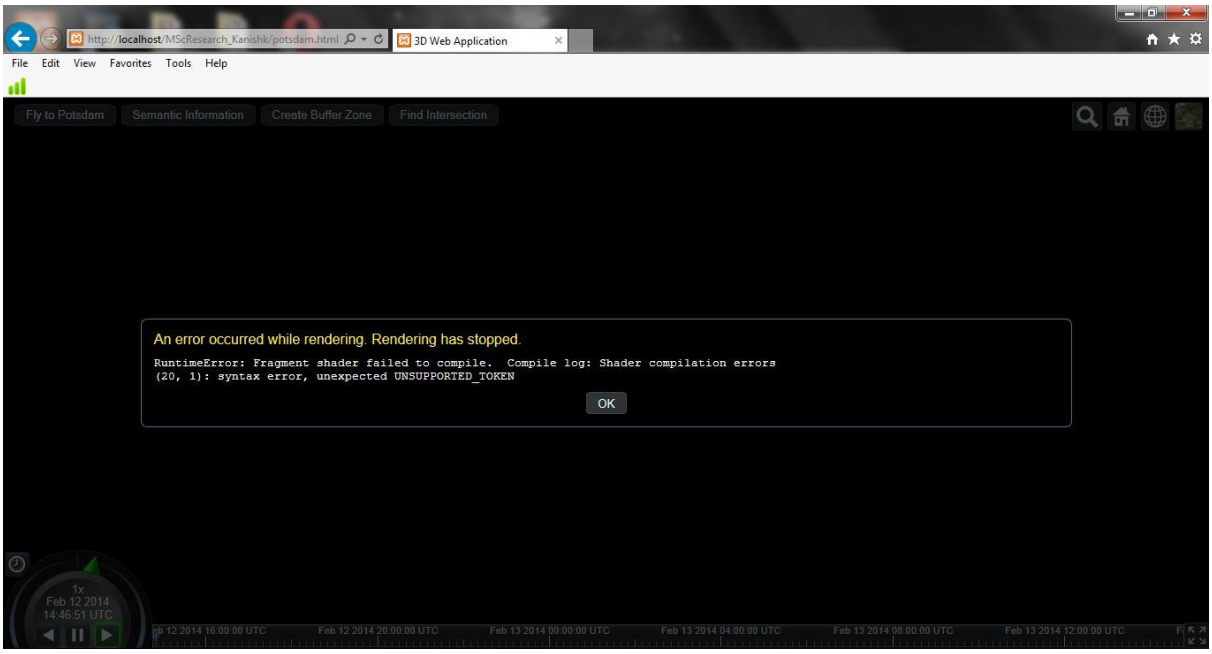


Figure 5-12: Results on Internet Explorer 11

6. CONCLUSION AND RECOMMENDATIONS

6.1. Conclusion

This research has successfully proposed and implemented the functionality to visualize as well as analyse the complex 3D city models on a web browser utilizing HTML5 and WebGL. The study of CityGML format and its visualization has been done to understand the complex 3D city models. The methodology has been developed to visualize the CityGML in the form of KML/KMZ. It enables the 3D buildings and semantics to be visualized on global scale on a virtual globe. In addition, the functionality has been developed to create dynamic 3D buffer zones by the user and further, to determine the buildings, either completely inside or partially intersecting the buffer zones. The development of such functionality can be helpful for decision makers or urban planners for a scenario such as evacuation planning.

6.1.1. Answers of research questions

1) What are the different techniques proposed in literature to visualize geometric and semantic attributes of CityGML file on a web browser efficiently?

The web based visualization of complex CityGML files is still at a very early stage and has not matured completely. During the research, several techniques were studied to visualize geometric and semantic attributes of CityGML on web browser. The three successful standards are X3D, JSON and KML, which have been used in various literature for web based visualization of CityGML. However, the purpose of their usage has been different. In this research, KML has been preferred over other two standards to visualize the CityGML contents for two reasons:

- The research focuses on visualization of CityGML on a WebGL based virtual globe. CityGML is written in local coordinates; KML supports global coordinates such as WGS84. Hence, the conversion of CityGML to KML directly allows to be visualized on a virtual globe.
- KML can be parsed easily using JavaScript in order to be visualized directly on a global scale.

2) Which is the best suitable WebGL based library to perform analysis and visualization of CityGML file on a web browser?

WebGL is a low level API to render the objects, due to which it is difficult to write programs using it. To provide ease of development, various WebGL based libraries have been developed, which abstract the low level programming. The three such libraries have been studied in the research: Three.js, Scene.js and GLGE. All the three libraries have been designed for different applications. While working with CityGML files involving large number of 3D building objects, scene.js has been found to be the most suitable library. Scene.js allows to render large number of

geometries at the same time. It allows to define each geometry uniquely, which enables them to be picked at a later stage. During the research, this library has been utilized with the help of Cesium API in order to render 3D building objects and buffer zones at the same time. Although Scene.js is more complex than three.js, but it serves the purpose in well manner as compared to latter.

3) How does the involvement of WebGL based virtual globe improve performance of computationally intensive algorithms?

In the research, Cesium has been used as a WebGL based virtual globe. The architecture of Cesium significantly improves the performance of the complex algorithm, due to following reasons:

- It allows to render large number of objects together utilizing scene.js.
- It involves web workers during the creation of geometries and performing other tasks, which allows to work in background in a thread-like environment. In case of large number of computations, it proves to be a huge improvement in performance as it allows to perform functions in parallel.
- During the intersection tests of building objects and buffer zones, it allows to create axis aligned bounding box (AABB) of the building objects. Since AABB allows to perform intersection tests on the same plane with the least cost, it improves the overall performance of the application during the analysis.

4) Which part of the functionality to be implemented on client side or server side in order to achieve best results?

The technological advancement in HTML5 and WebGL allows to develop powerful 3D web applications. Hence, the intention of the research has been to give maximum power in the hands of the client. As per the architecture, the role of the server is just to store the files; while all the functionalities such as parsing of geometries, creation of buffer zones and intersections have been performed by the client using JavaScript. The combination of HTML5 and WebGL has made it possible to perform such complex tasks on the client side in the most optimal way.

6.2. Recommendations

Considering the results obtained from the research work, following aspects need more attention for future:

1. CityGML parsing: Current research involves KML parsing for visualization of CityGML contents on a virtual globe. The recommendation is to develop the functionality to parse the CityGML directly for global scale. That will allow the user to directly visualize CityGML contents without any need of pre-processing.
2. Other 3D analyses: Utilizing the powerful combination of HTML5 and WebGL, the functionalities should be developed to perform other on-the-fly analyses, such as viewshed analysis, 3D density or spread analysis.

3. Dynamic data: Although the research focuses mainly on static objects, Cesium architecture is highly capable of performing operations with dynamic data. The dynamic data such as moving objects or crowd-sourced or sensor data can be added with such functionalities. That will allow to develop applications for working with real time data.

REFERENCES

Analytics Graphics, Inc. (2011). Cesium - WebGL Virtual Globe and Map Engine. Cesium. Retrieved May 27, 2013, from http://cesium.agi.com/

Brunt, P. (2013). GLGE WebGL Library/Framework. Retrieved November 26, 2013, from http://www.glge.org/

Christen, M., & Nebiker, S. (2011). OpenWebGlobe SDK An Open Source High-Performance Virtual Globe SDK for Open Maps. In *1st European State of the Map Conference, Vienna*. Retrieved from http://sotm-eu.org/slides/36_MartinChristen_OpenWebGlobe.pdf

Elvidge, C. D., & Tuttle, B. T. (2008). How virtual globes are revolutionizing earth observation data access and integration. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37, 137–139.

Ericson, C. (2005). Bounding Volumes. In *Real-time collision detection* (First., Vol. 1, pp. 77–174). United States of America: Elsevier Inc.

Federico Prandi, R. D. A. (2013). Using CityGML to deploy Smart-City services for Urban Ecosystems. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XL-4/W1, 87–92.

Gesquière, G., & Manin, A. (2012). 3D Visualization of Urban Data Based on CityGML with WebGL. *International Journal of 3-D Information Modeling (IJ3DIM)*, 1(3), 1–15.

GLSL Specification. (2014). OpenGL Shading Language. Retrieved March 5, 2014, from https://www.opengl.org/documentation/glsl/

Google. (2013). MapsGL - Google Maps. Retrieved May 27, 2013, from http://support.google.com/maps/bin/answer.py?hl=en&answer=1630790

Gröger, G., Kolbe, T. H., Nagel, C., & Häfele, K.-H. (2012). OGC City Geography Markup Language (CityGML) En-coding Standard. Open Geospatial Consortium Inc, OGC. Retrieved from http://www.opengeospatial.org/standards/citygml

Hickson, I., & Hyatt, D. (2010). HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft, 19.

Iglesias, D. G. (2012, March 5). Design and implementation of 3D buildings integration for a Webgl-Based Virtual Globe: a case study of Valencian Cadastre and Fide Building Mode. Retrieved from http://run.unl.pt/handle/10362/8327

Kay, L. (2013). SceneJS V3. Retrieved November 26, 2013, from http://scenejs.org/

Klokan Technologies. (2011). WebGL Earth - open source 3D digital globe written in JavaScript. Retrieved May 27, 2013, from http://www.webglearth.org/

KML Tutorial. (2013). KML Tutorial - Keyhole Markup Language — Google Developers. Retrieved February 13, 2014, from https://developers.google.com/kml/documentation/kml_tut#network_links

Kolbe, T. H., König, G., Claus, N., & Stadler, A. (2013). 3D City Database for CityGML Version 2.1.0. Institute for Geodesy and Geoinformation Science, Technische Universität Berlin. Retrieved from http://www.3dcitydb.net/3dcitydb/fileadmin/downloaddata/3DCityDB-Documentation-Addendum-v2_1.pdf

Koussa, C., & Koehl, M. (2010). Implementation of a 3D GIS in Internet Environment. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XXXVIII-4/W15, 129–134.

Mao, B. (2011). Visualisation and Generalisation of 3D City Models. Royal Institute of Technology (KTH), Sweden. Retrieved from http://kth.diva-portal.org/smash/record.jsf?pid=diva2:456906

Marrin, C. (2011). Webgl specification. Khronos WebGL Working Group.

Moser, J., Albrecht, F., & Kosar, B. (2010). Beyond Visualization–3D GIS Analyses for Virtual City Models. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XXXVIII-4/W15, 143–146.

Mrdoob. (2013). Home · mrdoob/three.js Wiki. Retrieved November 26, 2013, from https://github.com/mrdoob/three.js/wiki

OGC. (2012). OGC 3D Portrayal Interoperability Experiment (p. 76). Retrieved from http://www.opengeospatial.org/projects/initiatives/3dpie

Ortiz, S. (2010). Is 3d finally ready for the web? Computer, 43(1), 14–16.

Parisi, T. (2012). WebGL: Up and Running (First.). United States of America: O'Reilly Media, Inc.

Pender, T. A. (2002). UML weekend crash course. Wiley.

Prieto, I., Izkara, J. L., & del Hoyo, F. J. D. (2012). Efficient visualization of the geometric information of CityGML: application for the documentation of built heritage. In *Computational Science and Its Applications–ICCSA* 2012 (pp. 529–544). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-31125-3_40

RequireJS. (2013). RequireJS. Retrieved February 14, 2014, from http://requirejs.org/

Shephard, N. (2010). Answering Real-World Questions. Retrieved November 26, 2013, from http://www.esri.com/news/arcuser/1010/3danalysis.html

Taraldsvik, M. (2011). Exploring the Future: is HTML5 the solution for GIS Applications on the World Wide Web? technical report, NTNU. Retrieved from http://mats.taraldsvik.net/gd12/publications/html5andgis.pdf

W3C Recommendation. (2012). Web Workers Specification. Retrieved March 5, 2014, from http://www.w3.org/TR/workers/

W3C Recommendation. (2013). Extensible Markup Language (XML) 1.0 (Fifth Edition). Retrieved February 13, 2014, from http://www.w3.org/TR/REC-xml/#dt-cdsection

Walenciak, G., Stollberg, B., Neubauer, S., & Zipf, A. (2009). Extending Spatial Data Infrastructures 3D by Geoprocessing Functionality - 3D Simulations in Disaster Management and environmental Research (pp. 40–44). Presented at the *International Conference on Advanced Geographic Information Systems & Web Services*, IEEE Computer Society. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4782689

APPENDICES

Appendix A: Creation of 3D building on Cesium

Using JavaScript in Cesium API, the 3D buildings on the virtual globe can be created as follows:

Step 1: Creating the 3D polygon geometry

```
// Store coordinates of the building as an array in a variable
var position = ellipsoid.cartographicArrayToCartesianArray
(
    [

      Cesium.Cartographic.fromDegrees(13.045825395539099,52.40883222994816,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045816328061415,52.40883412104174,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045897718848895,52.40900085928126,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045908823045192,52.40899879495449,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045908269281316,52.40899767995136,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.046436964072003,52.40890014762996,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.046437558613613,52.40890133519832,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.046446427074974,52.40889929698004,
      7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.046364065273025,52.408732191841366
      ,7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.04635500816328,52.40873384935529,7
      .6700000000000002),
      Cesium.Cartographic.fromDegrees(13.04635557625896,52.40873496514208,7
      .6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045825948014178,52.408833353711415
      ,7.6700000000000002),
      Cesium.Cartographic.fromDegrees(13.045825395539099,52.40883222994816,
      7.6700000000000002)
    ]
);


// Create polygon geometry using the building coordinates

var polygon_geometry = new Cesium.PolygonGeometry.fromPositions
(
    {
        positions: position,
        vertexFormat: Cesium.PerInstanceColorAppearance.VERTEX_FORMAT,
        extrudedHeight: 1,
        perPositionHeight : true
    }
);
```

Step 2: Creating polygon instance of the geometry

```
var polygonInstance = new Cesium.GeometryInstance
(
    {
        geometry  : polygon_geometry,
        attributes:
        {
            color:Cesium.ColorGeometryInstanceAttribute.fromColor
            (
                new Cesium.Color(0.0, 1.0, 0.0, 1.0) // set color as Green
            )
        }
    }
);
```

Step 3: Creating primitive of the polygon

```
var primitive = new Cesium.Primitive
(
    {
        geometryInstances : PolygonInstance,
        appearance : new Cesium.PerInstanceColorAppearance //WebGL renderer
                    (
                        {
                            closed : true
                        }
                    )
    }
);
```

Output

Figure A- 1: Output of 3D building on Cesium

Appendix B: Creation of buffer zones on Cesium

Using JavaScript in Cesium API, the buffer zones can be created in the form of hemisphere as

Step 1: Retrieve centre position of the hemi sphere

```
// set coordinates of the centre
var centre_position = ellipsoid.cartographicToCartesian
(
    Cesium.Cartographic.fromDegrees(longitude, latitude, 0.0)
);


//create model matrix using centre position
// translates world coordinates to cartesian coordinates
var modelMatrix = Cesium.Matrix4.multiplyByTranslation
(
    Cesium.Transforms.eastNorthUpToFixedFrame(centre_position),
    new Cesium.Cartesian3(0.0, 0.0, 0.0)
);
```

Step 2: Creating the sphere geometry

```
// Create sphere geometry using the input value of radius
var sphereGeometry = new Cesium.SphereGeometry
(
    {
        vertexFormat : Cesium.PerInstanceColorAppearance.VERTEX_FORMAT,
        radius : input_radius // input value for radius
    }
);
```

Step 3: Creating the instance of the geometry

```
var sphereInstance = new Cesium.GeometryInstance
(
    {
        geometry    : sphereGeometry
        modelMatrix : modelMatrix,
        attributes  :
        {
            color : Cesium.ColorGeometryInstanceAttribute.fromColor
                    (
                        new Cesium.Color(1.0, 0.5, 0.0, 0.4)//orange color
                    )
        }
    }
);
```

Output

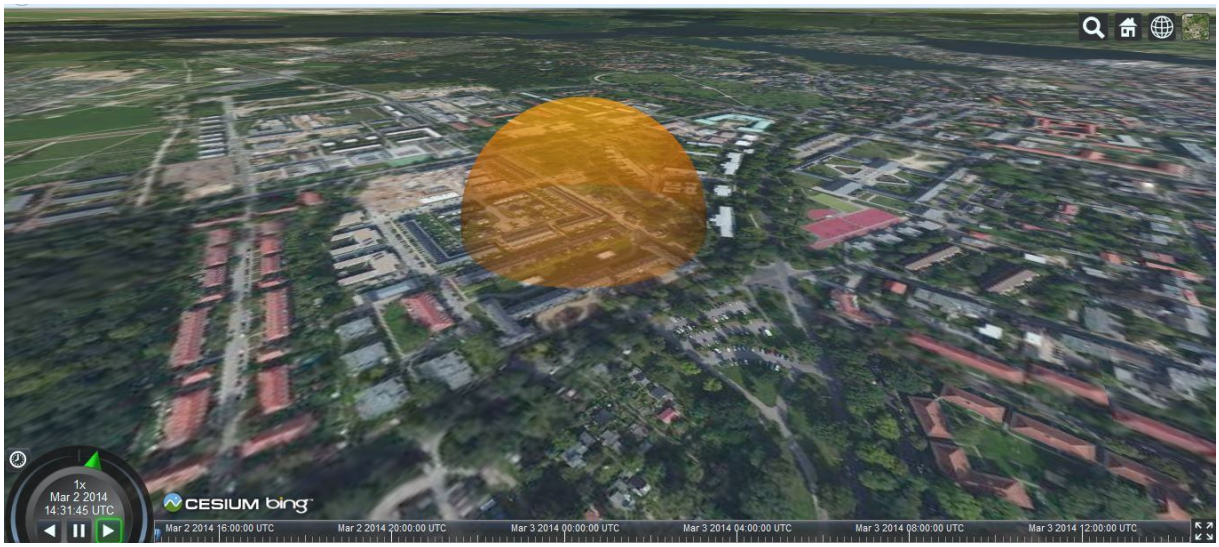


Figure B- 1: Creation of buffer zone on Cesium

Appendix C: Testing intersection between buffer zone and buildings

Step 1: Create axis aligned bounding box of the building geometry

```
var box = Cesium.AxisAlignedBoundingBox.fromPoints(building_coordinates);
```

Step 2: Compute the shortest distance between the centre of the buffer zone and axis aligned bounding box of the building

```
// centre -> centre of buffer zone
// box -> axis aligned bounding box

function shortest_distance(centre, box)
{
    var shortest_dist = 0.0;
    // compare x-axis
    if (centre.x < box.minimum.x)
        shortest_dist+= (box.minimum.x-centre.x)*(box.minimum.x-centre.x);
    if (centre.x > box.maximum.x)
        shortest_dist+= (centre.x-box.maximum.x)*(centre.x-box.maximum.x);
    // compare y-axis
    if (centre.y < box.minimum.y)
        shortest_dist+= (box.minimum.y-centre.y)*(box.minimum.y-centre.y);
    if (centre.y > box.maximum.y)
        shortest_dist+= (centre.y-box.maximum.y)*(centre.y-box.maximum.y);
    //compare z-axis
    if (centre.z < box.minimum.z)
        shortest_dist+= (box.minimum.z-centre.z)*(box.minimum.z-centre.z);
    if (centre.z > box.maximum.z)
        shortest_dist+= (centre.z-box.maximum.z)*(centre.z-box.maximum.z);

    return shortest_dist;
}
```

Step 3: Perform intersection test. If square of the shortest distance is less than or equal to the square of the radius, they intersect, otherwise they do not intersect.

```
// Test intersection between buffer zone and building geometry

function check_intersect(centre, radius, box)
{
    var squared_distance = shortest_distance (centre, box);
    if (squared_distance <= radius*radius)
        return true;
    else
        return false;
}
```

Output

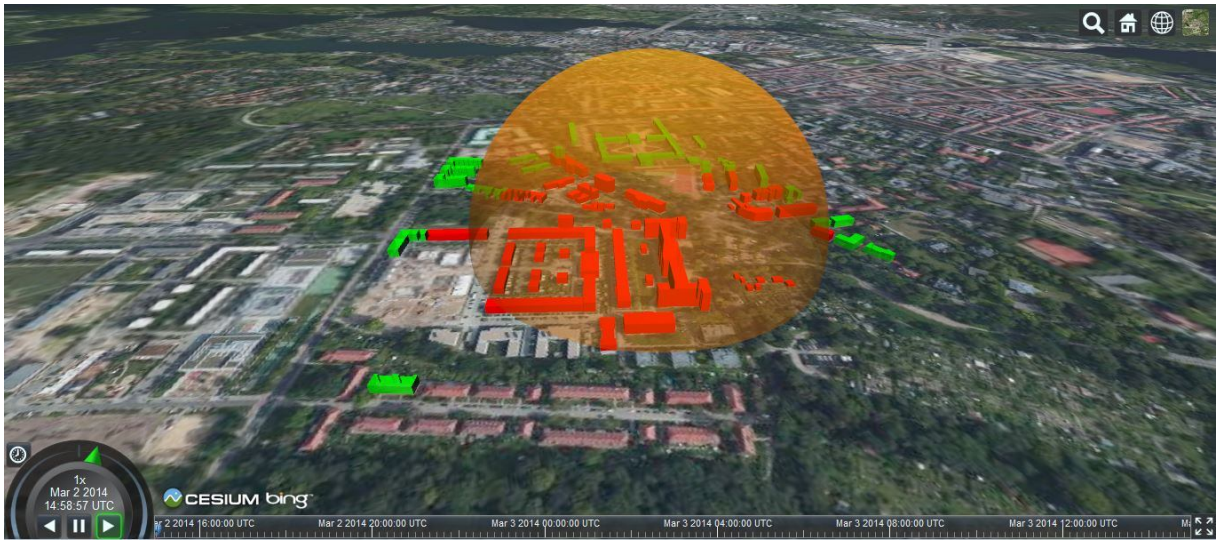


Figure C- 1: Testing intersection between buffer zone and buildings