MASSEY UNIVERSITY

Geographic Data Visualization with Immersive Virtual Reality

Author
Yang JIANG

Supervisor
Dr Arno LEIST

A thesis submitted in fulfillment of the requirements
for the degree of Master of Information Sciences
in the

Software Engineering
159.888 Computer Science Professional Project

October 28, 2016

Abstract

Virtual reality technology has been proved to be beneficial to visualize geographic data. It can be broken down into the pseudo-3D display with 3D interaction and the immersible depth of vision of a real 3D experience. The former not only has been already studied for a long time in the domain of visualizing the earth and environmental sciences, but also got a favorable result in both theory and practice area. On the other hand, the latter is yet to be completely revealed. This paper makes a demonstrate of taking advantage of the Keyhole Markup Language (KML) [13] as the geographic visualization markup language can not only benefit from the global geospatial group but contribute to all kind of communities. In consideration of the ranges any necessary sensors and the equipment costs, taking advantage of Google Cardboard [11] and Android platform have proved to be well-suited for constructing the immersive virtual reality environment. This paper also revealed an immersive virtual reality based intuitive nature system that provides attractive and efficient methods for simultaneously visualizing geographic data from the different source. They are exposed by presenting the implementation of an application that includes both front (client) and back-end (web server) for geographic data visualization.

Keywords: Geographic Information, Visualization, Virtual Reality

Contents

List of Figures

List of Tables

1 Introduction

There has been an increased interest in the exploration of Virtual Environments (VE) [19], sometimes called Virtual Reality. The first fifteen years of the 21st century has seen significant, rapid advancement in the development of virtual reality became much more dynamic, the term Virtual Reality itself became extremely popular, and there was a broad range of applications were developed relatively fast. They offer significant benefits in many areas, such as architectural walkthrough, scientific visualization, modeling, designing and planning, training and education, telepresence and teleoperating, cooperative working and entertainment [23]. Among these applications, virtual reality technology has been proved it offers new and exciting opportunities for users to interact visually with and explore 3D geographic data [19].

1.1 Overview and Objectives

In the previous practices of visualizing geographic data with virtual reality were mostly using 3D representations of objects and displayed them on a 2D monitor. This pseudo-3D nature of virtual reality is not enough for offering what people desire, and they want able to step into the world and interact with it, instead of watching the 2D projection image on the monitor. That is the ultimate motivation of virtual reality technology - a real 3D experience with immersive stereoscopic 3D visuals.

Nowadays, given the rapid advancement in the development of computer technology especially small and powerful mobile technologies have exploded while prices are continually driven down. The rise of smartphones with high-density displays and 3D graphics capabilities has enabled a generation of lightweight and functional virtual reality devices. It seems clear that we step into a critical period of immersive virtual reality industry while multiple virtual reality related products that finally seem to enter the market constantly. However, still, there is a lack of research in both theory and practice way for visualizing geographic data in the immersive virtual reality.

In order to evaluate how geographic data visualization with immersive virtual reality affect user interfaces and human-computer interactions, an immersive virtual reality application that composed of a database management system and a graphic display system for geographic data visualization is developed for the purpose of this study. This paper also highlighted the essential considerations that get involved in such implementations: the ranges and capabilities

of any necessary sensors to create the immersive virtual reality; evaluation of the minimum equipment costs; shared geographic visualization markup language; geodetic-mapping coordinates; performance of 3d graphic. In this thesis, a background of geographic data visualization is presented. Then, details of the related technology and implementation are described. Finally, there is a discussion and conclusion around the results and future research.

1.2 Background

The Geographic Information System (GIS) is a broad term, it often refers to many different technologies, processes, and methods that designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data [39]. A GIS combines a database management system and a graphic display system that tie to the process of spatial analysis [30]. Indeed, GIS has been widely used in the analysis of the Earth and environmental data, mostly used in 2D, map-based systems. However, significant problems have had exposed. First, GIS itself only handle 2D data; second, displays are limited to spatial views of the data; third, the capability of supporting user interaction with negligible data [31]. Nevertheless, the concept of taking advantage of GIS to visualize the earth and environmental sciences data has been already studied for a long time in both theory and practice area, and that is called Virtual Globe (VG) technology. Although, most of the virtual globe products are pseudo-3D nature based, but still, they allow users to interact with an environment that makes the data and information present easier to understand [32]. Therefore, it dramatically has become a powerful tool for navigating geospatial data in 3D and contribute to all kind of communities across different usage till now.

Essentially, the success of virtual globes is the improvement of human understanding in the following aspect. [32].

- **pseudo-3D** Allows users to interact with an environment that they naturally understand.
- **Transportability** Digital data are easily transported.
- **Scalability** Can be view at any scale.
- **Interactivity** Provides an interactive experience for users.
- **Choice of topics** Topics can be changed dynamically, and presented individually or together.
- **Currency** The data presented can be of any age, including real time.
- **Client-side** Puts the power in the hands of the user.

Virtual globe technology is beneficial to education. For teaching spatial thinking, virtual globes offer tremendous opportunities, and it can be expected that they will greatly influence how a new generation will perceive space and geographic processes, said by Nuernberger [26]. It also helps scientific collaboration research, such as the EarthSLOT [5]. Moreover, Butler points out virtual globes can be used as an invaluable tool in disaster response [3], [25]. Virtual globe technology has many exciting possibilities for environmental science. The easy-to-use,

intuitive nature system, provide attractive and efficient means and methods for simultaneously visualizing four-dimensional environmental data from different sources that driving a greater understanding and user experience of the Earth system [2].

The Open Geospatial Consortium (OGC) is committed to making quality open standards for the global geospatial group. These standards were decided through a consensus based process and are freely available for anyone to sharing of the world's geospatial data. They have made contributions to many communities including government, commercial organizations, non-governmental organizations, academic and research organizations [27]. To use a markup language maintained by OGC for the creation of 3D geographic maps and associated spatial data allows scientists to publish the latest information in a single, simple data file format without technical assistance. More importantly, it potentially allows environmental scientists to visualize 4D data (i.e. time-dependent three-dimensional data) from data files created in the different period.

A markup language maintained by the Open Geospatial Consortium [27] plays an essential role in virtual reality implementation. By taking the use of a markup language, scientists are able to publish data in a single, simple data file format without technical assistance. In spite of capabilities vary from products to products, but virtual globes always provide support for a file format data exchange and the ability to simultaneously display multiple datasets. Blower et al. point out [2] Google Earth which has the largest community creates Keyhole Markup Language (KML) [13] files as its primary method for visualizing data (KML is an international standard maintained by the OGC); NASA World Wind [24] imports data from tile servers, OGC web services and limited support for KML, it has more focus toward scientific users; ArcGIS Explorer [6] is a lightweight client to the ArcGIS Server, it can import data in a very wide range of GIS formats, including KML. Some of the virtual globes products are using Virtual Reality Modeling Language (VRML) [42] that is a language for describing 3D objects and interactive scenes on the World-Wide Web (WWW) [44], It has been superseded by X3D [45].

The KML is a somewhat limited language. It can only describe simple geometric shapes, such as points, lines, and polygons, and is not extensible. By compared with Geography Markup Language (GML), in many respects, GML 3.0+ is much more sophisticated and allows the rich description of geospatial features such as weather fronts and radiosonde profiles. For the above reasons, KML is currently not suitable as a fully-featured, general-purpose environmental data exchange format. Nonetheless, it still earns the acceptance from an increasing number of scientists. From the point of view of usability, KML spans a gap between very simple (e.g. GeoRSS) and more complex (GML) formats, which makes it easy for non-technical scientists to share and visualize simple geospatial information which can then be manipulated in other applications if required. After all, it is important to be aware of that virtual geographic data visualization (or KML) does not attempt to replace more sophisticated systems.

In recent years, given the rapid development of technique progress in computers and pipelined

3D graphics, the immersive virtual reality not only frequent occurrences nearly in all sorts of media, but also it has a mess of related products developed by manufacturers over the world. For example, Google has released similar virtual reality products such as the Google Cardboard, a DIY immersive virtual reality headset that drives by smartphone; Samsung has taken this concept further with products such as the Galaxy Gear, which is mass produced and contains features such as gesture control; the 3D camera that can capture a 360 degrees field of view. However, it is not mature enough to eliminate the equipment limitation and becomes a universal technology in daily human life by comparison to the pseudo-3D virtual reality technology. For instance, when it comes to exploring, routing or getting to places, most people should just reach for Google Earth or Google Map.

Immersive virtual reality provides an easy used, powerful, intuitive way of user interaction. The user can experience and manipulate the simulated 3D environment in the same way they act in the real world, without any preparation or understanding of the complicated user interface works. It soon became a perfect tool that is beneficial to architects, designers, physicists, chemists, doctors, surgeons, etc. Without a doubt VR has a great potential to change our life, the expectation from this technology is much more than it can offer yet [23].

2 Technology

In this chapter, details of the related technologies are presented. They are Android smartphone, a low-cost way to experience immersive virtual reality environment; OpenGL ES in Android; KML for geographic visualization; Golang RESTful web server for managing data in the back-end.

2.1 Virtual Reality Device

There are following reasons for using Android smartphone as the virtual reality device. The intention is to identify immersive virtual reality device that not only low-cost but also a standard, customer-friendly device. That is the smartphone, and it had an incredibly fast growth trend in the last few years and a good promising market prospect 2.1. After all, it contains all the necessary sensors and positioning systems to measure motion and accurately track device movements - Six degrees of freedom (DOF) - position coordinates (x, y and z offsets) and orientation (yaw, pitch and roll angles). Additionally, 15$ Google Cardboard kit turns Android or iOS smartphone to immersive virtual reality device. According to International Data Corporation (IDC), Android dominated the smartphone market with a share of 87.6% in the worldwide 2.2. Moreover, there is an existing Google VR SDK [12] for Android supports.

FIGURE 2.1: Global Smartphone Shipments Forecase [4]

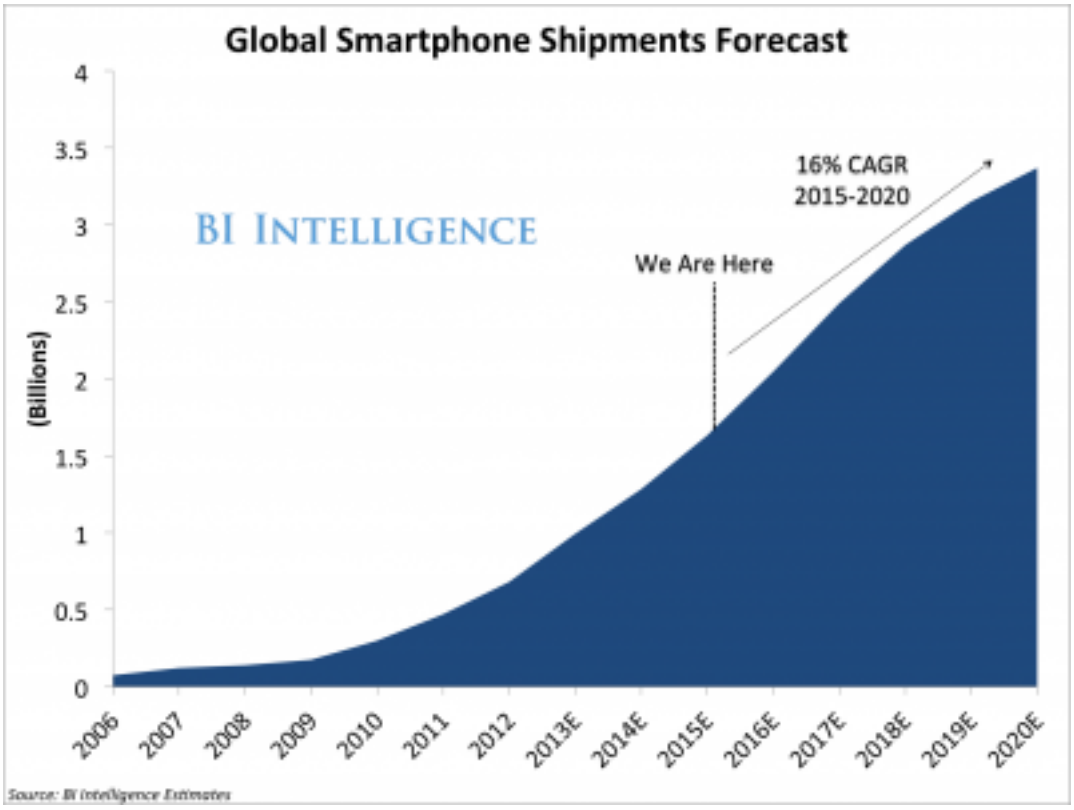
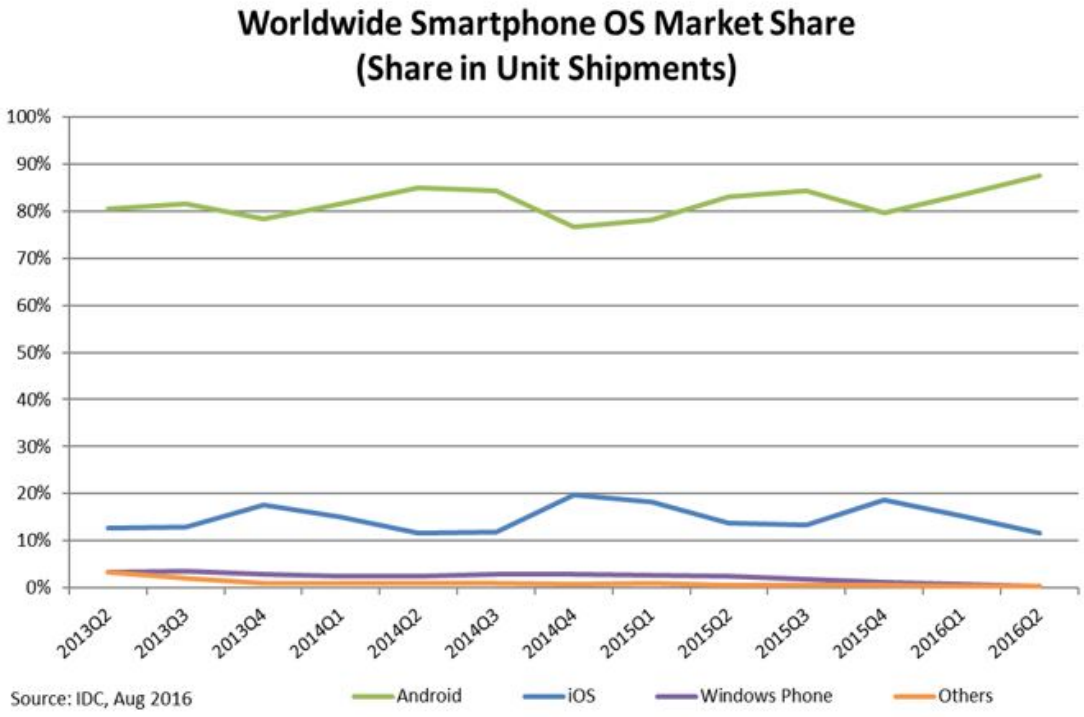


FIGURE 2.2: Smartphone OS Market Share [20]



2.2 OpenGL ES

Android includes support for high-performance 2D and 3D graphics with the Open Graphics Library, specifically, the OpenGL ES API [14]. OpenGL ES is a branch of the OpenGL specification intended for embedded devices. The Google VR SDK requires the device has a minimum OpenGL ES 2.0 support. Table 2.1 shows a version list of OpenGL ES API that Android supported.

TABLE 2.1: OpenGL ES API specification supported by Android

OpenGL ES Version	**Android Version**
OpenGL ES 1.0	Android 1.0 and higher
OpenGL ES 1.1	Android 1.0 and higher
OpenGL ES 2.0	Android 2.2 (API level 8) and higher
OpenGL ES 3.0	Android 4.3 (API level 18) and higher
OpenGL ES 3.1	Android 5.0 (API level 21) and higher

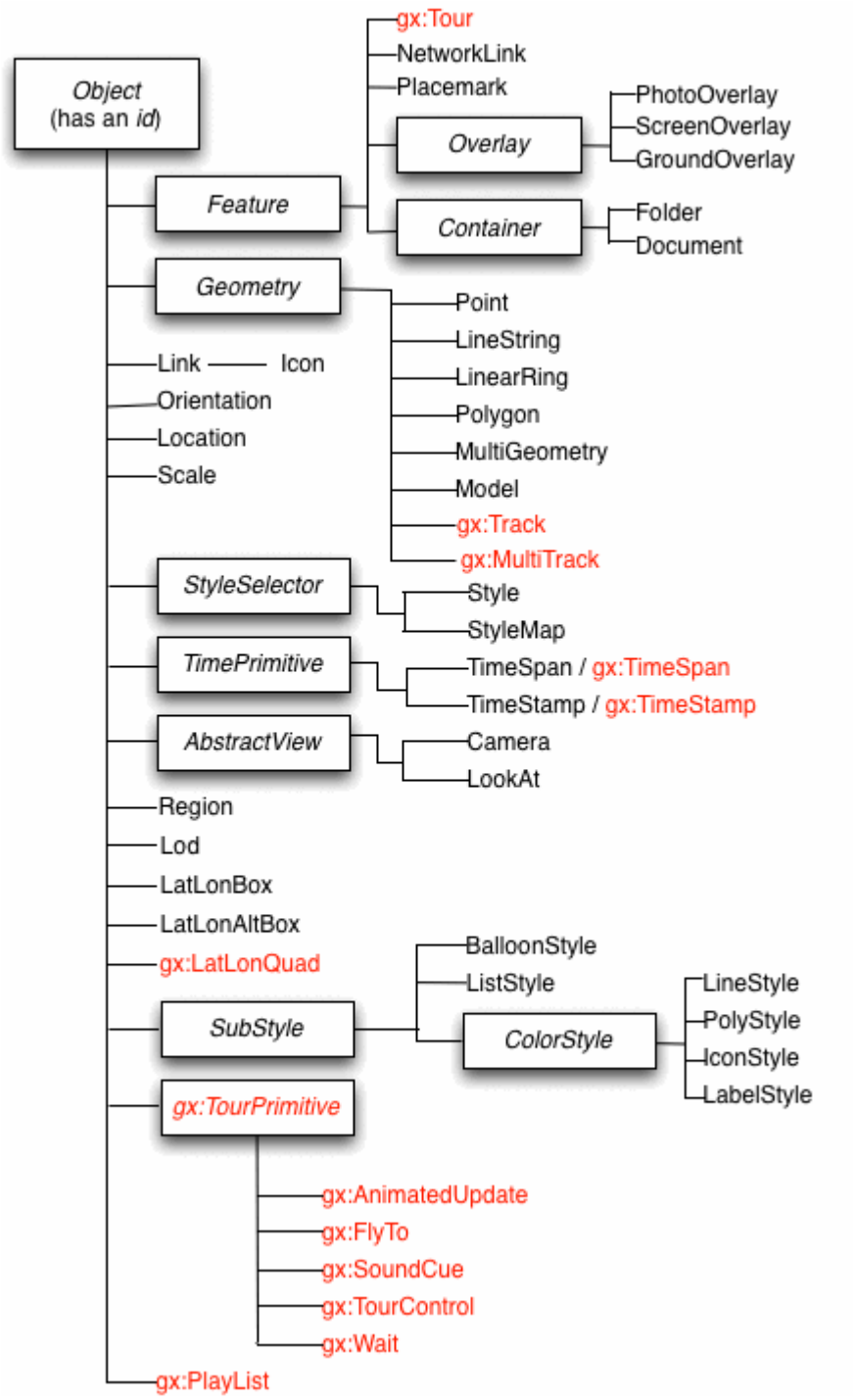
2.3 Geographic Visualization Markup Language

I was looking for a simple markup language for geographic data visualization. It should be not only able to represent a geographic data in the virtual reality, but also be beneficial to publish and consume data in interoperable formats without the needs of technical assistance.

The Keyhole Markup Language (KML) can be combined with other supporting files such as imagery in a zip archive, producing a KMZ file. KML offers features for expressing geographic annotation and visualization. The annotations of KML features are not designed as machine-readable XML, but a human readable plain text or simple HTML. The `Networklink` facility in KML contributes to a real-time data which is important in the environmental sciences. It allows all or part of the dataset to be automatically refreshed by the URL, to ensure the user always sees the latest information.

More importantly than the satisfaction of needs in the application, it supported by many virtual globes and other GIS systems. Therefore the KML already becoming a de facto standard [2] that can be manipulated in other softwares if required.

FIGURE 2.3: KML schema [13]



2.4 Network

Real-time data are very important in the environmental sciences [2]. One of the key strengths of virtual reality applications are not only easy-to-use, and intuitive nature, but also the ability

to efficiently incorporate new data. Therefore, a web server is needed. A RESTful web server to support communication with the client, and a remote file server to synchronize data are included in the application.

Go (often referred to as golang [17]) is an open source programming language, and it is compiled, concurrent, garbage-collected, statically typed language developed at Google in late 2007. It was conceived as an answer to some of the problems they were seeing and developing software infrastructure [8]. Surprisingly, the rise of Go was growing so fast that each month the contributors outside Go team itself are already more than the contributors inside the Go team. Additionally, Golang is well suited for developing RESTful API's. Its net/http standard library provides a set of key methods for interacting via the HTTP protocol. For the above reasons, the Golang is selected in this paper for developing the server.

On the client side (Android platform), Volley is being used for transmitting network data (Volley is an open sourced HTTP library that makes networking for Android apps easier and most importantly, faster [18]). Furthermore, Jsoup (Java HTML Parser [22]) is being introduced for analyzing HTML format response.

3 Implementation

In this chapter, details of the major implementation are revealed. First, briefly introducing Google VR SDK setup and drawing with OpenGL ES. Then an explanation of the web server design. After that, the creation of 3D virtual scene is highlighted. Finally, the implementation for device movement and object intersection detection are clarified.

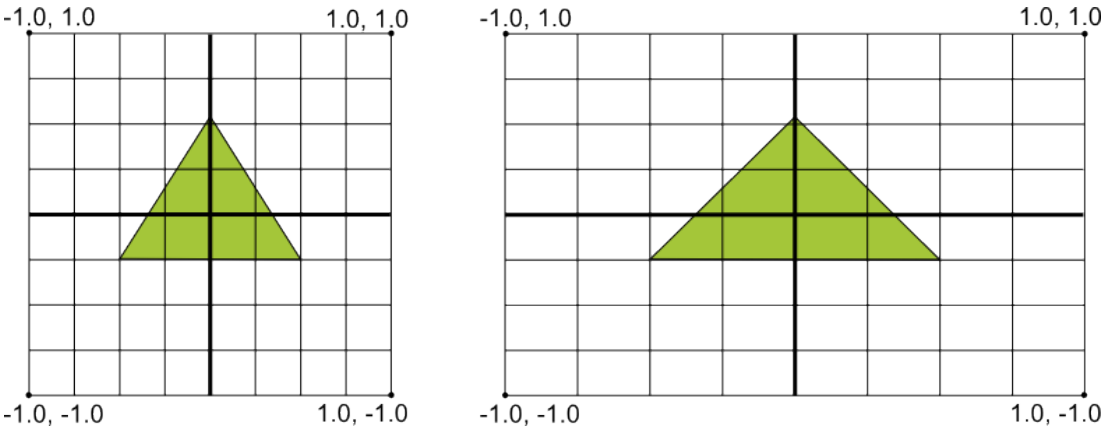
3.1 Google VR SDK

The Google VR SDK repository is free and accessible from*https://github.com/googlevr/gvr-android-sdk*, where we can get access to any necessary libraries and examples. The SDK libraries locate in the libraries directory of the repository as *.aar* files [9]. This project has two dependencies on `base` and `common` Google VR SDK modules.

3.2 OpenGL ES

OpenGL assumes a square coordinate system, by default, happily draws those coordinates onto the screen. However screens can vary in size and shape, that is to say, most screens are typically non-square screen. The illustration below 3.1 shows the assumed uniform coordinate system of an OpenGL frame on the left, and how these coordinates map to an exemplary non-square device screen in landscape orientation on the right.

FIGURE 3.1: Default OpenGL coordinate system (left) mapped to a typical Android device screen (right) [14]



Therefore, OpenGL projection modes and camera views have to be applied to the OpenGL rendering pipeline for coordinates transformation, so the graphic objects have the expected proportions on any display. The projection matrix will recalculate the coordinates of graphics objects, and the camera view matrix will create a transformation that renders objects from a specific eye position.

The implementation is divided into two phases. First, working out the model matrix, view matrix, and perspective matrix in CPU (Android programming in this case). Secondly pass them to GPU for the rest of calculation (OpenGL Shading Language Programming, i.e. GLSL or GLslang), such as explicit projection matrix, coordinates transformation, lighting, or more abstract circular ring 3.7. The GLSL shaders themselves are a set of strings that passed to the hardware driver for compiling within an application using the OpenGL API's entry points [40].

TABLE 3.1: OpenGL compute

What	**How**	**Where**
Model Matrix	translationMx * scaleM * rotationM * identityM(1)	CPU
Camera Matrix	Matrix.setLookAtM(positionV, lookAtV, upV)	CPU
View Matrix	eye.getEyeView() * cameraM	CPU
Perspective Matrix	eye.getPerspective(zNear, zFar)	CPU
Projection Matrix	perspectiveM * viewM * modelM	GPU
Vertex'	projectionM * vertex	GPU

3.3 Web Server

See the example below, a simple file server on port `8080` to serve a directory on disk "*/tmp*" under an alternate URL path "*/files/*", using `StripPrefix` to modify the request URL's path before the `FileServer` sees it.

```
http.Handle("/files/", http.StripPrefix("/files", http.FileServer(http.Dir("./tmp"))))
http.ListenAndServe(":8080"), nil)
```

For RESTful APIs setup, I introduce a free framework Go-Json-Rest [21], it is a thin layer designed by KISS principle (Keep it simple, stupid) and on top of native net/http package that helps building RESTful JSON APIs even easier.

3.3.1 Assets

The file server processes the requests and delivers the result (the particular file) in a standard web format back to the client. Table 3.2 indicates the folder structure served by the server.

TABLE 3.2: Assets structure

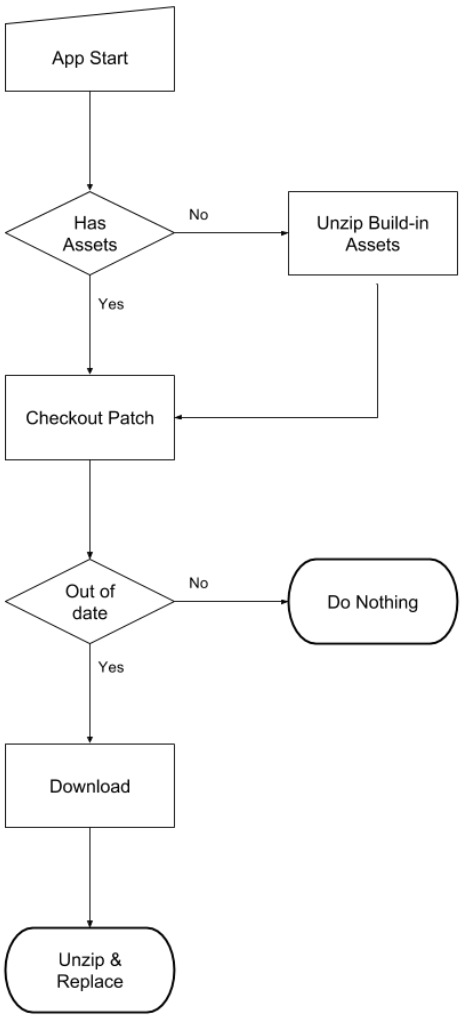
Path	Usage
\assets	Root
\assets\static.zip	The compressed Patch (see 3.3.2)
\assets\static\kml	KML storage (see 3.4.1)
\assets\static\layer	KML storage (see 3.4)
\assets\static\model	Extra model storage (see 3.6.3)
\assets\static\resource	Resource (eg. images) storage

3.3.2 Patch

A Path is for the server to guarantee the latest data (if any) will be pushed to each client (Android smartphone). It server as a compressed ZIP file, and it contains one or more files that require client to update. Patch validation is happening whenever the app starts. First, the client sent requests for the patch file "*http://xxx.xxx.xxx.xxx:8080/assets/static.zip*" from the file server. Before actual download the file, take the `lastModifiedTime` data of remote Patch from HTTP response headers, and compare it with the other `lastModifiedTime` data of local's Patch file. Only when the local's Patch is out of date, the client continues to download the remote Patch file and replacing any existing local files. For a special scenario when the app was just installed in the first time launch, also the network is disconnected. A built-in

default Patch that included in the APK (Android app binary) will be uncompressed to avoid no available data. Diagram 3.2 shows the simplified process.

FIGURE 3.2: Patch check



3.4 Scene

The Keyhole Markup Language (KML) is contributed to the application as the geographic visualization markup language. KML files from folder "*/assets/static/layer* are visible to the user, and each KML file represents an individual 3D scene which contains any necessary geographic data related to the topic.

As you can see from Table 3.2, there are two assets folders contains KML files - "*/assets/static/layer*" and "*/assets/static/kml*". These files are literally the same, but existing in different concepts for achieving the purpose of categorizing. By making use of `Networklink` facility, an individual KML file can contains one or more other KML files by given URLs. Therefore, folder

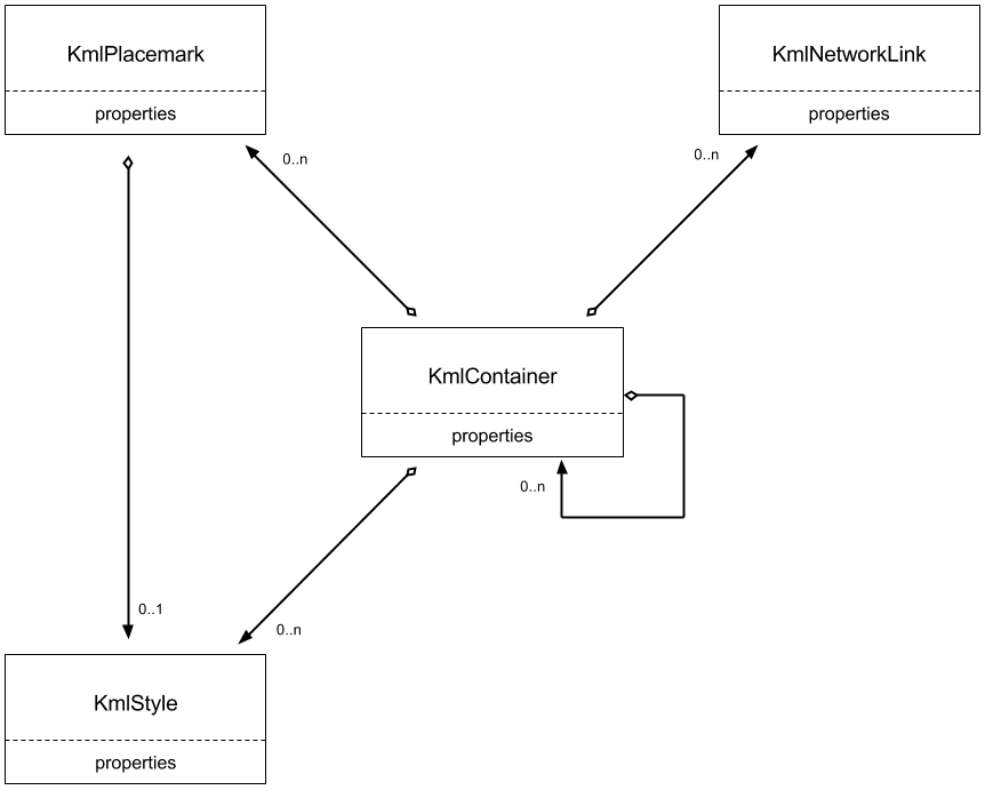
"*/assets/static/layer*" intends to be the scene topic (KML file) storage that visible and selectable to the user, and any inside topic could include one or more topics that exist in folder "*/assets/static/kml*".

It has many advantages to dividing the space with certain patterns during the space creation. Such as, runtime graphical analysis and optimization, intersection and collision detection.

3.4.1 Geographic Visualization Markup Language

There are only some of KML features from KML schema 2.3 are be used in this application. They are `Container`, `Style`, `Placemark`, and `NetworkLink`. The KML parser I am using is not coded from scratch, and it is based on the open-source library `android-maps-utils` [10] but with certain modification and extension: getting rid of `GoogleMap` dependency; extending `NetworkLink` facility support that was one of the unsupported features in the library.

FIGURE 3.3: kML parser simple



3.4.2 Space Partition

Space partition often used for optimizing collision detection algorithms among polygonal models. These algorithms are often expensive operations and can significantly slow FPS down.

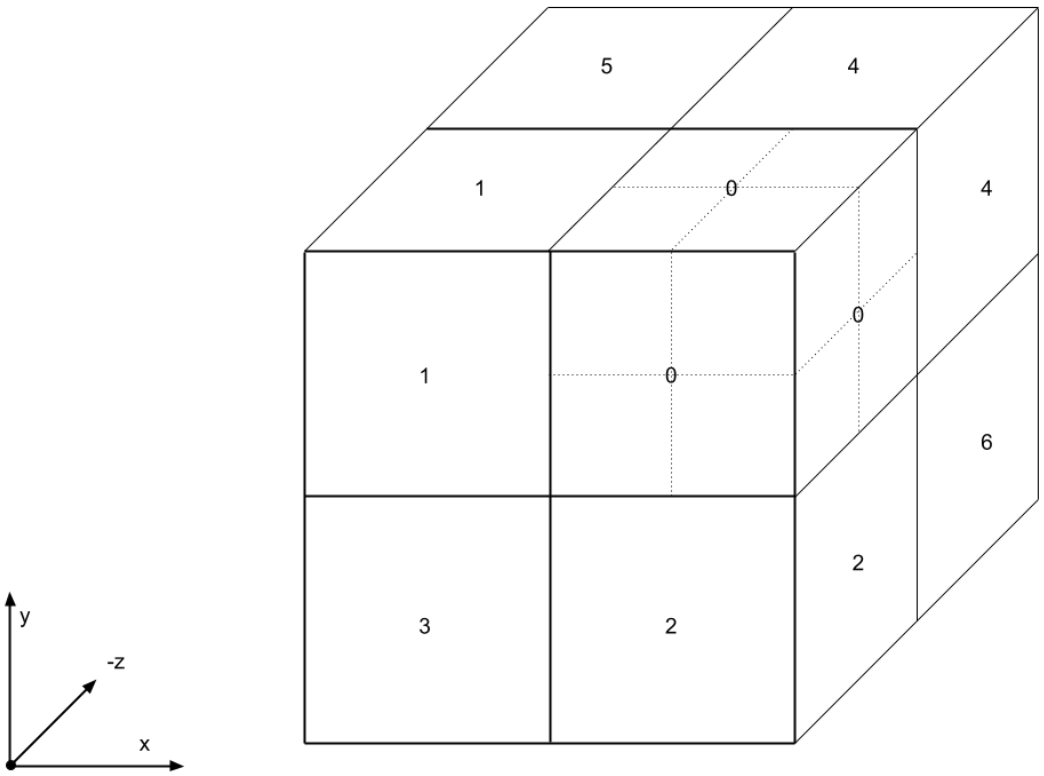
Although there is no collision detection in this application yet, there is an intersection detection between the ray tracing (see 3.7) and other objects. Space partition is contributed to reducing the ray-object intersection test load by skipping objects that locate far away from current ray tracing area. It avoids doing an n^2 times intersection detection on all objects.

A axis-aligned Octree is implemented for the space partition **??**. It has a predefined constant positive integer to decide whether or not a new partitioning should happen - a minimum number of objects allowed exist in the same cell. This number is important for the purpose of reducing intersection detections. I have taken 5, object complexity and cell complexity need to be considered.

- **Object Complexity** `Plackmarker` is doing ray-sphere detection (see 3.10.1)
- **Cell Complexity** Cell is doing ray-box detection (see 3.10.3)

If the number is positive infinity - whole space seen as a cell and no further space partition is required, this is not reduce anything but also increase to $n + 1$ times of detections (n times for ray-object, 1 times for ray-cell). If the number is 1 - each cell only contains one object, this also not reduces the number of detection times, but increase to at least $2\,n$ times. Since the ray-box detection action is much cheaper then ray-sphere's, an appropriate value can eventually reduce the overall intersection detections.

FIGURE 3.4: Octree division



See Diagram **??**, the parent cell has eight indexes indicate the different relative position inside the parent cell. These indexes are important for the next time of division, where the objects in parent cell need to be relinked to a new cell. On the other words, a new object will be linked to the parent cell only if the existed objects is less than the predefined constant value. If not, the parent cell will be spatially divided into eight cells. Then, the existing objects will be unlinked from the parent cell and relink to a new cell.

The integer index is not chosen randomly. It is defined by its geometric meaning - three boolean value that indicates the three axis-relative delta value. Table 3.3 gives the relationship between the index and three boolean values.

The three delta values of any position P in a cell with known center O are:

$$dx = P_x - O_x$$
$$dy = P_y - O_y$$
$$dz = P_z - O_z$$

The relationship between the index and three boolean values as follow:

TABLE 3.3: Octree Octant

Binary Index	**Octant**	**Geometric Meaning**
0x00000000	T, T, T	dx > 0, dy > 0, dz > 0
0x00000001	F, T, T	dx < 0, dy > 0, dz > 0
0x00000010	T, F, T	dx > 0, dy < 0, dz > 0
0x00000011	F, F, T	dx < 0, dy < 0, dz > 0
0x00000100	T, T, F	dx > 0, dy > 0, dz < 0
0x00000101	F, T, F	dx < 0, dy > 0, dz < 0
0x00000110	T, F, F	dx > 0, dy < 0, dz > 0
0x00000111	F, F, F	dx < 0, dy < 0, dz < 0

$\therefore$ The transformation from known index to three boolean values (Octant):

```
octant[] = (index & 1, index & 2, index & 4)
```

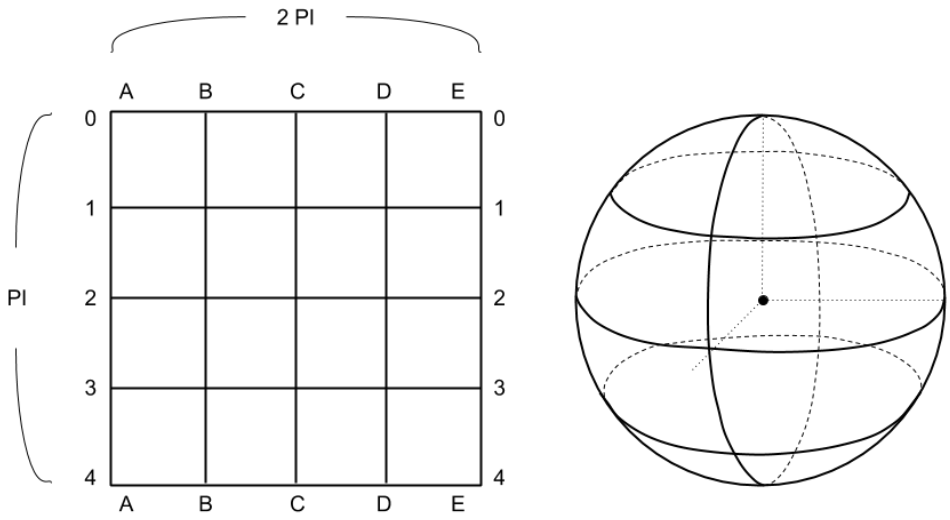
The transformation from known Octant to index:

```
For Each octionn[i]:
    index |= (1 < < i)
```

3.5 Earth

UV Sphere often used in the situation where requires a very smooth, symmetrical surface. In this application, the Earth model is created as the UV sphere. Similar to latitude and longitude lines of the Earth, it uses rings and segments (near the poles, the vertical segments converge on the poles). Therefore, the UV texturing for 2D earth image mapping to the 3D sphere's surface can be conveniently calculated during its vertex creation process.

FIGURE 3.5: UV sphere mapping

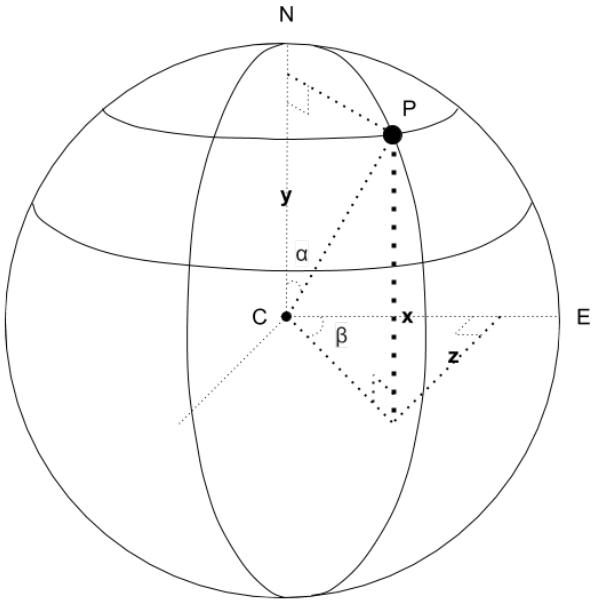


The Diagram 3.5 illustrates the mapping from 2D plane to 3D UV sphere's surface which has 5 rings and 4 segments. As we can seen, vertex $A0$, $A1$, $A2$, $A3$, $A4$ and $E0$, $E1$, $E2$, $E3$, $E4$ are duplicated; vertex $A0$, $B0$, $C0$, $D0$, $E0$ converge together in the pole, as well as $A4$, $B4$, $C4$, $D4$, $E4$. Also, in the UV sphere, each ring spans $2\,\pi$ radians, but each segment only spans π radians.

The total vertex count for a UV sphere is:

$$\texttt{VerticesCount} = \texttt{RingsCount} \times \texttt{SegmentsCount} \tag{3.1}$$

FIGURE 3.6: UV sphere vertex



If a vertex P on the UV sphere belongs to ring r and segment s:

$$v = r \times \frac{1}{\texttt{RingsCount} - 1}$$

$$u = s \times \frac{1}{\texttt{SegmentsCount} - 1}$$

$$\measuredangle\alpha = v \times \pi$$
$$\measuredangle\beta = u \times 2\,\pi$$

$\therefore$ The vertex P (x, y, z) can be calculated:

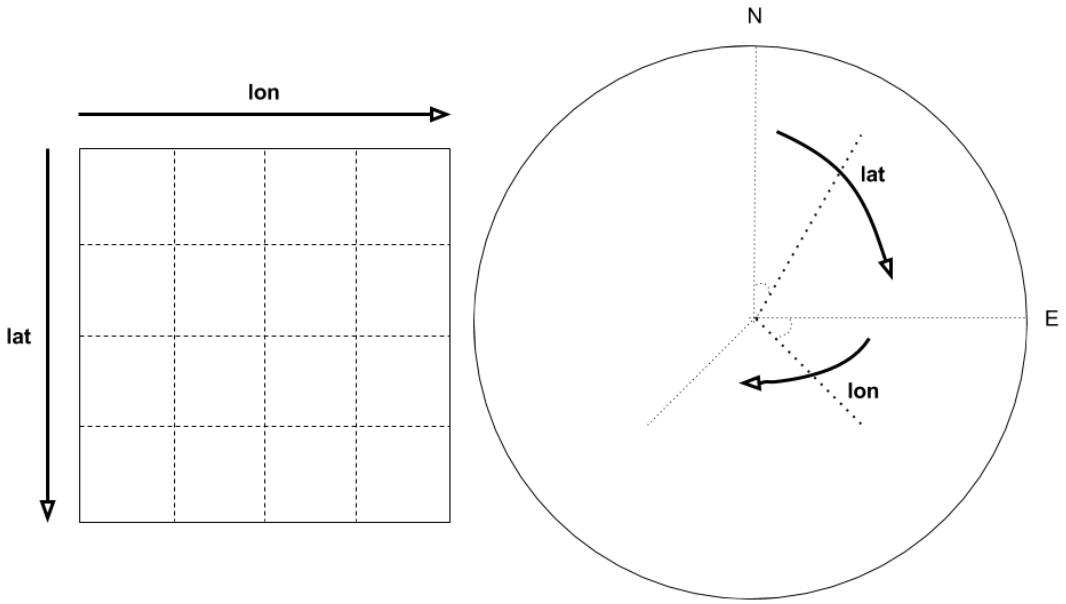
$$x = (\sin(\alpha) \times radius) \times \cos(\beta)$$
$$y = \cos(\alpha) \times radius$$
$$z = (\sin(\alpha) \times radius) \times \sin(\beta)$$

The UV texturing (x, y) mapping for vertex P is:

$$x = u$$
$$y = v$$

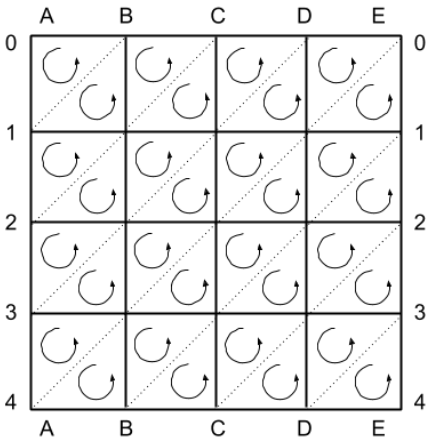
It is vital to recognize the flip-side effect caused by the processing order of texturing. Diagram 3.7 shows a flip on the longitude direction. Which is not looking correct from outside of the Earth, but it is a precise mapping when the user is viewing from the inside.

FIGURE 3.7: UV sphere flip-side effect



Given an ordering of each triangle's vertices, a triangle can appear to have a clockwise winding or counter-clockwise winding. Using OpenGL features Culling Face and Winding Order together to determine whether the triangle is visible from the front or the back side. In order to guarantee an inside visible only, `glFrontFace(GL_CCW)` and `glCullFace(GL_BACK)` can be adopted. The vertex indexes is ordered as Diagram 3.8.

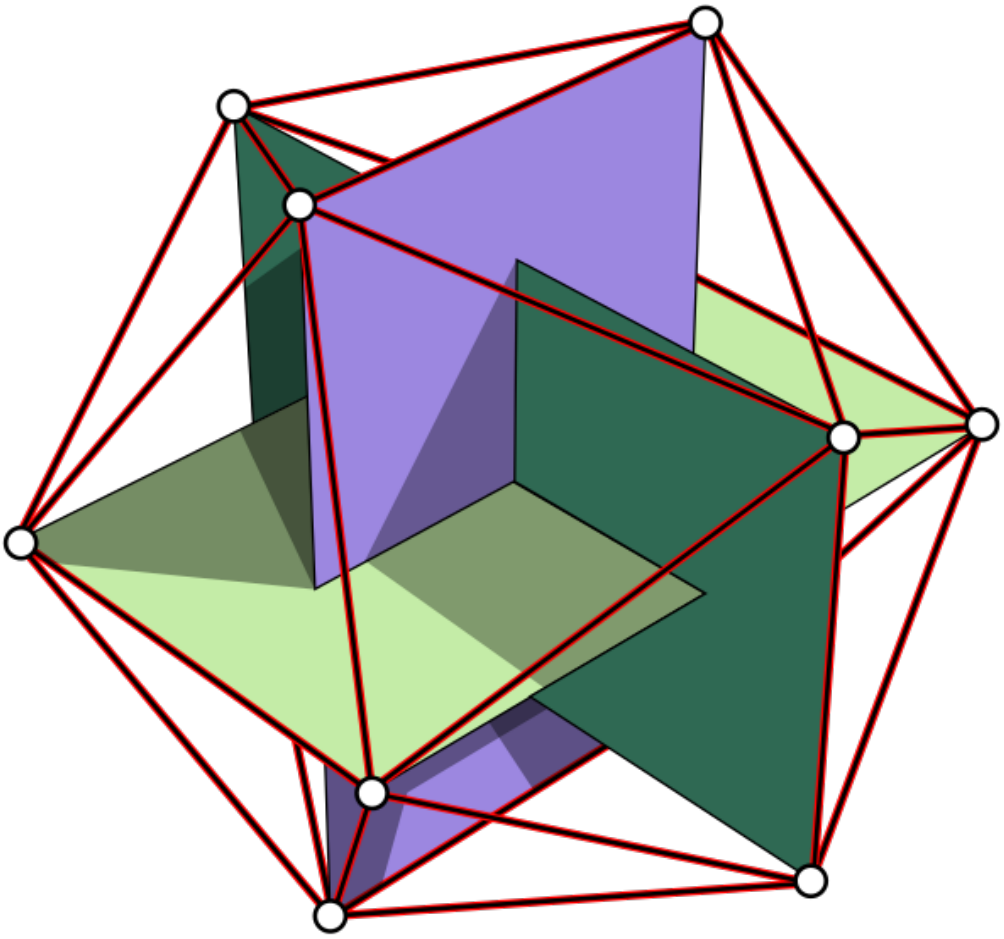
FIGURE 3.8: UV sphere CCW



3.6 Placemark

The vertex generation for `Placemark` is a recurring process of subdividing icosphere. Figure 3.9 is an icosahedron, the corners of three orthogonal rectangles are the initial vertices for `Placemark`.

FIGURE 3.9: Icosahedron rectangles [35]



Rounding the icosphere by subdividing a face to an arbitrary level of resolution 3.4. One face can be subdivided into four by connecting each edge's midpoint, then push the midpoints to the surface of the sphere 3.10.

FIGURE 3.10: Icosphere refinement

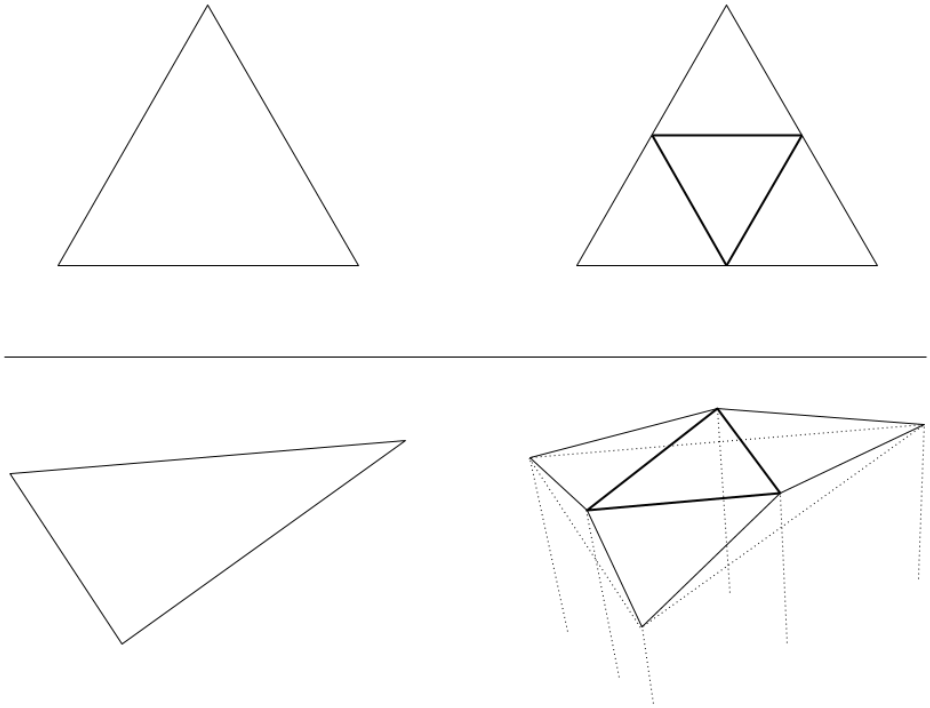


TABLE 3.4: Rounding Icosphere

Recursion Level	**Vertex Count**	**Face Count**	**Edge Count**
0	12	20	30
1	42	80	120
2	162	320	480
3	642	1280	1920

Rounding icosphere is not only increasing the vertex number but also adjusting vertex indexes. As we can see from the Diagram 3.11, the refinement of each face has three steps.

First, get the midpoint of each edge.

$$\overrightarrow{d} = \frac{\overrightarrow{A} + \overrightarrow{B}}{2}$$

$$\overrightarrow{e} = \frac{\overrightarrow{A} + \overrightarrow{C}}{2}$$

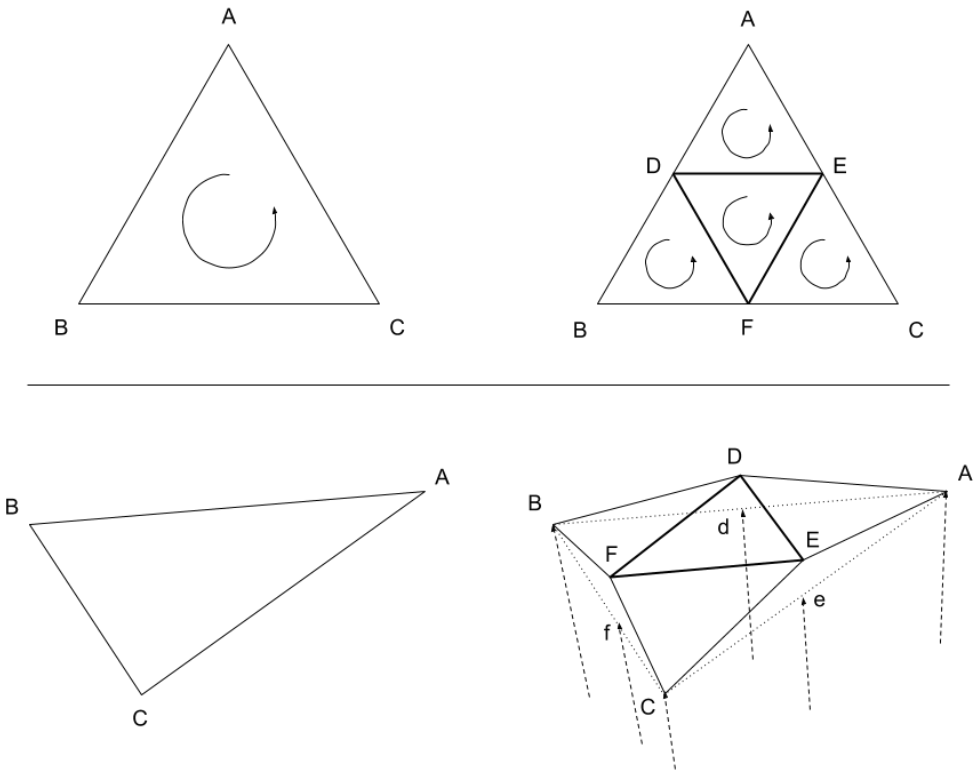
$$\overrightarrow{f} = \frac{\overrightarrow{B} + \overrightarrow{C}}{2}$$

Second, push midpoints to the surface of the unit sphere(1).

$$\overrightarrow{D} = \texttt{normalize}(\overrightarrow{d})$$
$$\overrightarrow{E} = \texttt{normalize}(\overrightarrow{e})$$
$$\overrightarrow{F} = \texttt{normalize}(\overrightarrow{f})$$

Third, remove $\triangle$`ABC` in the vertex indexes list, add $\triangle$`ADE`, $\triangle$`DBF`, $\triangle$`EFC`, and $\triangle$`DEF`.

FIGURE 3.11: Icosphere vertex index



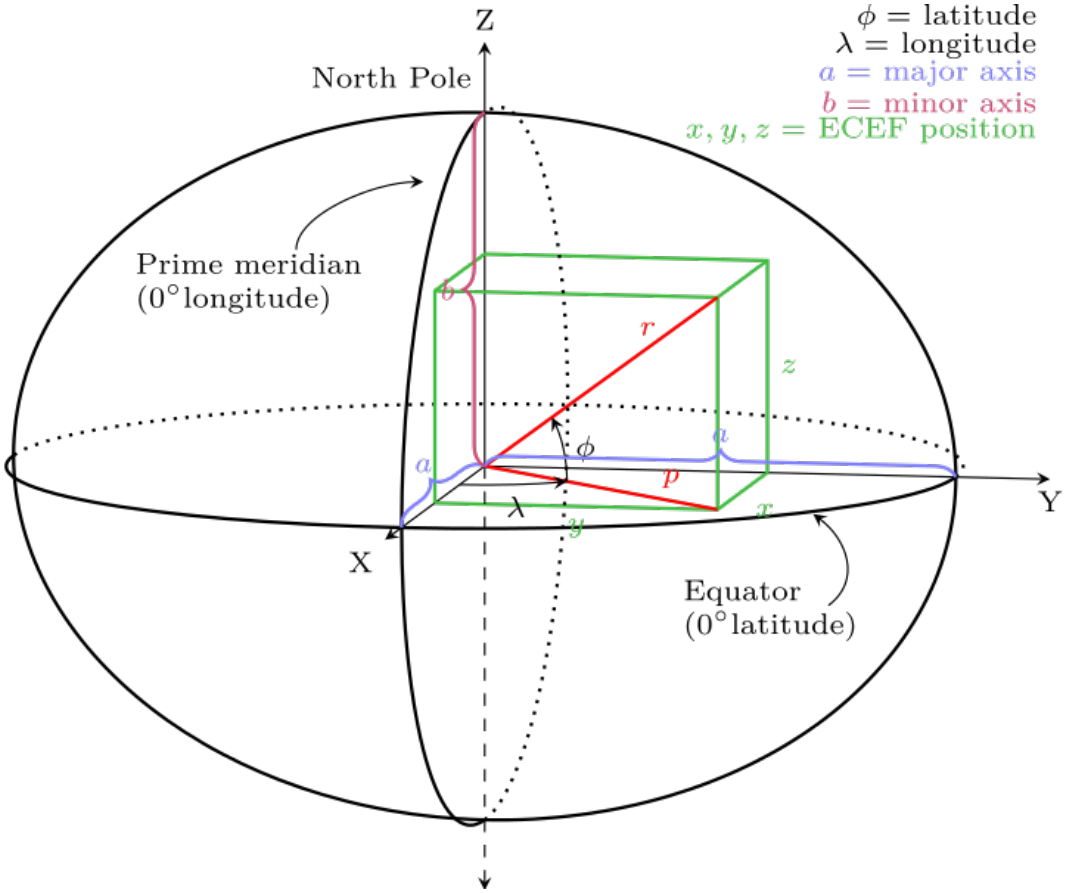
Finally, draw the `Placemark` that looks the same size regardless of the distance in perspective view.

3.6.1 Geographic Coordinate System

The Earth geographic coordinate system is a coordinate system that enables everywhere on the Earth to be specified by a set of numbers or symbols [38]. A common geodetic mapping coordinates are latitude, longitude, and altitude (LLA), which also is the raw location data decorated in KML file.

However, the LLA coordinates cannot be directly used from a program. Therefore, I introduce "earth-centered, earth-fixed" (ECEF) coordinate system for converting LLA coordinates to position coordinates. As we can see from Diagram 3.12, the origin of the axis (0, 0, 0) is located at the center of the Earth, the z-axis and y-axis are pointing towards the north and east, and the x-axis intersects the Earth at 0 latitude and 0 longitude.

FIGURE 3.12: Earth-centered, earth-fixed [37]



The ECEF coordinates are expressed in a reference system that is tended to be associated with geodetic mapping representations. In a system that practically requires high precision, an accurate method to approximate the Earth's shape is required. There have been researches on the elliptical earth since the 1980s. Nowadays, The new World Geodetic System was called WGS 84 which is currently being used by the Global Positioning System (GPS). It is geocentric

and globally consistent within $\pm$1m. The WGS 84 has a series of parameters (table 3.5) that define the shape of the ellipsoid (the Earth), they include a semi-major axis (a), a semi-minor axis (b) 3.13, its first eccentricity (e_1) and its second eccentricity (e_2).

FIGURE 3.13: Ellipsoid parameters

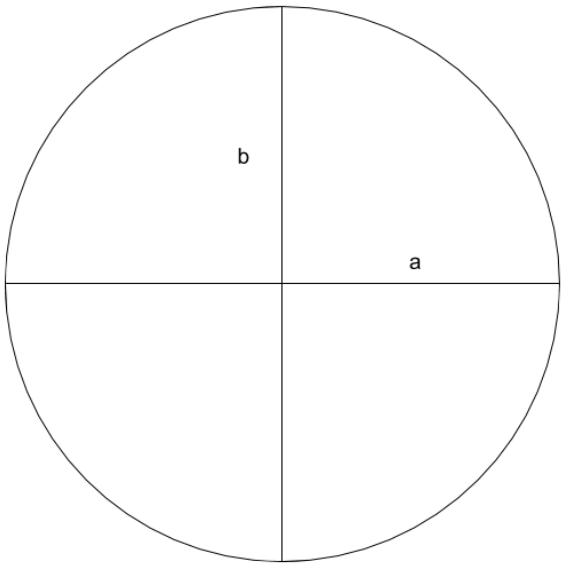
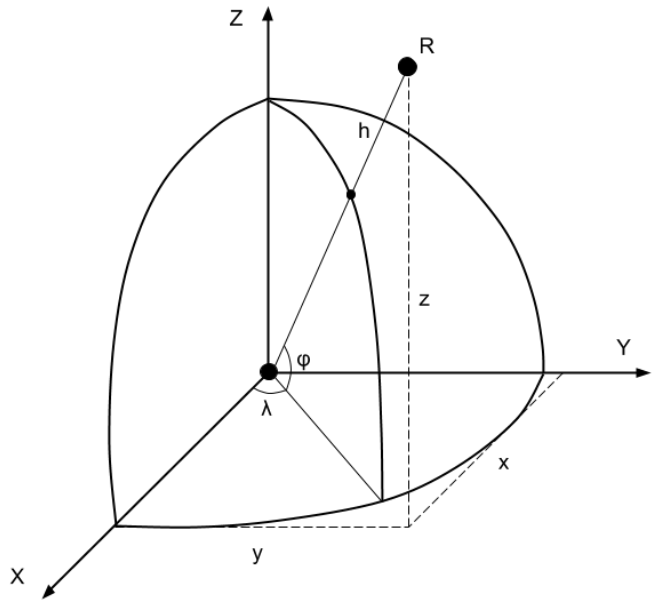


TABLE 3.5: WGS 84 parameters

Parameter	**Notation**	**Value**
Reciprocal of flattening	$\frac{1}{f}$	298.257 223 563
Semi-major axis	a	6 378 137 m
Semi-minor axis	b	$a\,(1-f)$
First eccentricity squared	e_1^2	$1-\frac{b^2}{a^2}=2\,f-f^2$
Second eccentricity squared	e_2^2	$\frac{a^2}{b^2}-1=\frac{f\,(2-f)}{(1-f)^2}$

In this application, where high accuracy is not required, simply using a equals to b (equals to the radius of the sphere). The conversion from LLA to ECEF as follow.

FIGURE 3.14: LLA to ECEF



$$x = (\mathbf{N} + \mathbf{h})\ \cos(\varphi)\ \cos(\lambda)$$
$$y = (\mathbf{N} + \mathbf{h})\ \cos(\varphi)\ \sin(\lambda)$$
$$z = (\frac{b^2}{a^2}\,\mathbf{N} + \mathbf{h})\ \sin(\varphi)$$

Where;

$$\varphi = \texttt{Latitude}$$
$$\lambda = \texttt{Longitude}$$
$$\mathbf{h} = \texttt{Altitude}$$
$$\mathbf{N} = \texttt{Radius of Curvature} = \frac{a}{\sqrt{1 - e^2\ \sin(\varphi)^2}}$$

The final transformation from the ECEF to program graphic coordinates. This is relevant to how texture mapping and graphic system be implemented.

`(x, y, z)` : y-east, z-north (up), x points to 0 latitude and 0 longitude.

⇓ Reversal x, and switch z and y.

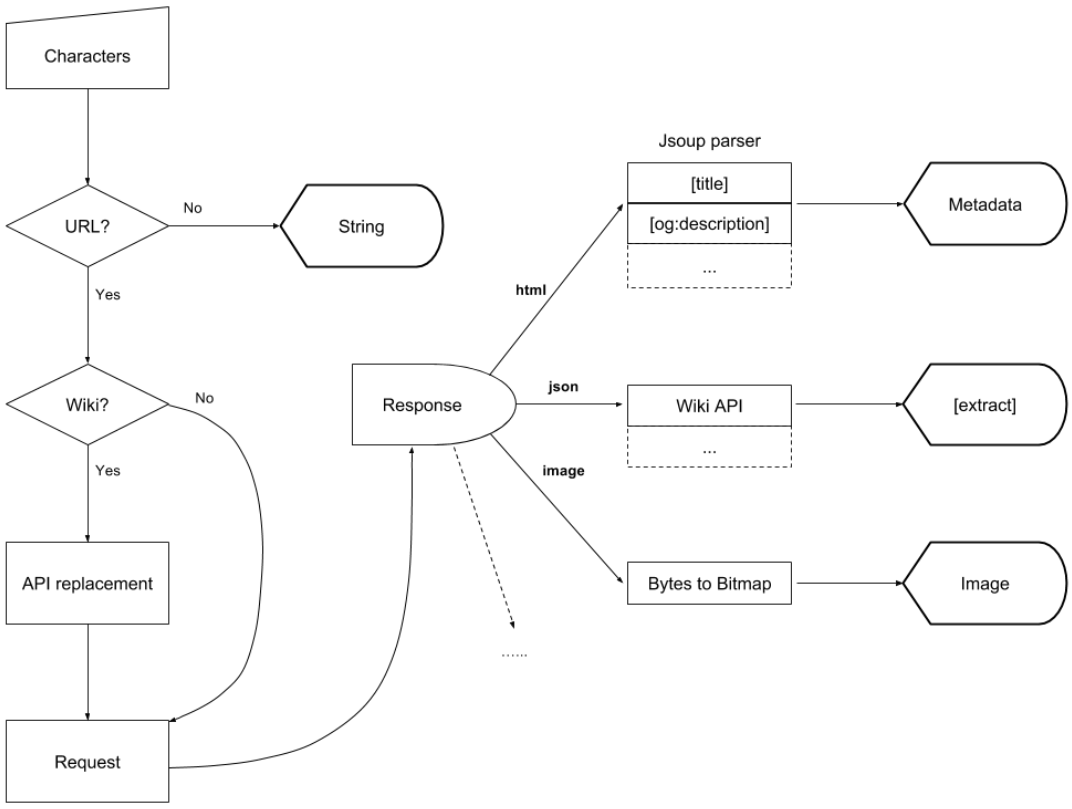
`(-x, z, y)` : x-east, y-north (up), z points to 0 latitude and 180 longitude.

3.6.2 Description

The raw description data of a `Placemark` decorated in KML file is a series of characters that could include any URL. Therefore, an analysis of the description is required before transforming for display. In this application, it currently supports following information display.

- **Plain text** Display the raw description data.
- **Image** Display the image from the URL.
- **Wikipedia** Display a extracted explanation on the topic.
- **HTML** Display `title` and `og:description` (one of the Open Graph metadata tags [28]) data (if it exists).

FIGURE 3.15: Description analysis



Treat the raw description data as plain text if it does not have a URL included. The `Content-Type` from the response of URL request, defines what kind of byte data is coming over, such as

`image/jpeg`, `application/json`, or `text/html`. I am using Jsoup (a Java library for working with real-world HTML [22]) for extracting target data from the HTML source. The most efficient way to get an extracted explanation or description on the topic from a Wikipedia page is to take use of the Opensearch API [36]. Therefore, a transformation from Wikipedia URL to Opensearch URL is required for an expected JSON response with `extract` tag included.

Given any valid Wikipedia page,

Replace `.wikipedia.org/wiki/`

⇓

To `.wikipedia.org/w/api.php?`**APIs**

Where **APIs** is:

```
 format=json
&action=query
&redirects=1
&prop=extracts
&exintro=
&explaintext=
&indexpageids=
&titles=
```

For instance,

```
https://en.wikipedia.org/wiki/Virtual_reality
                          ⇓
https://en.wikipedia.org/w/api.php?
                           format=json&action=query&redirects=1
                          &prop=extracts&exintro=&explaintext=
                          &indexpageids=&titles=Virtual_reality
```

3.6.3 Extra Model

A `Placemark` offers an ability to display a particular model that can be decorated as a URL in the `ExtendedData` (an element for adding custom data to a KML feature). In the sample data, there are some Wavefront OBJ models [43] are created by the Blender (free software) and being used in this application.

A simple OBJ parser is implemented, it yet supports a full OBJ features [29], such as syntax `mtllib` and `usemtl` are ignored. However, the main features that contain vertex related data for the model creation are supported, see Table 3.6.

TABLE 3.6: OBJ syntax

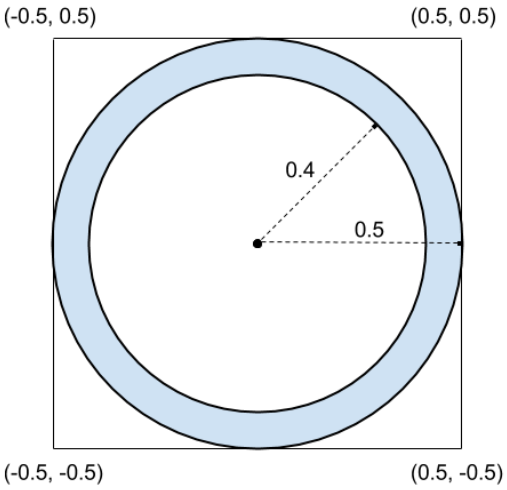
Starting character / word	**Meaning**
`v`	Geometric vertices
`vt`	Texture coordinates
`vn`	Vertex normals
`f`	Face, composed of `v` / `vt` (optional) / `vn` (optional)

3.7 Ray Pointer

A ray pointer is displayed in the center of sight for tracing the path of eyes. It is important to understand that the centering is a trick in the immersive virtual reality applications which works with glass lenses. There are different implementation on both eye's viewports to convert a world-space vertex into inverse-lens distorted screen space. As a result, the user can see a slightly overlapping effect on an object when it is not on focused. This is acting the same in reality, and a blurred overlapping effect is happening when people are focusing on different objects in the same direction. Therefore, the centering pointer for ray-tracing always locates at the surface of the distant objects. It removes unnecessary distraction and guarantees a clear focus on the object that user is looking at.

The ray pointer is drawn as `GL_POINTS`, but rounding to a ring shape (see Figure 3.16) in its Fragment Shader.

FIGURE 3.16: Ray Pointer ring




```
// Transform coord from range [0, 1] to [-0.5, 0.5]
vec2 coord = gl_PointCoord - vec2(0.5);
float length = length(coord);
if (0.4 < length && length < 0.5)
    FragColor = vec4(u_Color, 1.0);
```

3.8 Information Display

A flat plane 3D rectangular `Textfield` is implemented for displaying certain information, such as plain text and image. This information will be drawn as a texture and mapping to the `Textfield` vertex including the plain text. The calculation for the height of the total text with a certain pre-defined container width can be done by taking use of the Android native `android.text.StaticLayout`.

The `Textfield` is been used for presenting details of `Placemark`, and the KML chooser menu. The menu which contains multiple `Textfield` that are laid out on the top of a 3D rectangular `Panel` with a small vertical dimension.

The vital part of the implementation for flat rectangular objects is to estimate and give them a right rotation (in front of eyes with zero relative rotation angle) based on the users' 3D head pose. Therefore, a head poses related quaternion matrix [34] is needed to this end. I take the `head.quaternion (x, y, z, w)` provided by Google VR SDK from each frame, then convert the quaternion to a rotation matrix. First, reverse direction `(-x, -y, -z, w)` for facing to eye. Then, compute the rotation matrix by the given inhomogeneous expression [41].

$$R = \begin{bmatrix} 1 - 2\left(q_z^2 + q_w^2\right) & 2\left(q_y\, q_z - q_x\, q_w\right) & 2\left(q_x\, q_z + q_y\, q_w\right) \\ 2\left(q_y\, q_z + q_x\, q_w\right) & 1 - 2\left(q_y^2 + q_w^2\right) & 2\left(q_z\, q_w - q_x\, q_y\right) \\ 2\left(q_y\, q_w - q_x\, q_z\right) & 2\left(q_x\, q_y + q_z\, q_w\right) & 1 - 2\left(q_y^2 + q_z^2\right) \end{bmatrix} \tag{3.2}$$

3.9 Camera Movement

The ability to move around in virtual reality is important to satisfy user's need. Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions [15]. The motion sensors measure acceleration forces and rotational forces along three axes. They include accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

In general, the three physical axes (x, y, and z) data from Accelerometer sensor and Linear Acceleration sensor are useful to track and calculate device movement. Linear Acceleration is same as Accelerometer which measures the acceleration force in meter per second repeatedly, except the Linear Acceleration sensor is a synthetic sensor with gravity filtered out.

$$\texttt{LinearAcceleration} = \texttt{Accelerometer} - \texttt{Gravity}$$

$$v = \int a \cdot dt$$

$$x = \int v \cdot dt$$

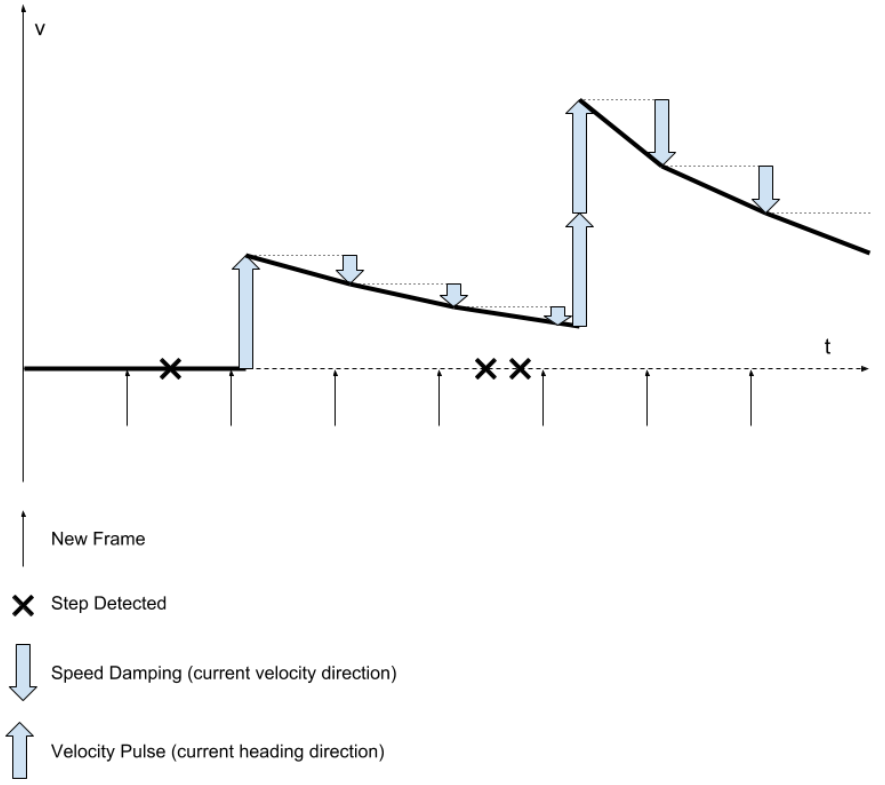
However, there is a technical limitation. First of all, we take the accelerometer data and remove gravity that is called gravity compensation, whatever is left is linear movement. Then we have to integrate it once to get velocity, and integrated again to get the position, which is called double integral. If the first integral creates drift, the double integrals are nasty that they create horrible drift. In such noise, using acceleration data for navigation is not accurate, and it is hard to do any kind of linear movement [7].

Therefore, I introduce Step Detector sensor and pedometer algorithm pedestrian navigation - user move forward on the current heading direction. First of all, during each frame life cycle, speed damping is calculated by a percentage stable and stop the camera in a certain of time regardless of current velocity. This is simply for avoiding that camera taking too long to stop.

Secondly, each detected step causes a constant velocity pulse in the heading direction, see Diagram 3.17.

$$p_1 = p_0 + v_0 \cdot dt$$
$$v_1 = v_0 + a \cdot dt$$

FIGURE 3.17: Camera movement



$$\vec{V_0} = \vec{V_0} \cdot \texttt{SpeedDamping}$$

$$\vec{P_1} = \vec{P_0} + \vec{V_0} \cdot dt$$

$$\vec{V_1} = \vec{V_0} + \overrightarrow{\texttt{HeadingDirection}} \cdot \texttt{Pulse} \cdot \texttt{DetectedStep}$$

$$\texttt{SpeedDamping} \in [0,\ 1]$$
$$\texttt{Pulse} \in [0,\ \infty)$$

3.10 Ray Intersection

In order to allow user interacts with virtual reality environment, the ability of intersection detection for ray-tracing is needed. In this way, it not only information can be easier present and understand, but also the user is able to interact (or select) objects in the 3D world.

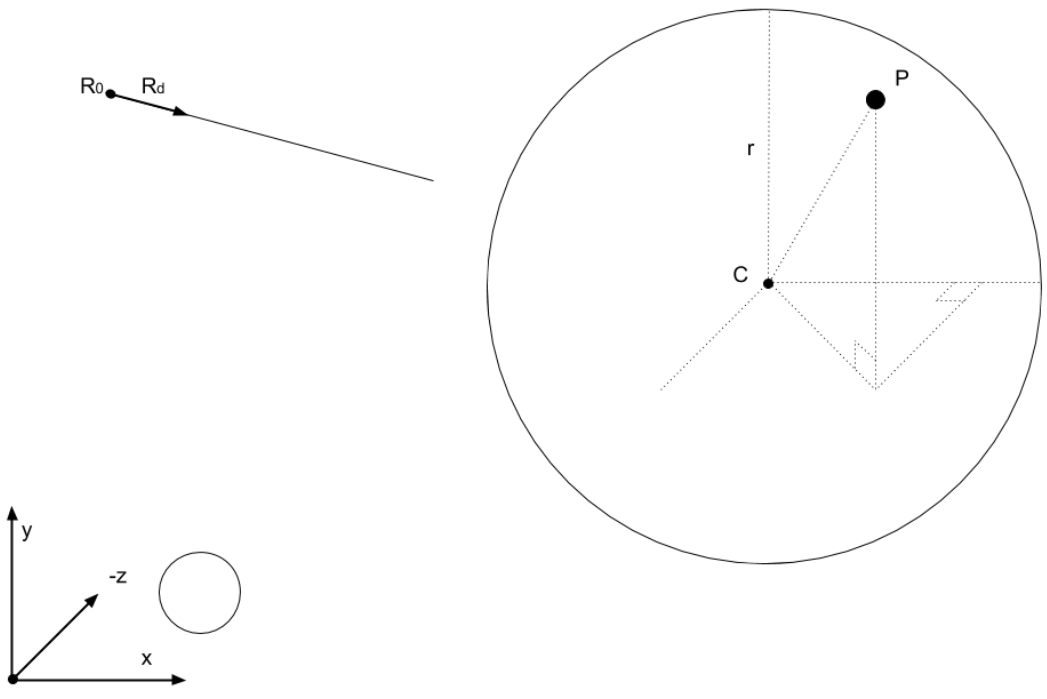
A ray can be described in an equation with known starting position, $\overrightarrow{R_0}$ and its direction $\overrightarrow{R_d}$:

$$\overrightarrow{R(t)} = \overrightarrow{R_0} + \overrightarrow{R_d} \cdot t \tag{3.3}$$

In this section, I separately reveal the intersection detection for basic elements. They are ray-sphere, ray-plane (3D rectangular), ray-box (2D and 3D).

3.10.1 Ray-Sphere

FIGURE 3.18: Ray-Sphere intersection



A point P on the surface of sphere can be described in an equation:

$$(x_p - x_c)^2 + (y_p - y_c)^2 + (z_p - z_c)^2 = r^2 \tag{3.4}$$

If the ray intersects with the sphere at any position P, it must match both equation 3.3 and 3.4. Therefore the solution of t from the cointegrate equation (as shown below) implies whether or not the ray will intersect with the sphere.

$$\begin{aligned}
(x_{R_0} + x_{R_d} \cdot t - x_c)^2 + (y_{R_0} + y_{R_d} \cdot t - y_c)^2 + (z_{R_0} + z_{R_d} \cdot t - z_c)^2 \quad &= r^2 \\
&\vdots \\
x_{R_d}^2\, t^2 + (2\, x_{R_d}\, (x_{R_0} - x_c))\, t + (x_{R_0}^2 - 2\, x_{R_0}\, x_c + x_c^2) & \\
+\, y_{R_d}^2\, t^2 + (2\, y_{R_d}\, (y_{R_0} - y_c))\, t + (y_{R_0}^2 - 2\, y_{R_0}\, y_c + y_c^2) & \\
+\, z_{R_d}^2\, t^2 + (2\, z_{R_d}\, (z_{R_0} - z_c))\, t + (z_{R_0}^2 - 2\, z_{R_0}\, z_c + z_c^2) \quad &= r^2
\end{aligned}$$

It is essentially can be seen as a quadratic formula:

$$a\, t^2 + b\, t + c = 0 \tag{3.5}$$

Where the solution of t are:

$$t = \begin{cases} \dfrac{-b \pm \sqrt{b^2 - 4\, a\, c}}{2\, a} & \text{if } \; b^2 - 4\, a\, c > 0 \\ \dfrac{-b}{2\, a} & \text{if } \; b^2 - 4\, a\, c = 0 \\ \varnothing & \text{if } \; b^2 - 4\, a\, c < 0 \end{cases} \tag{3.6}$$

A further step to get rid of formula complexity by taking in geometric vector calculation.

$\because$ Equation 3.4 and equation 3.5,

$$\begin{aligned}
a &= x_{R_d}^2 + y_{R_d}^2 + z_{R_d}^2 \\
b &= 2\, (x_{R_d}\, (x_{R_0} - x_c) + y_{R_d}\, (y_{R_0} - y_c) + z_{R_d}\, (z_{R_0} - z_c)) \\
c &= (x_{R_0} - x_c)^2 + (y_{R_0} - y_c)^2 + (z_{R_0} - z_c)^2 - r^2
\end{aligned}$$

& Geometric vector equation for $\overrightarrow{R_d}$ (ray's direction) and $\overrightarrow{V_{c\_}R_0}$ (vector from center of the sphere points to the ray's starting position):

$$|\overrightarrow{R_d}| \quad = \sqrt{x_{R_d}^2 + y_{R_d}^2 + z_{R_d}^2} = 1$$

$$\overrightarrow{V_{c\_R_0}} \quad = \overrightarrow{R_0} - \overrightarrow{C} = \overrightarrow{(x_{R_0} - x_c,\ \ y_{R_0} - y_c,\ \ z_{R_0} - z_c)}$$

$\therefore$ Value of a, b and c can be described as below:

$$a = 1$$

$$b = 2 \cdot \overrightarrow{R_d} \cdot \overrightarrow{V_{c\_R_0}}$$

$$c = \overrightarrow{V_{c\_R_0}} \cdot \overrightarrow{V_{c\_R_0}} \cdot r^2$$

& Introducing α and β to get rid of complexity for the equation 3.6 of t:

$$\frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a} = -\alpha \pm \sqrt{\beta}$$

Where:

$$\alpha = \frac{1}{2}\,b$$
$$\beta = \alpha^2 - c$$

$\therefore$ The solution formula for t can also be optimized as below:

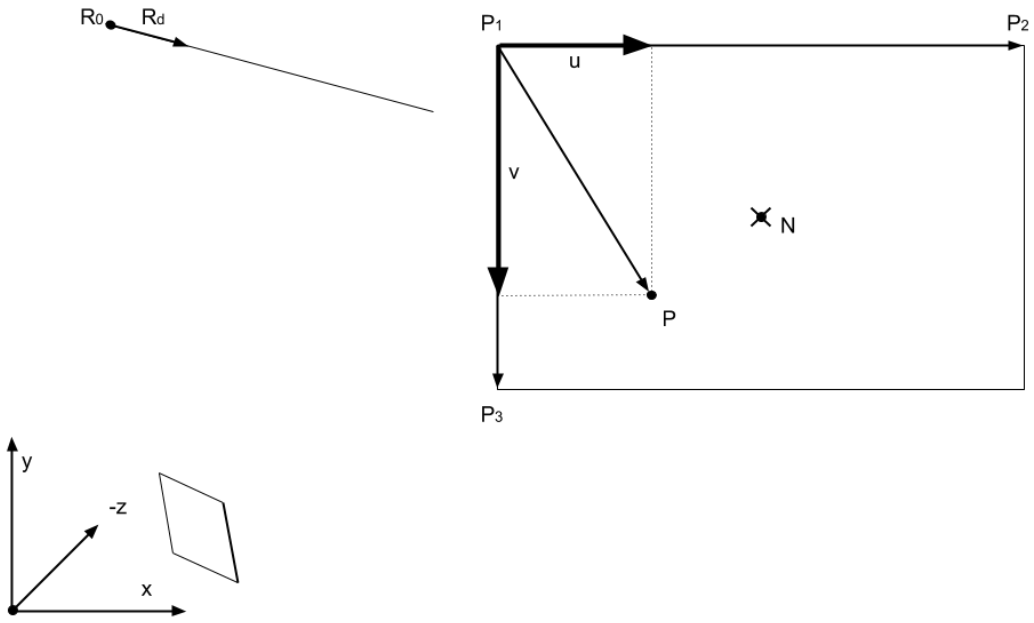
$$t = \begin{cases} -\alpha \pm \sqrt{\beta} & \text{if } \ \beta > 0 \\ -\alpha & \text{if } \ \beta = 0 \\ \varnothing & \text{if } \ \beta < 0 \end{cases}$$

$\therefore$ Given the known value of $\overrightarrow{R_0}$, $\overrightarrow{R_d}$ and $\overrightarrow{V_c}$, it is able to solve the t. The intersection position at any valid t can be obtained as follow:

$$\overrightarrow{P} = \overrightarrow{R_0} + \overrightarrow{R_d} \cdot t$$

3.10.2 Ray-Plane

FIGURE 3.19: Ray-Plane intersection



A point P on the plane which means it perpendicular to the $\overrightarrow{N}$ of the plane. If the P also belongs to the ray, then it can be described in a quadratic equation:

$$\begin{aligned} (\overrightarrow{P} - \overrightarrow{P_1}) \cdot \overrightarrow{N} = 0 \\ \overrightarrow{P} = \overrightarrow{R_0} + \overrightarrow{R_d} \cdot t \end{aligned} \tag{3.7}$$

$\therefore$ The solution for the t is:

$$t = \begin{cases} \dfrac{-\overrightarrow{N} \cdot (\overrightarrow{R_0} - \overrightarrow{P_1})}{\overrightarrow{N} \cdot \overrightarrow{R_d}} & \text{if } \overrightarrow{N} \cdot \overrightarrow{R_d} \nsim 0 \ \ (\text{or} > \texttt{EPSILON}) \\ \\ \varnothing & \text{if } \overrightarrow{N} \cdot \overrightarrow{R_d} \sim 0 \ \ (\text{or} < \texttt{EPSILON}) \end{cases}$$

After all, taking a valid t in equation 3.7 can only get the position $\overrightarrow{P}$ where the ray intersects

with the plane. Therefore, we have to verify whether or not the $\overrightarrow{P}$ belongs to a specific rectangular with certain width and height.

$\because$ Given vector projection equation of $\overrightarrow{A}$ on $\overrightarrow{B}$:

$$\begin{aligned} A_B &= |A| \cdot \cos(\theta) \\ A_B &= \frac{A \cdot B}{|B|} \end{aligned} \tag{3.8}$$

$\therefore$ The vector projection μ and ν of a valid $\overrightarrow{P}$ on both edges should greater than 0 but smaller than the length of the edge.

$$\mu = \frac{(\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1})}{|\overrightarrow{P_2} - \overrightarrow{P_1}|}$$

$$\nu = \frac{(\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1}))}{|\overrightarrow{P_3} - \overrightarrow{P_1}|}$$

$$\mu \in [0, \ |\overrightarrow{P_2} - \overrightarrow{P_1}|]$$
$$\nu \in [0, \ |\overrightarrow{P_3} - \overrightarrow{P_1}|]$$

Finally, transform them into a more optimized expression, and only if μ' and ν' satisfy the conditions below, the intersection position $\overrightarrow{P}$ is valid:

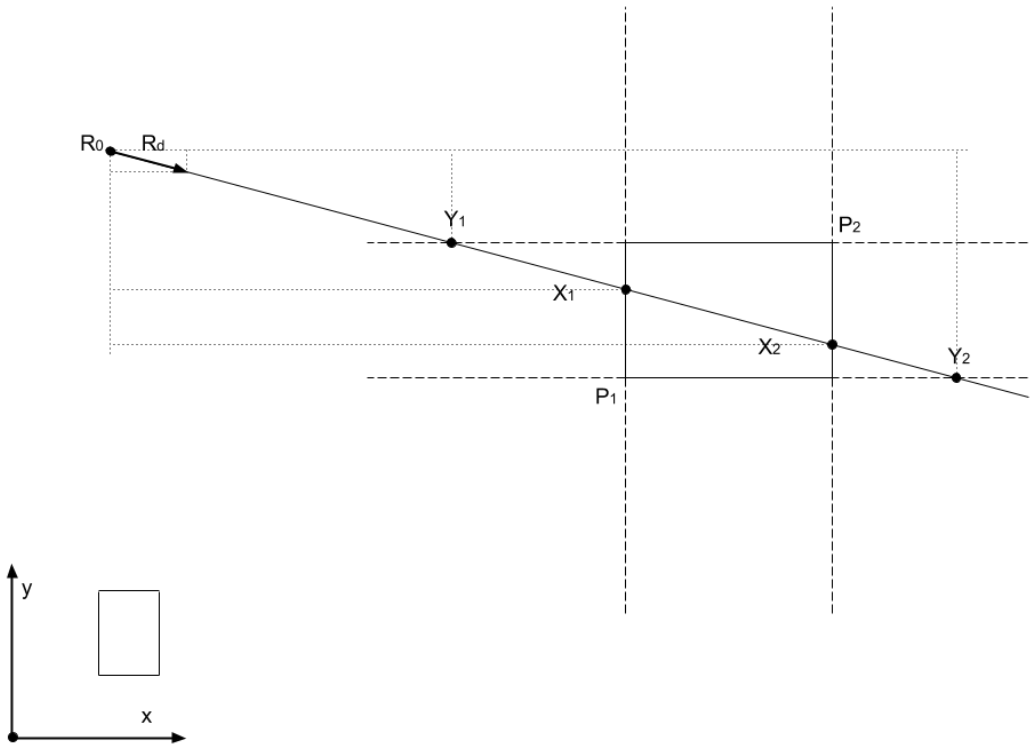
$$\mu' = (\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1})$$
$$\nu' = (\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1})$$
$$\mu' \in [0, \ (\overrightarrow{P_2} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1})]$$
$$\nu' \in [0, \ (\overrightarrow{P_3} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1})]$$

3.10.3 Ray-Box

See section 3.4.2, space partition is implemented in the 3D world that separates the space to invisible boxes that each box may or may not contain other objects. Ray-box intersection implementation for avoids unnecessary ray-object intersection tests. In this section, explanation divided into two parts, ray-box in 2D and which have inspired to ray-box 3D implementation.

Ray-Box 2D

FIGURE 3.20: Ray-Box 2D intersection



Given known R_0 (ray's starting position), R_d (ray's direction), P_1 (left-bottom corner), P_2 (right-top corner), we can get the value of X_1, X_2, Y_1 and Y_2. When ray intersecting with the 2D box, they have such meaning:

- **Vertical Area** Vertical route area between the left edge and right edge (include the area beyond the box)
- **Horizontal Area** Horizontal route area between the top edge and bottom edge (include the area beyond the box)
- $\mathbf{X}_1$ The distance from R_0 to the "in" point of Vertical Area in the x-axis direction.
- $\mathbf{X}_2$ The distance from R_0 to the "out" point of Vertical Area in the x-axis direction.
- $\mathbf{Y}_1$ The distance from R_0 to the "in" point of Horizontal Area in the y-axis direction.
- $\mathbf{Y}_2$ The distance from R_0 to the "out" point of Horizontal Area in the y-axis direction.

$\therefore$

$$X_1 = \begin{cases} x_{P_1} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_2} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad X_2 = \begin{cases} x_{P_2} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_1} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases}$$

$$Y_1 = \begin{cases} y_{P_1} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_2} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases} \qquad Y_2 = \begin{cases} y_{P_2} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_1} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

& The relative distance in x-axis and y-axis direction:

$$t_{X_1} = \frac{X_1}{x_{R_d}} \qquad t_{Y_1} = \frac{Y_1}{y_{R_d}}$$

$$t_{X_2} = \frac{X_2}{x_{R_d}} \qquad t_{Y_2} = \frac{Y_2}{y_{R_d}}$$

$\therefore$ A valid intersection exist only if following equation satisfied:

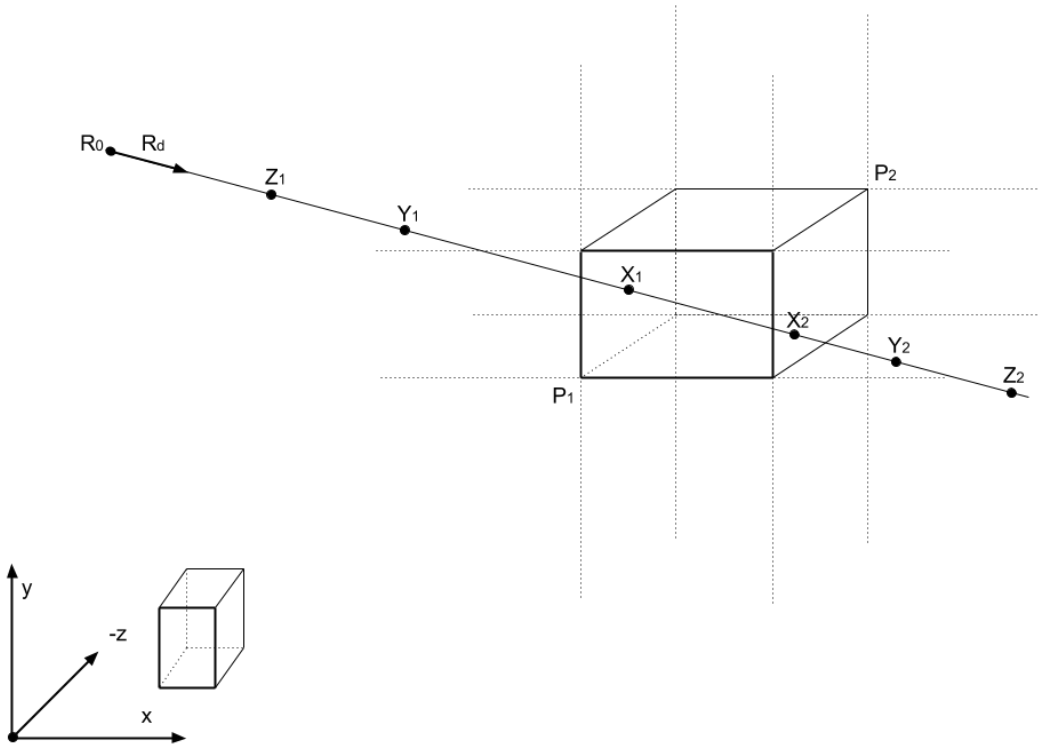
$$t_{X_1} < t_{X_2}$$
$$t_{Y_1} < t_{Y_2}$$

Which is:

$$\texttt{max}\left(t_{X_1},\ t_{Y_1}\right) < \texttt{min}\left(t_{X_2},\ t_{Y_2}\right)$$

Ray-Box 3D

FIGURE 3.21: Ray-Box 3D intersection



Given known R_0 (ray's starting position), R_d (ray's direction), P_1 (left-bottom-front corner), P_2 (right-top-back corner), we can get the value of X_1, X_2, Y_1, Y_2, Z_1 and Z_2. When ray intersecting with the 3D box, they have such meaning:

- **Vertical Area** Vertical route area within the scope of left-face, right-face, back-face and front face (include the area beyond the box)
- **Horizontal Area** Horizontal route area within the scope of top-face, bottom-face, back-face and front face (include the area beyond the box)
- **Z-depth Area** Z-depth route area within the scope of top-face, bottom-face, left-face and right face (include the area beyond the box)
- $\mathbf{X}_1$ The distance from R_0 to the "in" point of Vertical Area in the x-axis direction.
- $\mathbf{X}_2$ The distance from R_0 to the "out" point of Vertical Area in the x-axis direction.
- $\mathbf{Y}_1$ The distance from R_0 to the "in" point of Horizontal Area in the y-axis direction.
- $\mathbf{Y}_2$ The distance from R_0 to the "out" point of Horizontal Area in the y-axis direction.
- $\mathbf{Z}_1$ The distance from R_0 to the "in" point of Z-depth Area in the z-axis direction.
- $\mathbf{Z}_2$ The distance from R_0 to the "out" point of Z-depth Area in the z-axis direction.

$\therefore$

$$X_1 = \begin{cases} x_{P_1} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_2} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad X_2 = \begin{cases} x_{P_2} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_1} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases}$$

$$Y_1 = \begin{cases} y_{P_1} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_2} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases} \qquad Y_2 = \begin{cases} y_{P_2} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_1} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

$$Z_1 = \begin{cases} z_{P_1} - z_{R_0} & \text{if } z_{R_d} > 0 \\ z_{P_2} - z_{R_0} & \text{if } z_{R_d} < 0 \end{cases} \qquad Z_2 = \begin{cases} z_{P_2} - z_{R_0} & \text{if } z_{R_d} > 0 \\ z_{P_1} - z_{R_0} & \text{if } z_{R_d} < 0 \end{cases}$$

& The relative distance in different axis direction:

$$t_{X_1} = \frac{X_1}{x_{R_d}} \qquad t_{Y_1} = \frac{Y_1}{y_{R_d}} \qquad t_{Z_1} = \frac{Z_1}{z_{R_d}}$$

$$t_{X_2} = \frac{X_2}{x_{R_d}} \qquad t_{Y_2} = \frac{Y_2}{y_{R_d}} \qquad t_{Z_2} = \frac{Z_2}{z_{R_d}}$$

$\therefore$ A valid intersection exist only if following equation satisfied:

$$t_{X_1} < t_{X_2}$$
$$t_{Y_1} < t_{Y_2}$$
$$t_{Z_1} < t_{Z_2}$$

$\therefore$ Which is:

$$\texttt{max}\,(t_{X_1},\ t_{Y_1},\ t_{Z_1}) < \texttt{min}\,(t_{X_2},\ t_{Y_2},\ t_{Z_2}) \tag{3.9}$$

4 Discussion

Our approach to explore geographic data visualization with immersive virtual reality is to actually develop one. We take use of Google Cardboard for turning an Android phone into an immersive virtual reality device. A virtual reality-specific application is developed to import KML format geographic data source and display them in the application. The user is able to make a six degrees of freedom (DOF) - position coordinates (x, y and z offsets) and orientation (yaw, pitch and roll angles) - movement. User is able to do simple interactions: selecting a placemark; viewing the information of the placemark on a popup message board; displaying a customized 3D model (eg: OBJ model) of the placemark; or any further information from a URL (which could be a image or a piece of summarized information extracted from Wikipedia or any HTML text).

By comparison to virtual globes, geographic data visualization with immersive virtual reality device allows to do similar things, but it has more easy used intuitive interface. Firstly, they can share with geographic data that created by a universal markup language (eg: KML), which means almost every data based feature in the existing virtual globe can also migration to the immersive virtual reality application. Secondly, they are both able to have a remote server database that provides synchronous data, such as a server processes the requests and delivers the result in a standard web format back to the client.

There are five human senses provide the information and passed to our brain for capturing our attention: sight (70%), hearing (20%), smell (5%), touch (4%), and taste (1%) [23]. The immersive virtual reality has certainly improved the feedback of sight sense, and also by given the existing Spatial Audio technology (such as [16]), it is able to use a spatial audio as a simultaneous response from the user for "fooling" the hearing sense.

Sensor fusion creates a huge drift during the nasty double integration process, we alternatively using the Step sensor (pedometer) as the pedestrian navigation (it is not the most logical way). It allows move forward in the current heading direction. Nonetheless, it doing very well for navigating through all scene that satisfies our application purpose.

The performance of this immersive virtual reality application is good (55 - 60 FPS) when the there is less than 250 placemarks exist in the scene. Although, we have an Octree based object intersection algorithm avoid most of the invalid recursive detection, and optimized matrix reconstitution, but there is a performance limitation of actual OpenGL ES native call in the Android SDK. To solve this issues require further investigation for reducing the times of OpenGL

ES render call for each frame.

Also, there is a limitation of gesture recognition and perception technology that suppress the development of immersive virtual reality technology.

Due to the time, and geographic data resource limitation, this project is simply developed with some unfinished features, which are very important as a geographic visualization tool, but it not particularly critical for an exploring purpose.

A key requirement for environmental scientists is to be able to visualize four-dimensional data (i.e. time-dependent three-dimensional data). Indeed, we are able to visualize the environment data from the data file which created from the different period of time. We also can do a fake real-time data visualization by a certain frequently refreshing rate on both client and server, but there is a limitation on both client (performance) and server (data creation), none of them make any sense. However, an implementation of dynamic graphic animation would be excellent for improving user understanding of any environmental data visualization. That is to say, an animation transform from one piece of time-dependent data visualization to another.

One of the main features of geographic data visualization is the Level Of Detail (LOD) rendering based on distance from the viewpoint. Textures of the virtual reality environment should be separately prepared, and attached as the circumstances may require. They are updatable and detailed on different levels. It can also provide a solution for visualizing a large amount of overlapping data.

Most of the geographic data markup language (eg: KML) supports multiple layers in a single file so that we need a layer switch, and more geometric shapes supporting, such as lines and polygons. Under the LOD implementation, we are able to see the architectural structure or plan if we are close enough.

There is always room for improvement, especially in the immersive virtual reality visualization, when something related to human intuitive nature system, because the feeling will be always not real enough compare to the real world interaction.

5 Conclusion

The immersive virtual reality provides a highly integrated easy-to-use, intuitive real-time 3D GIS for geographic data visualization. Due to the limitation of human-machine interaction, the VR is not yet able to do everything that the pseudo-3D virtual globes can do, but it has the potential to do more than people expected when there is a revolution for gesture recognition and perception.

There are at least three reasons indicate that Android Phones are extremely suitable to use as an immersive virtual reality device. First, always about the money, 15$ Google Cardboard kit turns Android or iOS smartphone to immersive virtual reality device; second, the existing VR specific open source SDK provided by Google, includes necessary graphic, and spatial audio development; third, Android includes support for high-performance graphics with OpenGL.

The most logical way to calculate the movement in the immersive virtual reality environment is to use Gyroscope to measures angular velocity relative to the body, or in other words, to get the device orientation. Then, using Accelerometer to inject the correction term that keeps the orientation correct with respect to gravity, and a correction due to the magnetic north from Compasses is also required. However, it is really hard to get an accurate position out of them due to a horrible drift comes from the nasty double integration process. Alternatively, a pedestrian navigation was implemented for this project is based on the Step sensor (pedometer) with a certain algorithm to calculate the velocity was turn out working very well. The limitation of this approach is can only move forward in the current heading direction.

KML is not only a human-readable markup language can and very suit for visualizing geographic data, but also it has very well compatibility with current major virtual globes, such as the well-known Google Earth. Moreover it also powerful enough to describe a small subregion, such as a building hierarchical plan. On the other hand, immersive virtual reality also can be used as a tool that able to visualizing different sort of data, or natural system by integrating another or new spatial markup language.

A Source

Related source repository:

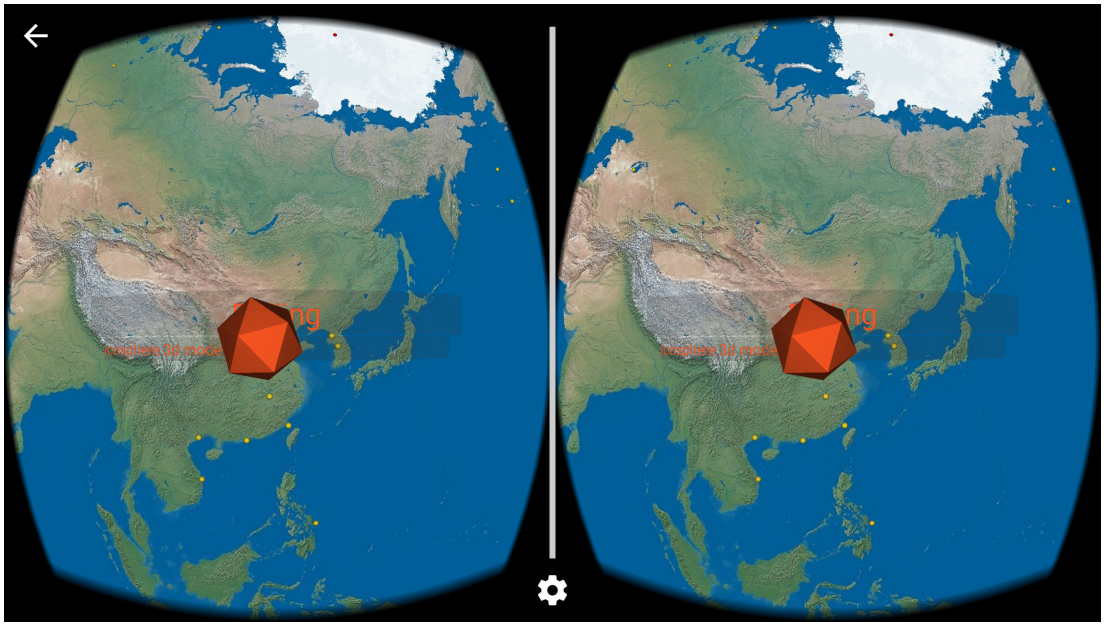
https://github.com/jiangyang5157/virtual-reality

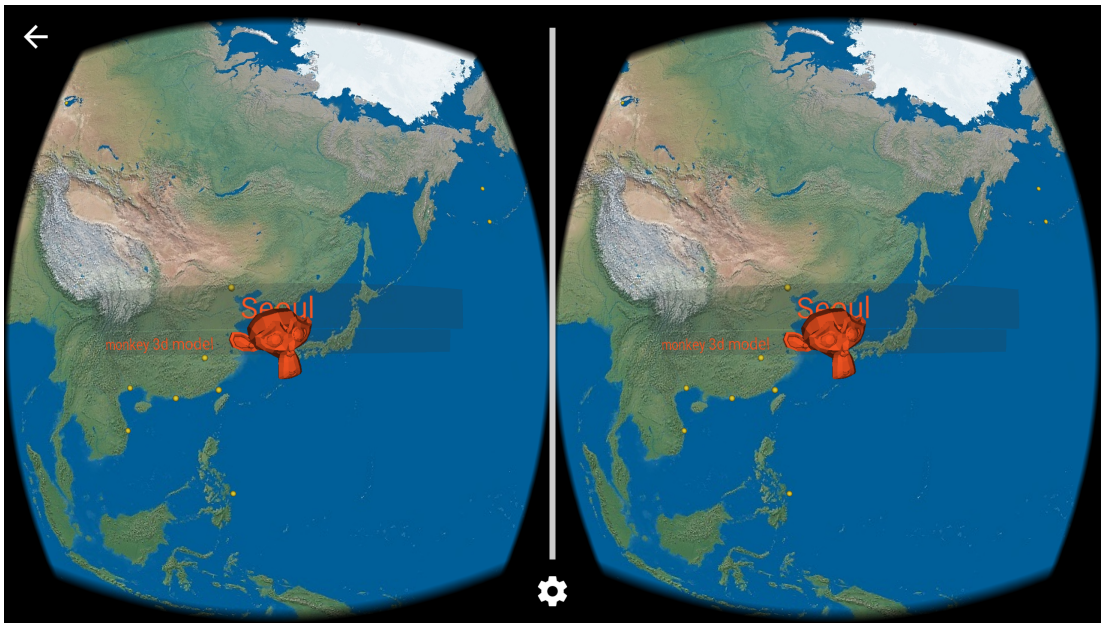
https://github.com/jiangyang5157/tookit

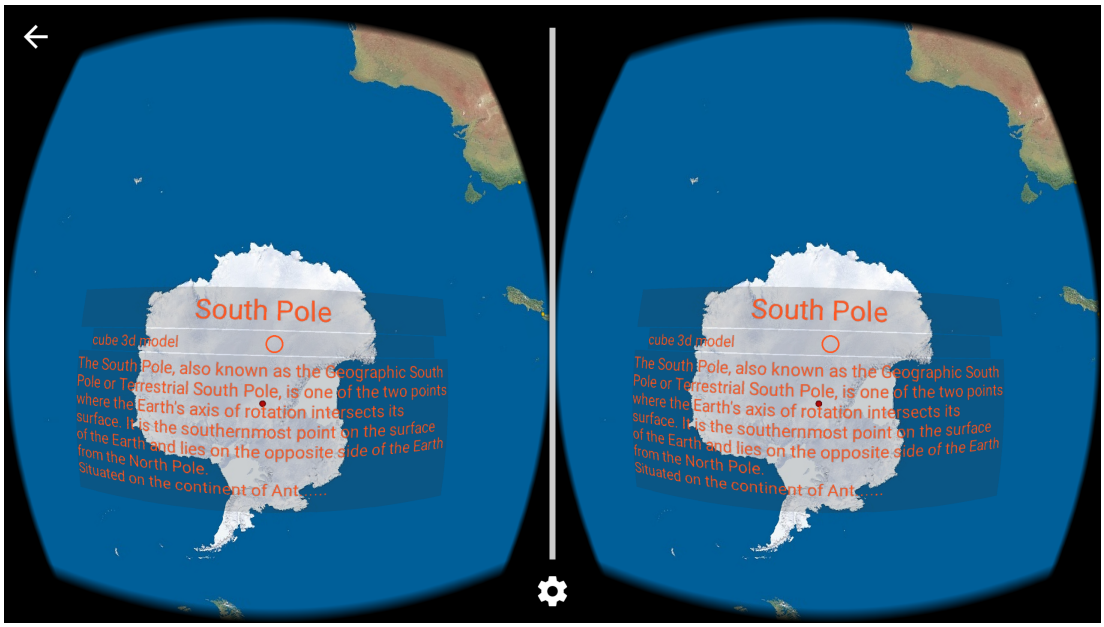
https://github.com/jiangyang5157/vr-server

https://github.com/jiangyang5157/massey-master-thesis-2016

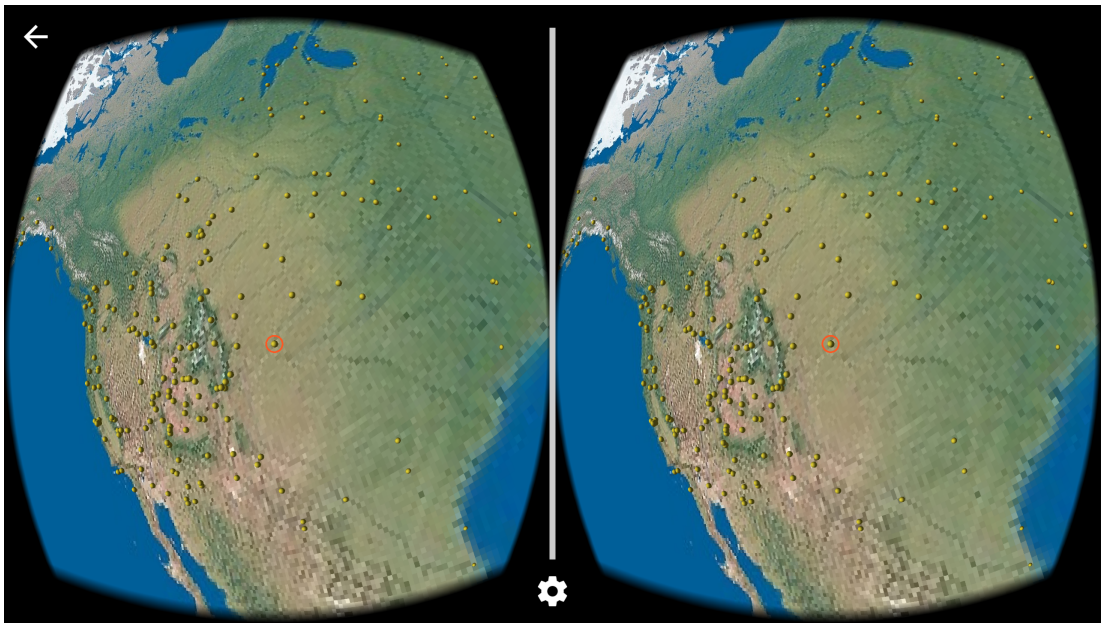
B Screenshots

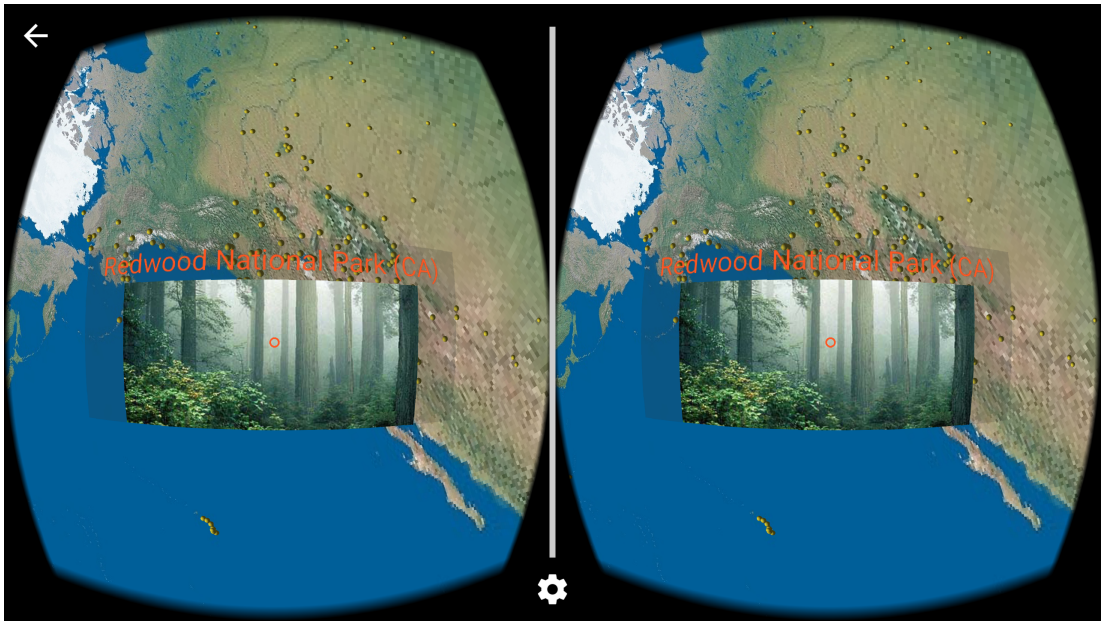




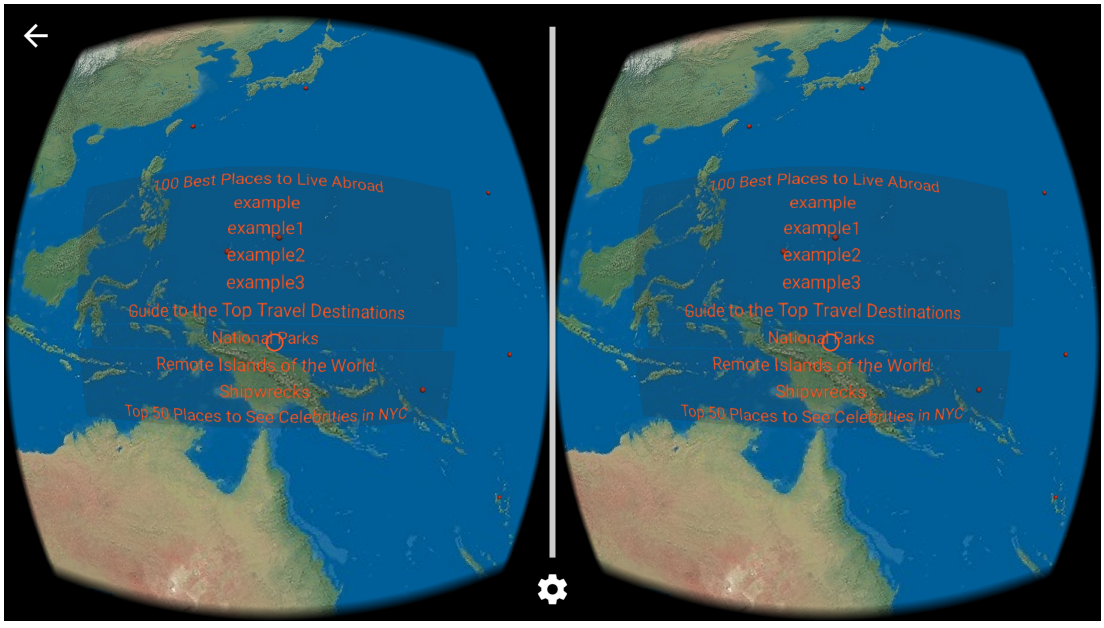

South Pole
cube 3d model
The South Pole, also known as the Geographic South Pole or Terrestrial South Pole, is one of the two points where the Earth's axis of rotation intersects its surface. It is the southernmost point on the surface of the Earth and lies on the opposite side of the Earth from the North Pole.
Situated on the continent of Ant...
South Pole
cube 3d model
The South Pole, also known as the Geographic South Pole or Terrestrial South Pole, is one of the two points where the Earth's axis of rotation intersects its surface. It is the southernmost point on the surface of the Earth and lies on the opposite side of the Earth from the North Pole.
Situated on the continent of Ant...


National Park of American Samoa (AS)
National Park of American Samoa (AS)




Redwood National Park (CA)
Redwood National Park (CA)


Sydney Australia
Sydney Australia


100 Best Places to Live Abroad
example
example1
example2
example3
Guide to the Top Travel Destinations
National Parks
Remote Islands of the World
Shipwrecks
Top 50 Places to See Celebrities in NYC
100 Best Places to Live Abroad
example
example1
example2
example3
Guide to the Top Travel Destinations
National Parks
Remote Islands of the World
Shipwrecks
Top 50 Places to See Celebrities in NYC

Bibliography

[1] T. Barnes. (2011). Fast, branchless ray/bounding box intersections, [Online]. Available: https://tavianator.com/fast-branchless-raybounding-box-intersections.

[2] J. D. Blower, A. Gemmell, K. Haines, P. Kirsch, N. Cunningham, A. Fleming, and R. Lowry, "Sharing and visualizing environmental data using virtual globes", 2007.

[3] D. Butler, "Virtual globes: The web-wide world", *Nature*, vol. 439, no. 7078, pp. 776–778, 2006.

[4] T. Danova. (2015). The global smartphone market report, [Online]. Available: http://www.businessinsider.com.au/global-smartphone-market-forecast-vendor-platform-growth-2015-6?r=US&IR=T.

[5] Earthslot. (2016). Earth maps & world geography, [Online]. Available: http://www.earthslot.org/.

[6] Esri. (2016). Explorer for arcgis, [Online]. Available: http://www.esri.com/software/arcgis/explorer.

[7] Google. (2010). Sensor fusion on android devices: A revolution in motion processing, [Online]. Available: https://www.youtube.com/watch?v=C7JQ7Rpwn2k&feature=youtu.be&t=23m21s.

[8] ——, (2012). Go at google: Language design in the service of software engineering, [Online]. Available: https://talks.golang.org/2012/splash.article.

[9] ——, (2016). Aar format, [Online]. Available: http://tools.android.com/tech-docs/new-build-system/aar-format.

[10] ——, (2016). Android maps utils, [Online]. Available: https://github.com/googlemaps/android-maps-utils/tree/master/library/src/com/google/maps/android/kml.

[11] ——, (2016). Google cardboard, [Online]. Available: https://vr.google.com/cardboard/.

[12] ——, (2016). Google vr sdk for android, [Online]. Available: https://developers.google.com/vr/android/.

[13] ——, (2016). Keyhole markup language, [Online]. Available: https://developers.google.com/kml/.

[14] ——, (2016). Opengl es, [Online]. Available: https://developer.android.com/guide/topics/graphics/opengl.html.

[15] ——, (2016). Sensors overview, [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html.

[16] ——, (2016). Spatial audio, [Online]. Available: https://developers.google.com/vr/concepts/spatial-audio.

[17] ——, (2016). The go programming language, [Online]. Available: https://golang.org/.

[18] ——, (2016). Transmitting network data using volley, [Online]. Available: https://developer.android.com/training/volley/index.html.

[19] B. Huang and H. Lin, "A java/cgi approach to developing a geographic virtual reality toolkit on the internet", *Computers & Geosciences*, vol. 28, no. 1, pp. 13–19, 2002.

[20] IDC. (2016). Smartphone os market share, [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[21] A. Imbert. (2016). Go json rest, [Online]. Available: https://github.com/ant0ine/go-json-rest.

[22] Jsoup. (2016). Jsoup: Java html parser, [Online]. Available: https://jsoup.org/.

[23] T. Mazuryk and M. Gervautz, "Virtual reality-history, applications, technology and future", 1996.

[24] NASA. (2016). Nasa world wind, [Online]. Available: https://worldwind.arc.nasa.gov/.

[25] I. Nourbakhsh, R. Sargent, A. Wright, K. Cramer, B. McClendon, and M. Jones, "Mapping disaster zones", *Nature*, vol. 439, no. 7078, pp. 787–788, 2006.

[26] A. Nuernberger. (2006). Virtual globes in the classroom, [Online]. Available: http://www.csiss.org/SPACE/resources/virtual-globes.php#GISdata.

[27] OGC. (2016). Open geospatial consortium, [Online]. Available: http://www.opengeospatial.org/.

[28] OGP. (2014). The open graph protocol, [Online]. Available: http://ogp.me/.

[29] Paulbourke. (). Object files, [Online]. Available: http://paulbourke.net/dataformats/obj/.

[30] T. M. Rhyne, "Going virtual with geographic information and scientific visualization", *Computers & Geosciences*, vol. 23, no. 4, pp. 489–491, 1997.

[31] T. M. Rhyne, W. Ivey, L. Knapp, P. Kochevar, and T. Mace, "Visualization and geographic information system integration: What are the needs and the requirements, if any?", in *Visualization, 1994., Visualization'94, Proceedings., IEEE Conference on*, IEEE, 1994, pp. 400–403.

[32] B. T. Tuttle, S. Anderson, and R. Huff, "Virtual globes: An overview of their history, uses, and future challenges", *Geography Compass*, vol. 2, no. 5, pp. 1478–1505, 2008.

[33] User3146587. (2014). 3d ray-quad intersection, [Online]. Available: http://stackoverflow.com/questions/21114796/3d-ray-quad-intersection-test-in-java.

[34] J. V. Verth. (2013). Understanding quaternions, [Online]. Available: http://www.essentialmath.com/GDC2013/GDC13_quaternions_final.pdf.

[35] Wikipedia. (2006). Icosahedron golden rectangles, [Online]. Available: https://commons.wikimedia.org/wiki/File:Icosahedron-golden-rectangles.svg.

[36] ——, (2016). Api, [Online]. Available: https://www.mediawiki.org/wiki/API:Opensearch.

[37] ——, (2016). Ecef, [Online]. Available: https://en.wikipedia.org/wiki/ECEF.

[38] ——, (2016). Geographic coordinate system, [Online]. Available: https://en.wikipedia.org/wiki/Geographic_coordinate_system.

[39] ——, (2016). Geographic information system, [Online]. Available: https://en.wikipedia.org/wiki/Geographic_information_system.

[40] ——, (2016). Opengl shading language, [Online]. Available: https://en.wikipedia.org/wiki/OpenGL_Shading_Language.

[41] ——, (2016). Rotation matrices, [Online]. Available: https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles.

[42] ——, (2016). Vrml, [Online]. Available: https://en.wikipedia.org/wiki/VRML.

[43] ——, (2016). Wavefront obj file, [Online]. Available: https://en.wikipedia.org/wiki/Wavefront_.obj_file.

[44] ——, (2016). World wide web, [Online]. Available: https://en.wikipedia.org/wiki/World_Wide_Web.

[45] ——, (2016). X3d, [Online]. Available: https://en.wikipedia.org/wiki/X3D.