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

November 7, 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 virtual reality 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) [20], 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 [24]. 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 [20].

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 was 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 markup language; geodetic-mapping coordinates; performance of 3D graphic system. In this thesis, firstly, a background of geographic data visualization is presented. Then, details of the related technology and implementation are described. Then, application performance testing results are illustrated. Finally, is discussion and conclusion, simply summarizing the paper.

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 [40]. A GIS combines a database management system and a graphic display system that tie to the process of spatial analysis [31]. 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 [32]. 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 [33]. 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. [33].

- **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 [27]. It also helps scientific collaboration research, such as the EarthSLOT [4]. Moreover, Butler

points out virtual globes can be used as an invaluable tool in disaster response [2], [26]. 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 [1].

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 [28]. 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 [28] 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 [1] 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 [25] imports data from tile servers, OGC web services and limited support for KML, it has more focus toward scientific users; ArcGIS Explorer [5] 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) [43] that is a language for describing 3D objects and interactive scenes on the World-Wide Web (WWW) [45], It has been superseded by X3D [46].

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 [24].

2 Technology

As an immersive virtual reality application, there are few things need to consider. A device can be utilized and efficiently perform development tasks, including graphic system and network support; A markup language for decorating geographic data. There are some major technologies listed in this chapter.

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: Smartphone shipments forecase [3]

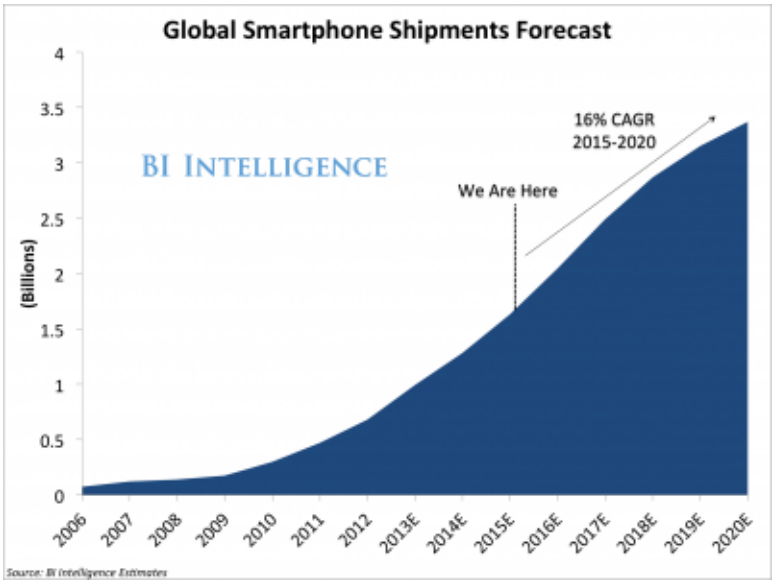
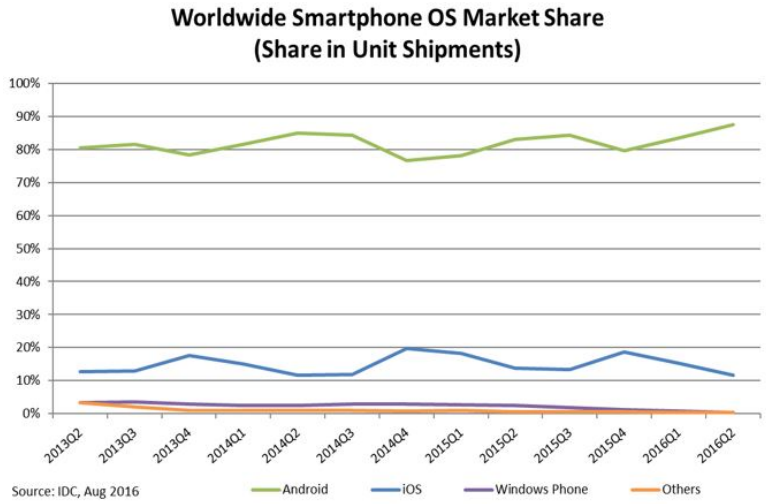


FIGURE 2.2: Smartphone OS market share [21]



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

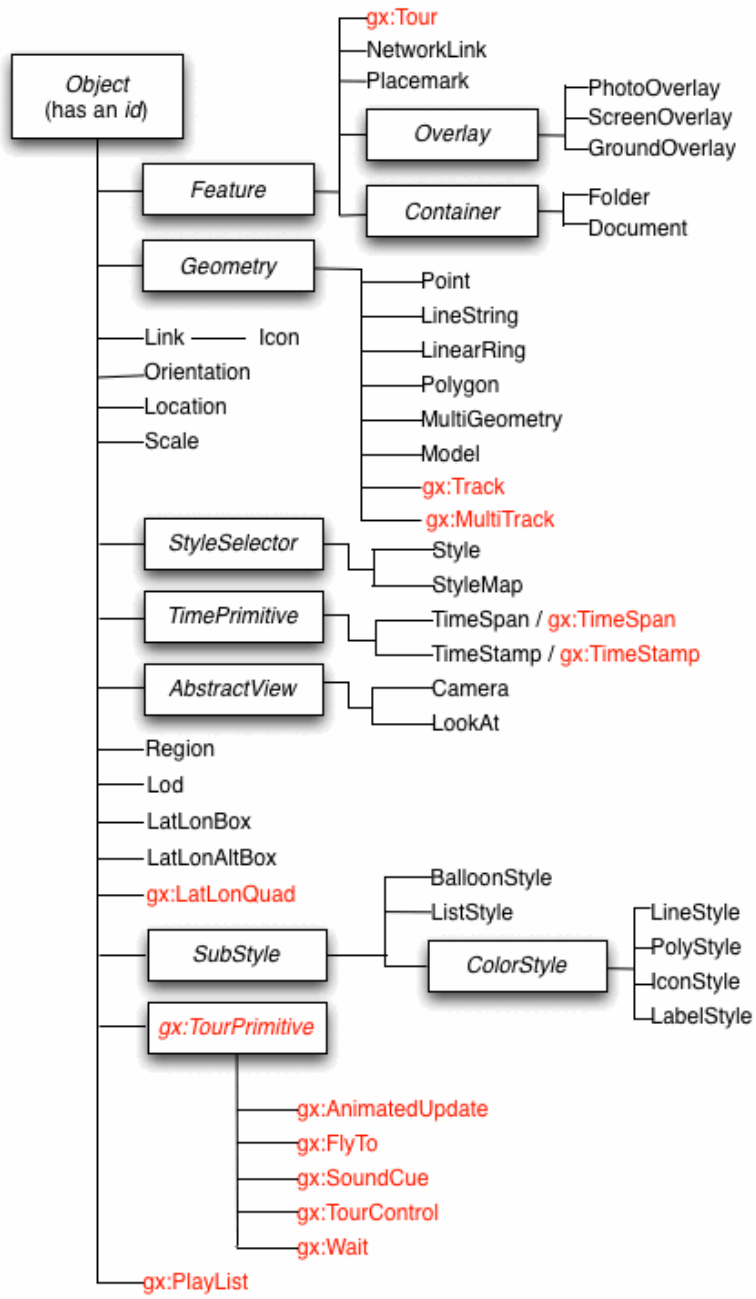
OpenGL ES Version	**Android Version**
OpenGL ES 1.0	Android 1.0 and higher
OpenGL ES 1.1	Android 1.0 and higher
OpenGL ES 2.0	Android 2.2 (API level 8) and higher
OpenGL ES 3.0	Android 4.3 (API level 18) and higher
OpenGL ES 3.1	Android 5.0 (API level 21) and higher
OpenGL ES 3.2 (September 2016)	Android 7.0 (API level 24) and higher

2.3 Geographic Visualization Markup Language

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. KML has Network links supported (`NetworkLink` is a KML facility of many 2.3), which gives the power to serve content from a local or remote location. It generally used to distribute data to large numbers of clients. In another word, if the data requires an update, it has to be changed at the source location (remote server) and all users receive the updated data automatically.

As the markup language, more important than the satisfaction of needs is that the KML has been supported by many virtual globes and other GIS systems. It is becoming a de facto standard [1] that can be manipulated in other software if required.

FIGURE 2.3: KML schema [13]



2.4 Network

Real-time data are very important in the environmental sciences [1]. 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 [18]) 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 [7]. 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 was selected for developing the server.

On the client side (Android platform), Volley is being used for transmitting network data. It is an open sourced HTTP library that makes networking for Android apps easier and most importantly, faster [19]. Also, application is taking use of Jsoup (Java HTML Parser [23]) 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 Android OpenGL ES support. Then an explanation of the web server design. After that, is the creation of 3D scene and the implementation of device movement. Finally, the ray-model intersection detection is highlighted.

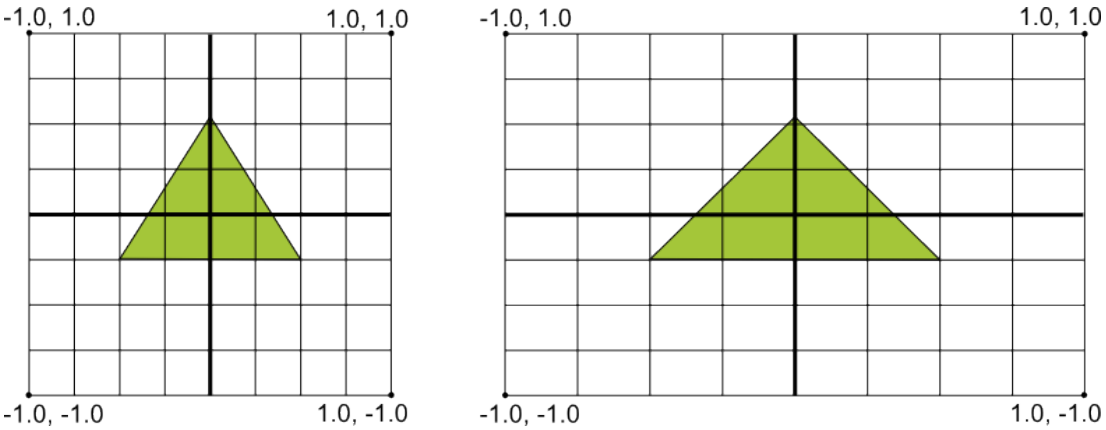
3.1 Google VR SDK

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. Firstly, working out the model matrix, view matrix, and perspective matrix in CPU (Android programming). Secondly, passing them to GPU for the rest of calculation (OpenGL Shading Language Programming, i.e. GLSL or GLslang), such as projected vertex, lighting, or coordinates transformation to different shape. 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 [41].

TABLE 3.1: OpenGL compute

What	**How**	**Where**
Model Matrix	translationM * scaleM * rotationM * identityM(1)	CPU
Camera Matrix	lookAt(positionV, lookAtV, upV)	CPU
View Matrix	eye.viewM * cameraM	CPU
Perspective Matrix	eye.perspective(zNear, zFar)	CPU
ModelView Matrix	viewM * modelM	GPU
Projection Matrix	perspectiveM * modelViewM	GPU
Vertex'	projectionM * vertex	GPU

3.3 Web Server

Setup a file server by Golang is efficient and convenient. See the following example: 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 API support[1], I introduce a free framework Go-Json-Rest [22], 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 particular file back to the client. Table 3.2 list 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

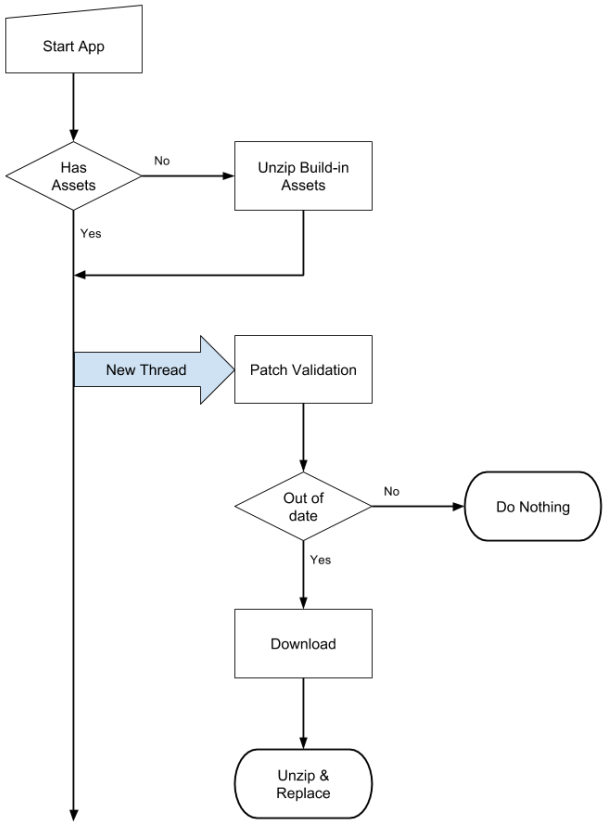
3.3.2 Patch

Patch is for the server to guarantee the latest data (if any) will be pushed to all client applications. It server as a compressed ZIP file, and it contains one or more files for client to update. Patch validation is happening whenever the app starts. First, client sent request to file server for retrieving the patch file $http://xxx.xxx.xxx.xxx:8080/assets/static.zip$. 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

[1] **At this stage, the RESTful API has not been actually used, the only setup for the purpose of testing.**.

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 Application binary) will be uncompressed to avoid no available data. Diagram 3.2 illustrates 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.

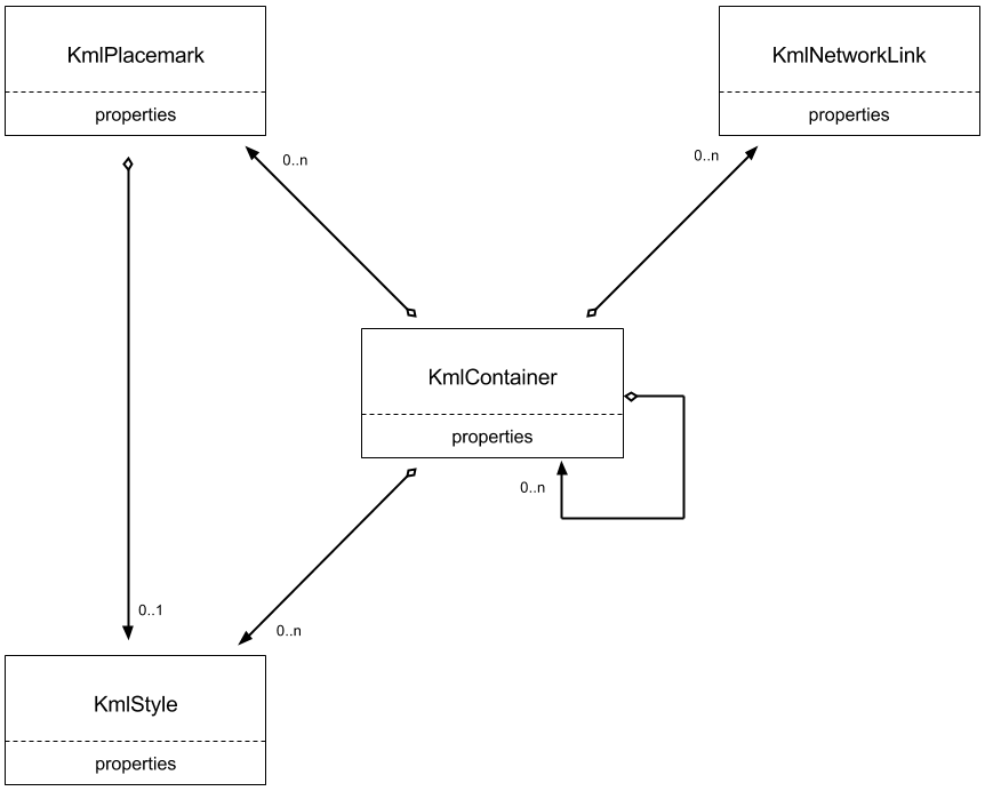
As you can see from Table 3.2, there are two assets folders contains KML files - "*/assets/static/layer*" and "*/assets/static/kml*". These files are literally the same, but existing in different concepts for achieving the purpose of categorizing. By making use of `Networklink` facility, an individual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

Space partition divides the space with certain patterns. It has many advantages, such as run-time graphical analysis, optimized intersection and collision detection.

3.4.1 Geographic Visualization Markup Language

Only some of KML features from KML schema 2.3 are be used in the 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 which is one of the unsupported features in the library. Details of KML syntax can be found in its reference [13].

FIGURE 3.3: kML parser simple



3.4.2 Space Partition

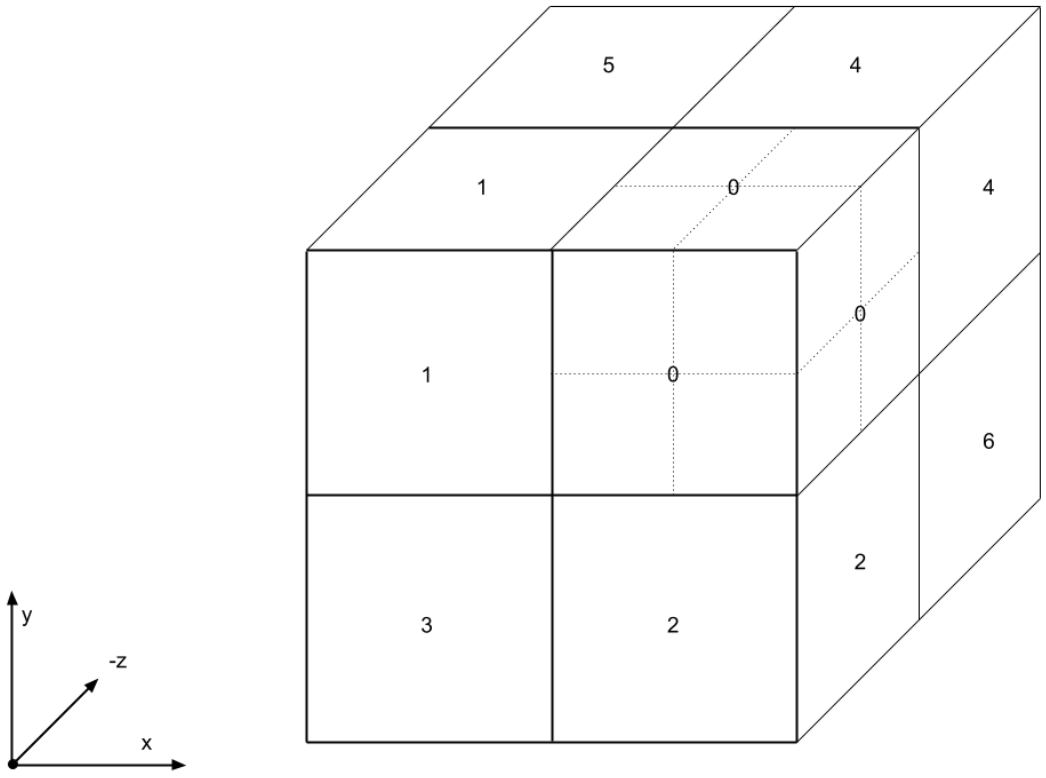
Space partition often used for optimizing collision detection algorithms among polygonal models. These algorithms are often expensive operations and can significantly slow FPS down. Although there is no collision detection in this application yet, there is an intersection detection between the ray tracing (see 3.7) and other objects. Space partition is contributed to reducing

the ray-model intersection test load by skipping invalid 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 3.4. It has a predefined constant positive integer to decide whether or not a new partitioning should happen. This number is important for the purpose of reducing intersection detections, and it indicates a minimum number of objects allowed exist in the same cell. In view of the object complexity (ray-placemark) and the cell complexity (ray-box), the number 5 was adopted, and it got a significant performance improvement (see 4.2.2).

If the number is positive infinity, whole space 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-model, 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.

FIGURE 3.4: Octree division



See Diagram 3.4, 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.

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

TABLE 3.3: Octree octant

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

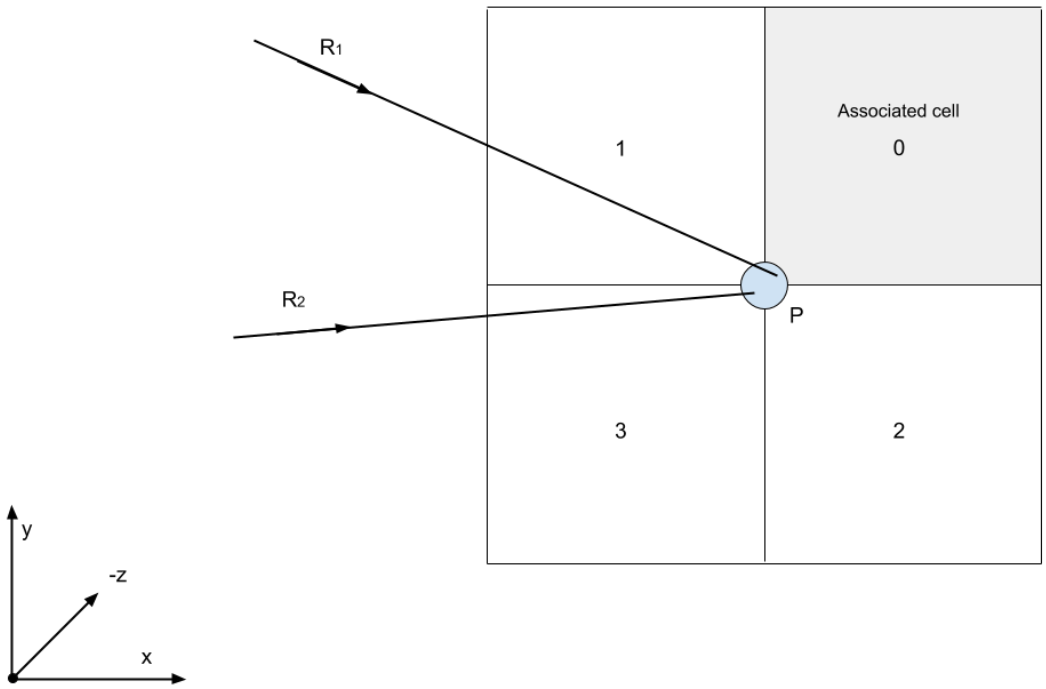
∴ The conversion between among delta, index and octant:

```
private boolean[] getOctant(OcTreeObject obj){
  boolean[] octant = new boolean[3];
  for (int i = 0; i < 3; i++) {
    float delta = obj.center[i] - this.center[i];
    octant[i] = delta >= 0;
  }
  return octant;
}

private boolean[] getOctant(int index) {
  return new boolean[]{
      (index & 1) == 0, // 0, 2, 4, 6
      (index & 2) == 0, // 0, 1, 4, 5
      (index & 4) == 0, // 0, 1, 2, 3
  };
}

private int getIndex(boolean[] octant) {
  int ret = 0;
```

```
  for (int i = 0; i < 3; i++) {
    if (!octant[i]) {
      ret |= (1 << i);
    }
  }
  return ret;
}
```

When the `Placemark` is literally land on the axis, the associated cell does not conform to the fact. The initialization of boolean `octant[]` value is based on three axes' delta values. However, the implementation is designed to make a `Placemark` only has one associated cell. As we can see from 3.5, ray R_1 and R_2 are both intersecting with the `Placemark`, but the intersection from ray R_2 will be considered as invalid due to it did not intersect with the associated cell 0 first.

FIGURE 3.5: Octree associated cell

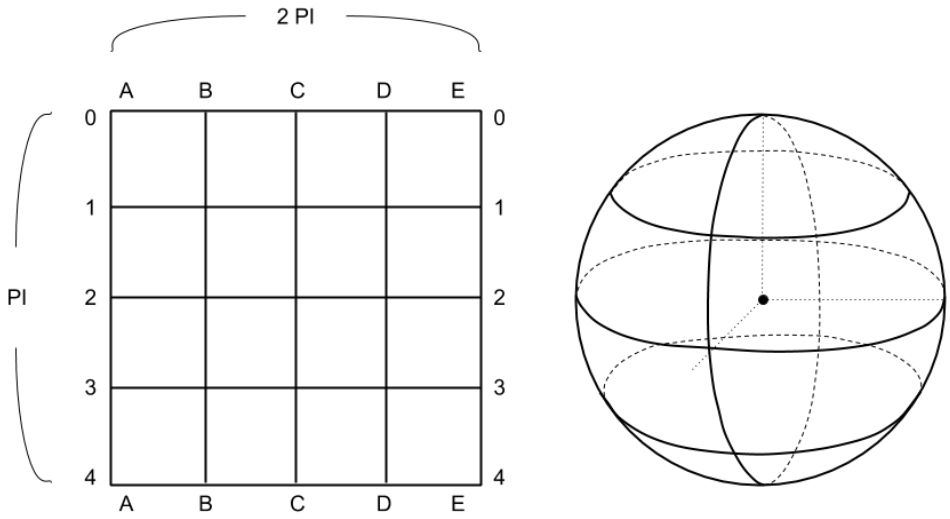


3.5 Earth

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 3.6: UV sphere mapping

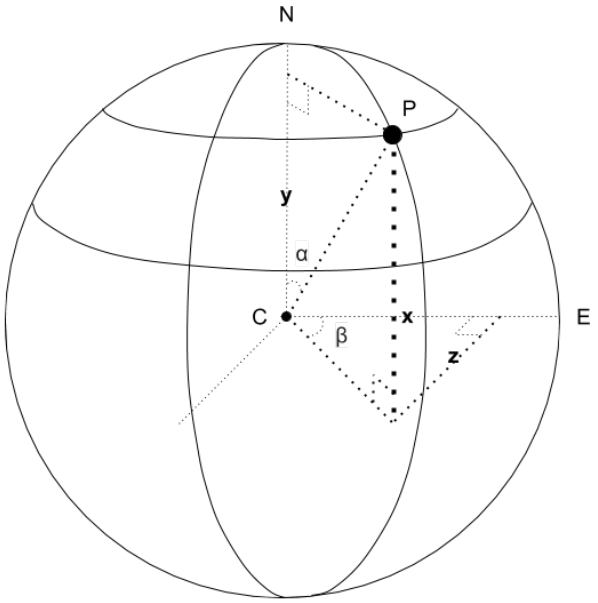


The Diagram 3.6 illustrates the mapping from 2D plane to 3D UV sphere's surface which has 5 rings and 4 segments. As we can seen, vertex $A0$, $A1$, $A2$, $A3$, $A4$ and $E0$, $E1$, $E2$, $E3$, $E4$ are duplicated; vertex $A0$, $B0$, $C0$, $D0$, $E0$ converge together in the pole, as well as $A4$, $B4$, $C4$, $D4$, $E4$. 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 3.7: 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:

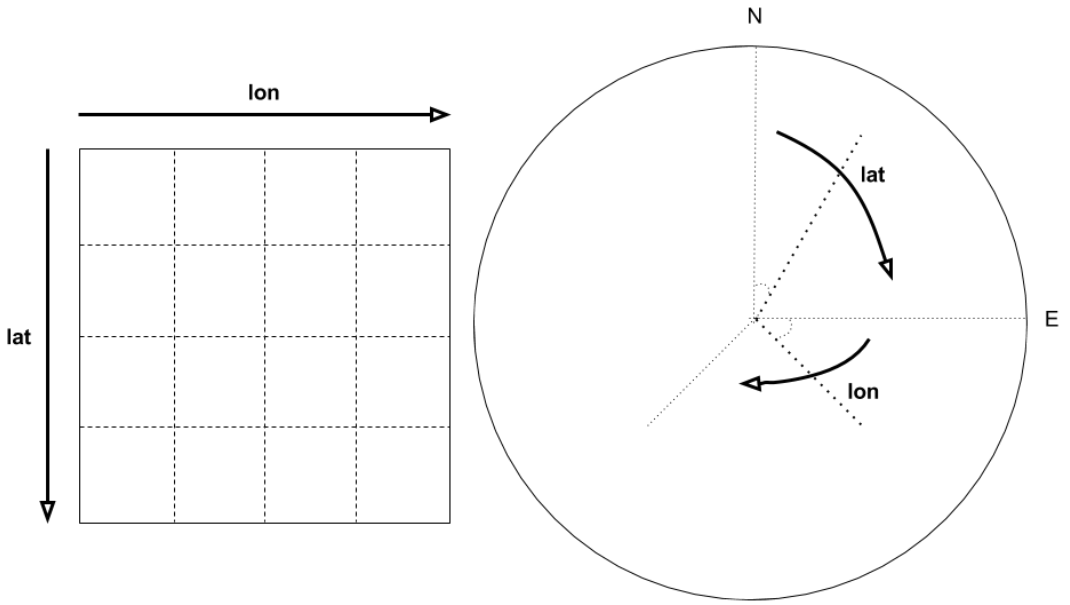
$$x = (\sin(\alpha) \times \texttt{radius}) \times \cos(\beta)$$
$$y = \cos(\alpha) \times \texttt{radius}$$
$$z = (\sin(\alpha) \times \texttt{radius}) \times \sin(\beta)$$

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

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

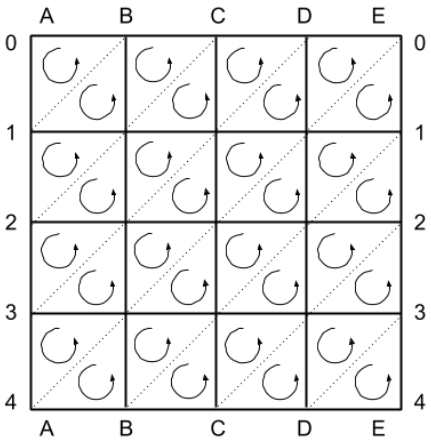
It is vital to recognize the flip-side effect caused by the processing order of texturing. Diagram 3.8 shows a flip on the longitude direction. Which is not looking correct from outside of the Earth, but it is a precise mapping when the user is viewing from the inside.

FIGURE 3.8: 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 3.9.

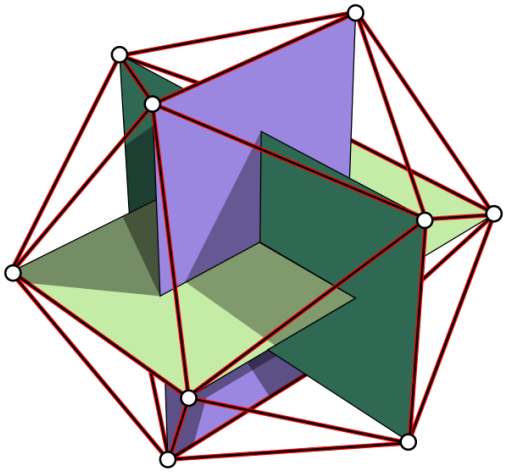
FIGURE 3.9: UV sphere CCW



3.6 Placemark

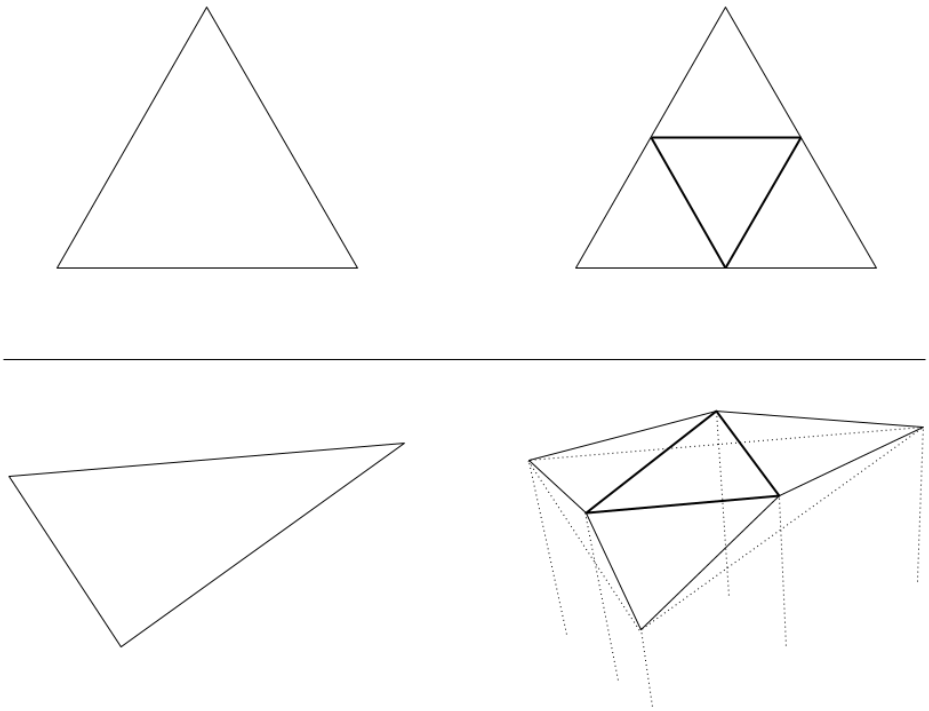
Vertex generation for `Placemark` is a recurring process of subdividing icosphere. Figure 3.10 is an icosahedron, the corners of three orthogonal rectangles are the initial vertices for `Placemark`.

FIGURE 3.10: Icosahedron rectangles [35]



Rounding the icosphere by subdividing a face to an arbitrary level of resolution **??**. Each face can be subdivided into four by connecting each edge's midpoint, then push the midpoints to the surface of the sphere 3.11.

FIGURE 3.11: Icosphere refinement



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).

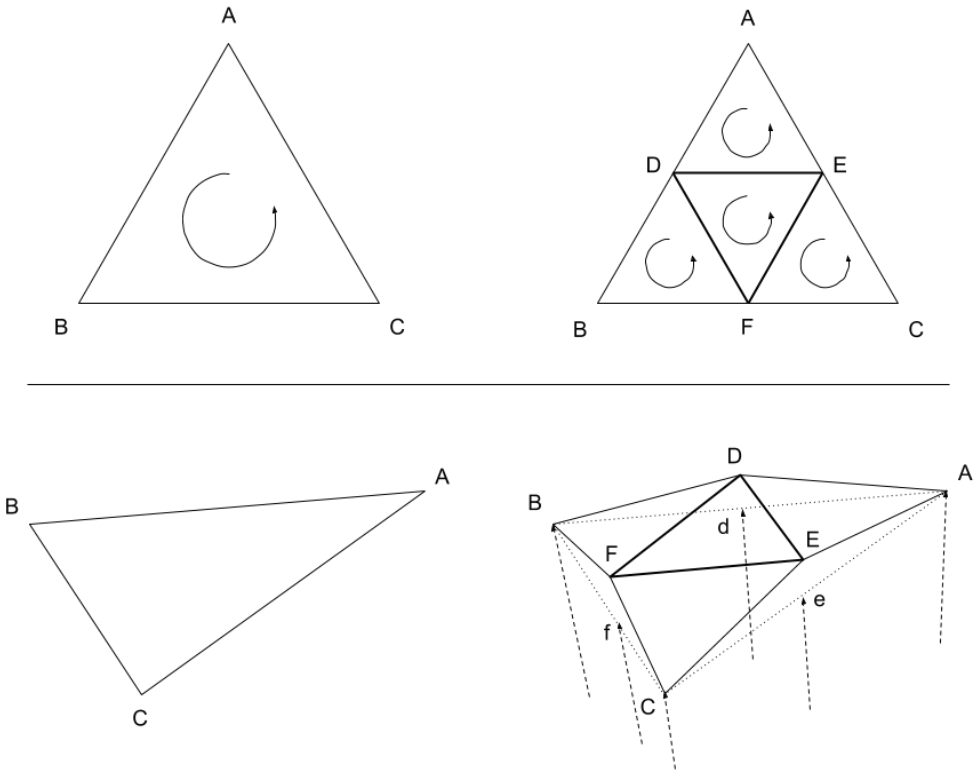
$$\overrightarrow{D} = \texttt{normalize}(\overrightarrow{d})$$

$$\overrightarrow{E} = \texttt{normalize}(\overrightarrow{e})$$

$$\overrightarrow{F} = \texttt{normalize}(\overrightarrow{f})$$

Third, Also, adjustment of triangle's index array is required. Remove $\triangle$ABC in the vertex indexes list, add new $\triangle$ADE, $\triangle$DBF, $\triangle$EFC, and $\triangle$DEF.

FIGURE 3.12: Icosphere vertex index

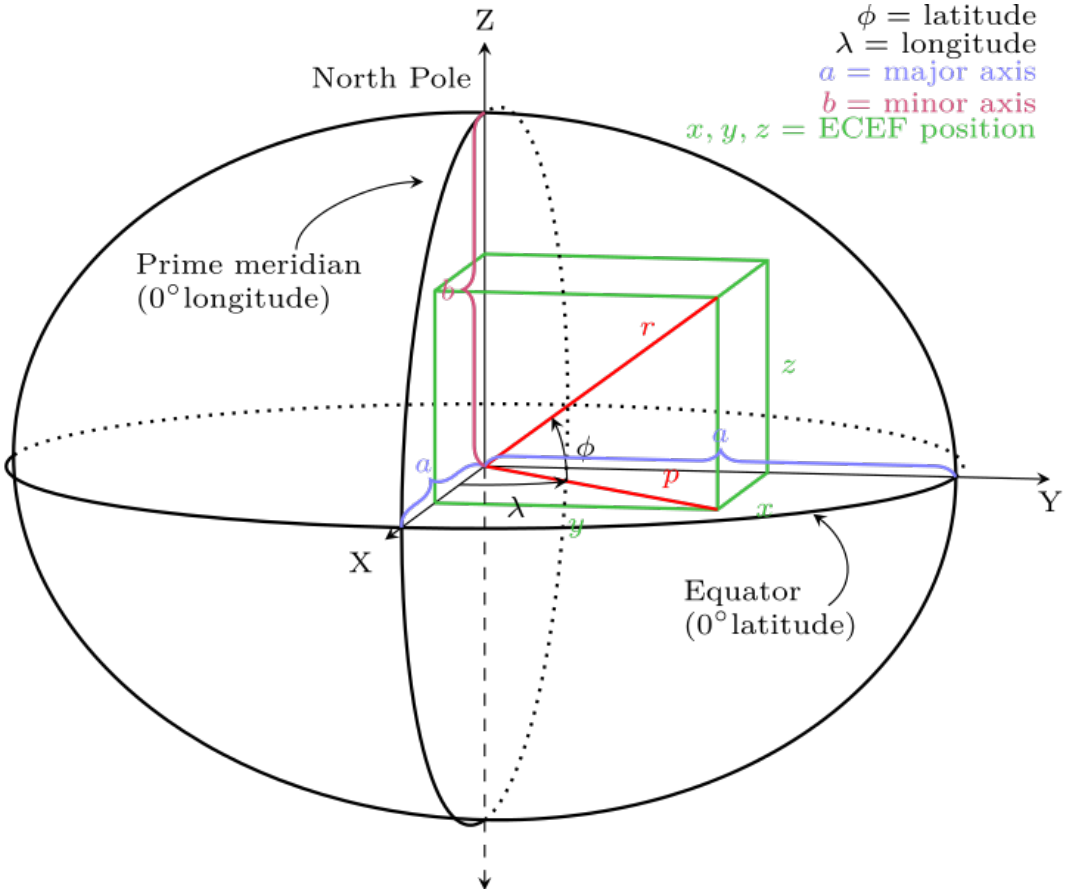


3.6.1 Geographic Coordinate System

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 [39]. 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 3.13, 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 3.13: 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 3.4) that define the shape of the ellipsoid (the Earth), they include a semi-major axis (a), a semi-minor axis (b) 3.14, its first eccentricity (e_1) and its second eccentricity (e_2).

FIGURE 3.14: Ellipsoid parameters

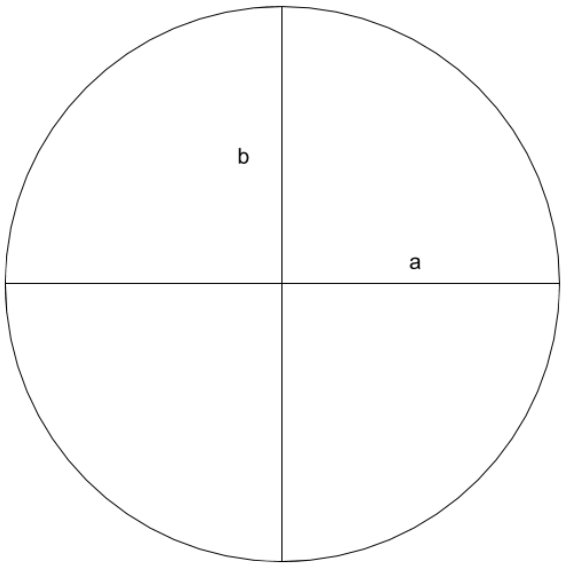
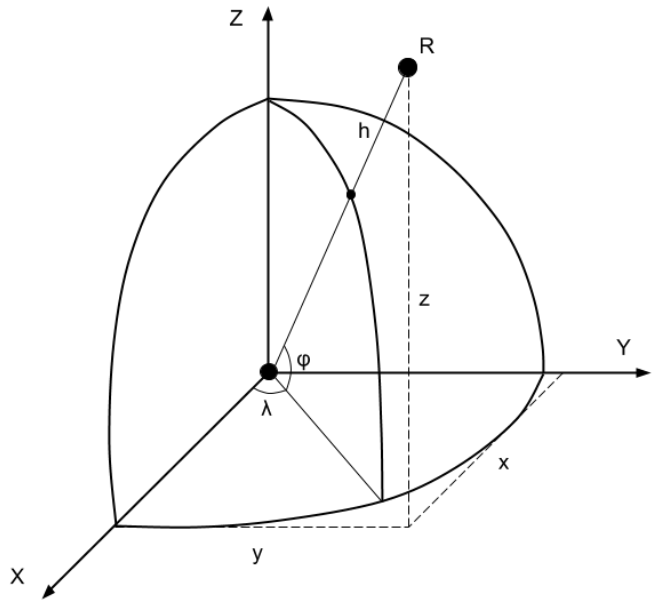


TABLE 3.4: WGS 84 parameters

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 3.15: 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.

$\Downarrow$ Reversal x, and switch z and y.

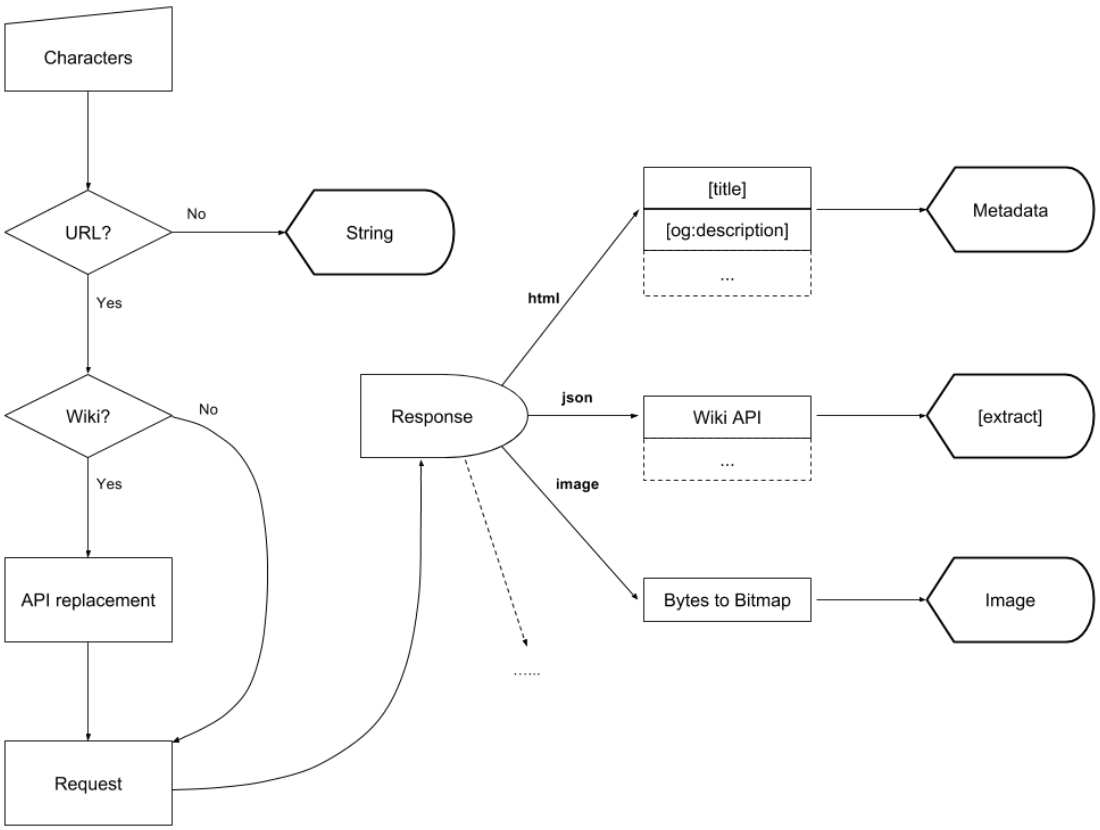
`(-x, z, y)` : x-east, y-north (up), z points to 0 latitude and 180 longitude.

3.6.2 Description

Raw description data of a `Placemark` is a series of characters decorated in the KML file, including any URL substring. Therefore, an analysis of the description is required before it transforms to be display. In the application, following data conversion has been implemented:

- **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 [29]) data (if it exists).

FIGURE 3.16: Description analysis



Treat the raw 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 [23]) 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 the APIs are:

```
 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 [44] 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 [30], 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 3.5.

TABLE 3.5: OBJ syntax

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

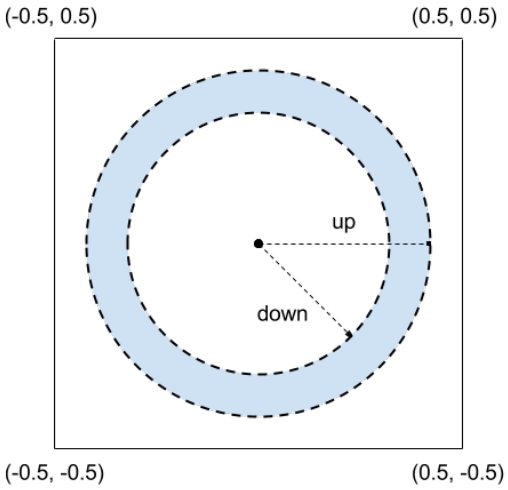
3.7 Ray

A sight indicator is displayed in the center of sight for tracking the user's eyes. It is important to understand that the immersive virtual reality device works with glass lenses. Therefore the centering is not exactly the center of both view container. There are 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 if the ray indicator is not be placed in a right distance. This phenomenon exists in reality as well. People can either get a better look at on a distant object or the fingers near to eyes.

3.7.1 Ray Pointer

Placing a ray pointer on the surface of the objects guarantees a clear display on both ray pointer and the target that user is staring at. It is drawn as `GL_POINTS` type, but round to a ring shape (see Figure 3.17) in the Fragment shader [38].

FIGURE 3.17: Ray point to ring



The `gl_PointCoord` in the Fragment Shader has a range from 0 to 1 regardless of the point size. Therefore, center the point for circular calculation by shifting with 0.5 offset on both x-axis and y-axis:

```
// transform coord from range [0, 1] to [-0.5, 0.5]
vec2 coord = gl_PointCoord - vec2(0.5);
```

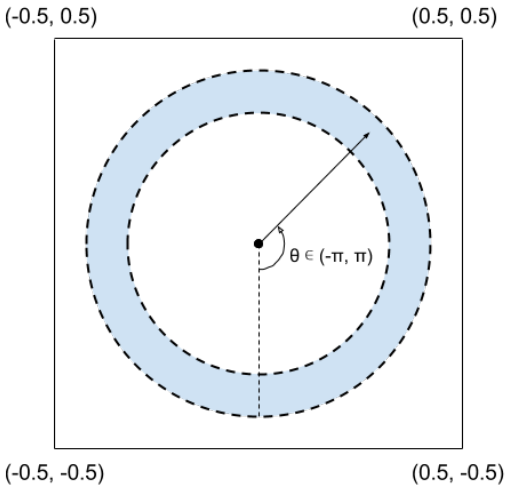
It is clear that a transformation from point to ring can be performed by ignoring invalid pixels in the point:

```
float length = length(coord); // distance from the center
bool belongsToPointer = pointerDown < length && length < pointerUp;
```

3.7.2 Ray Spinner

A ray spinner is an indicator gives a user feedback when the user is staring at a target. The ray spinner has a ring shape, but it is rendered as a dynamical arc by the given radian.

FIGURE 3.18: Ray ring to spinner



```
bool belongsToSpinner = spinnerDown < length && length < spinnerUp;
float theta = atan (-coord.x, coord.y); // (-PI, PI)
belongsToSpinner = belongsToSpinner && theta < u_Spinner;
```

The `u_Spinner` is a dynamical radian which came from CPU, and it has a range from $-\pi$ to π. The `genType atan(genType y, genType x)` returns the angle whose trigonometric arc tangent is $\frac{y}{x}$. The signs of `y` and `x` are used to determine the quadrant that the angle lies in. The value returned by `atan` is in the range from $-\pi$ to π.

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 [42].

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

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.

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

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

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

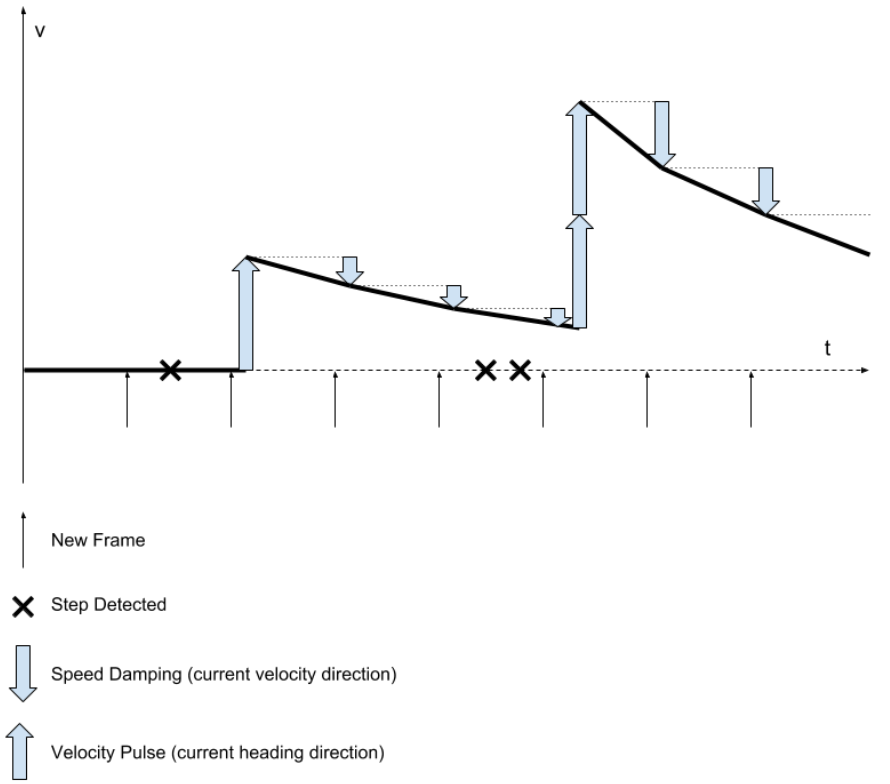
However, there is a technical limitation. First of all, we take the accelerometer data and remove gravity that is called gravity compensation, whatever is left is linear movement. Then we have to integrate it once to get velocity, and integrated again to get the position, which is called double integral. 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 [6].

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 3.19.

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

FIGURE 3.19: Camera movement



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

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

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

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

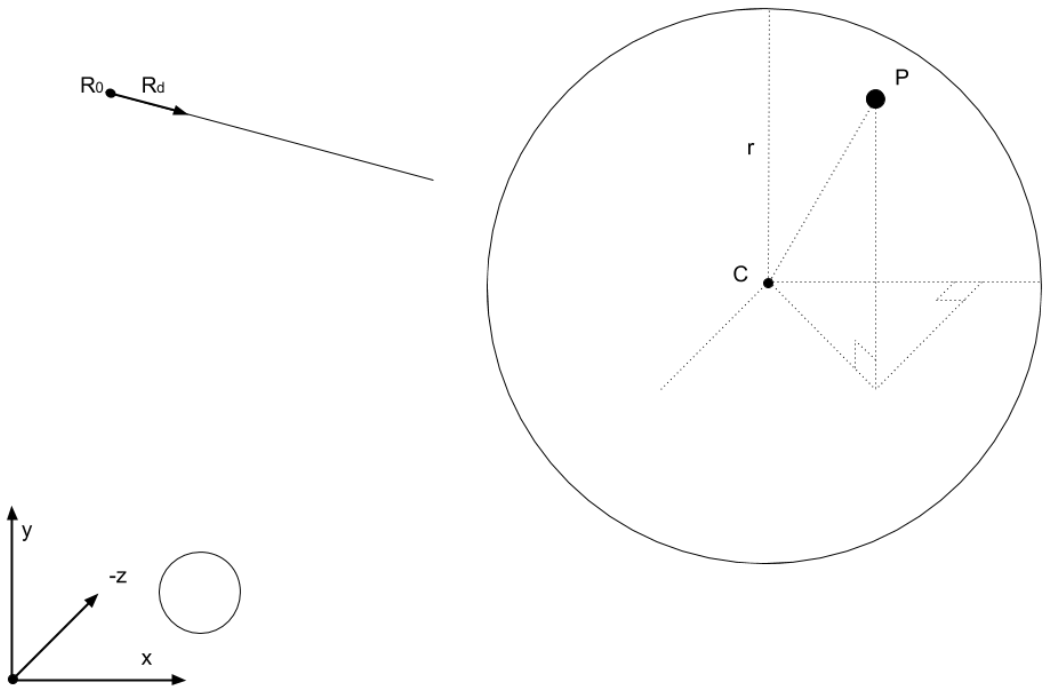
3.10 Ray Intersection

A ray can be described in an equation with known starting position, $\overrightarrow{R_0}$ and its direction $\overrightarrow{R_d}$ (see Equation 3.3). 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.

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

3.10.1 Ray-Sphere

FIGURE 3.20: 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:

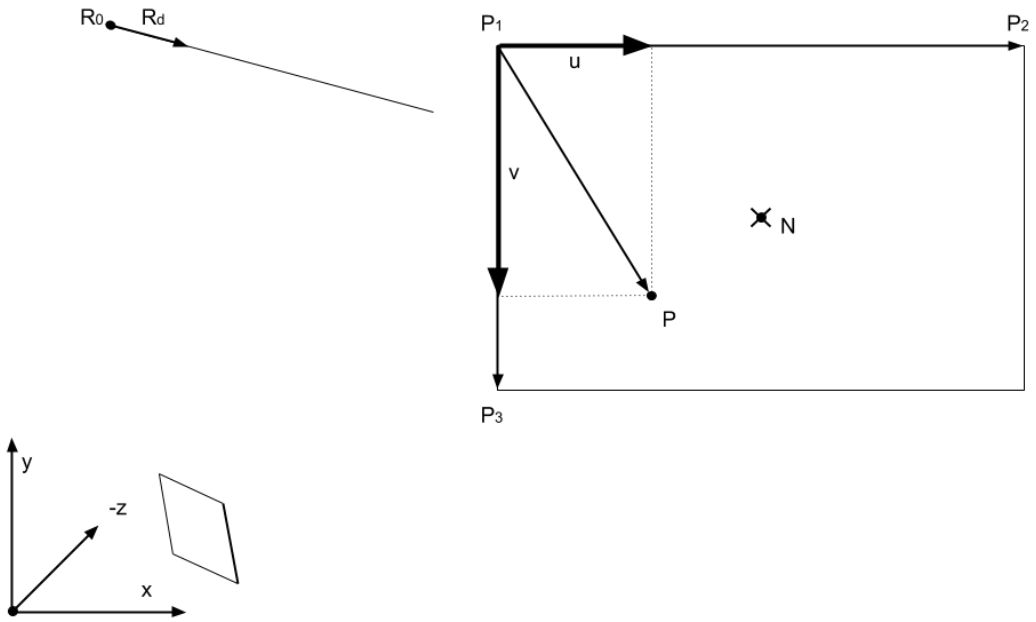
$$t = \begin{cases} -\alpha \pm \sqrt{\beta} & \text{if } \beta > 0 \\ -\alpha & \text{if } \beta = 0 \\ \varnothing & \text{if } \beta < 0 \end{cases}$$

$\therefore$ 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 3.21: 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:

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

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.

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

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

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

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:

$$\begin{aligned} \mu' &= (\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1}) \\ \nu' &= (\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1}) \end{aligned}$$

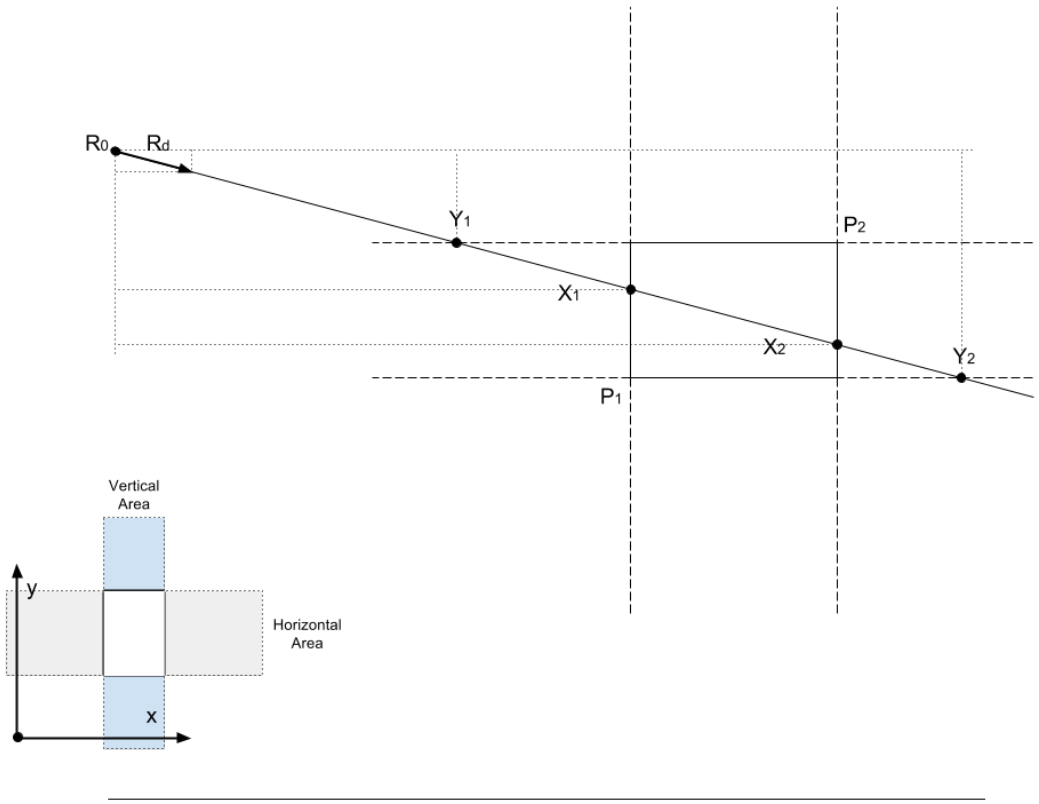
$$\begin{aligned} \mu' &\in [0,\ (\overrightarrow{P_2} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1})] \\ \nu' &\in [0,\ (\overrightarrow{P_3} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1})] \end{aligned}$$

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-model intersection tests. In this section, explanation divided into two parts, ray-box in 2D and which have inspired to ray-box 3D implementation. In the current implementation, I assume the boxes are located aligned with the axis, which is exactly how space partition works. This limitation can be solved with a certain rotation transformation in advance.

Ray-Box 2D

FIGURE 3.22: 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:

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

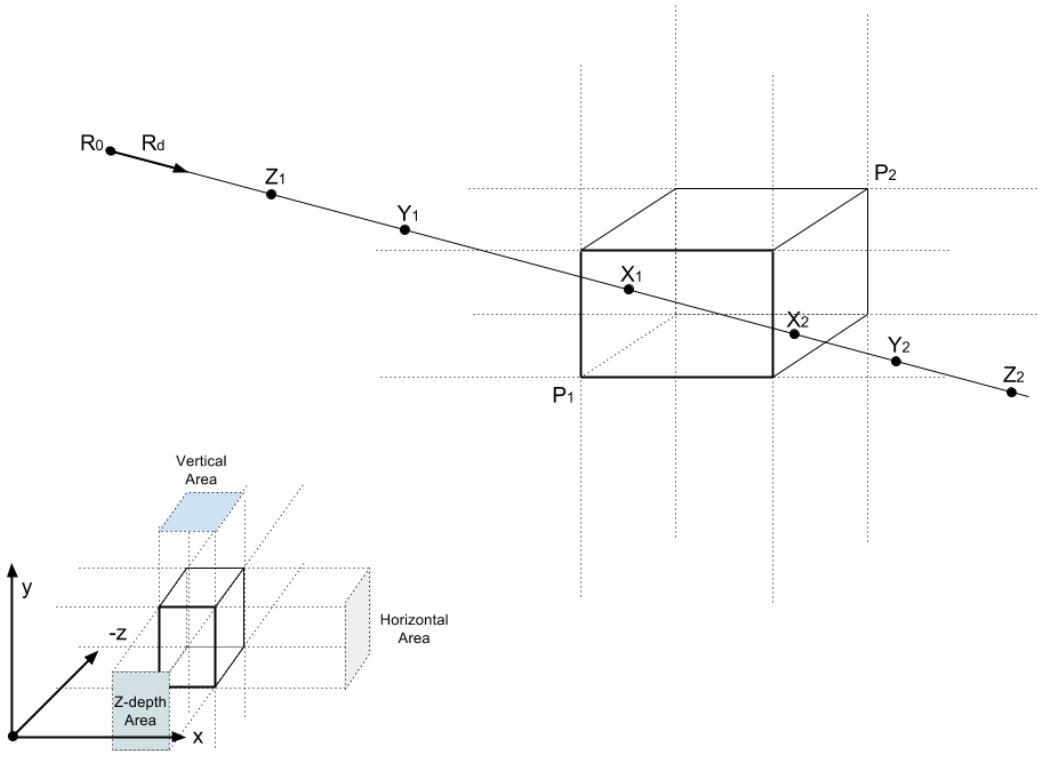
$$t_{X_2} = \frac{X_2}{x_{R_d}} \qquad t_{Y_2} = \frac{Y_2}{y_{R_d}}$$

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

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

Ray-Box 3D

FIGURE 3.23: 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 equations are satisfied:

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

3.10.4 Ray-Point

In order to maintain a same visual size on `Placemark`, application draw the `Placemark` looks the same size regardless of the distance in perspective view.

```
const float reciprocalScaleOnScreen = 0.01;
float w = (mvp * vec4(0, 0, 0, 1)).w * reciprocalScaleOnScreen;
vec4 position = vec4(a_Position.xyz * w , 1);
```

Therefore, ray-sphere intersection test for `Placemark` no longer works. In this section, I introduce ray-point intersection test for `Placemark` selection by checking if the ray is pointing at a particular direction in the camera space with a tolerance of pitch and yaw.

First, the final projected vertex in camera space (see Table 3.1) can be expressed as:

```
viewM = eye.viewM * cameraM;
Vertex' = perspectiveM * viewM * modelM * position;
```

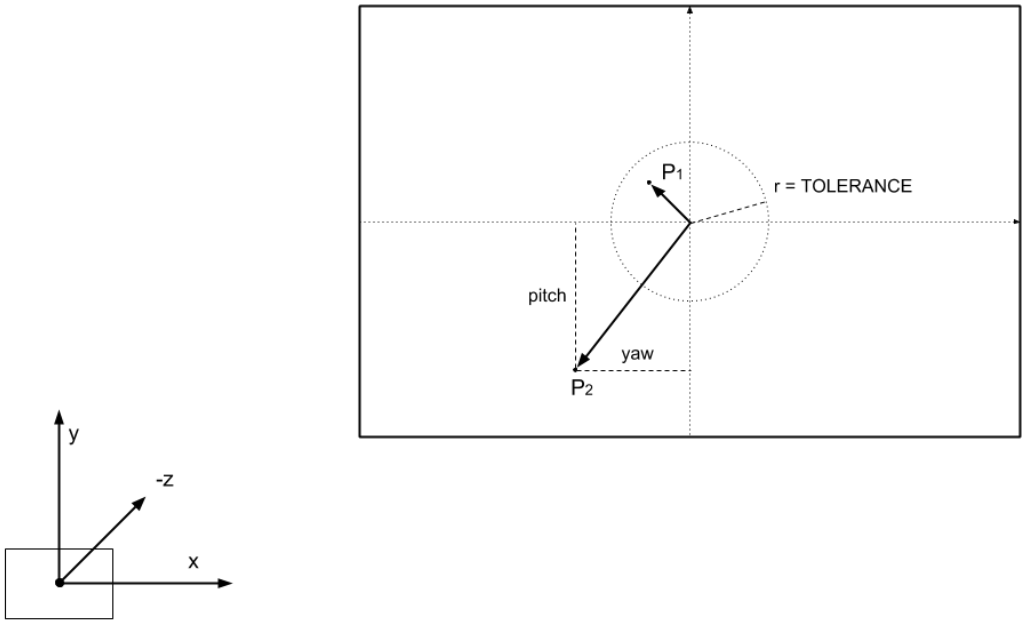
But, the expression of position in the camera space for ray-point intersection test is:

```
positionInCameraSpace = position - camera.position;
Position' = head.viewM * modelM * positionInCameraSpace;
```

Then, we can get the pitch and yaw value for intersection test:

```
pitch = atan2(Position'[1], -Position'[2]);
yaw = atan2(Position'[0], -Position'[2]);
abs(pitch) ∈ [-TOLERANCE, TOLERANCE]
abs(yaw) ∈ [-TOLERANCE, TOLERANCE]
```

FIGURE 3.24: Ray-Point intersection

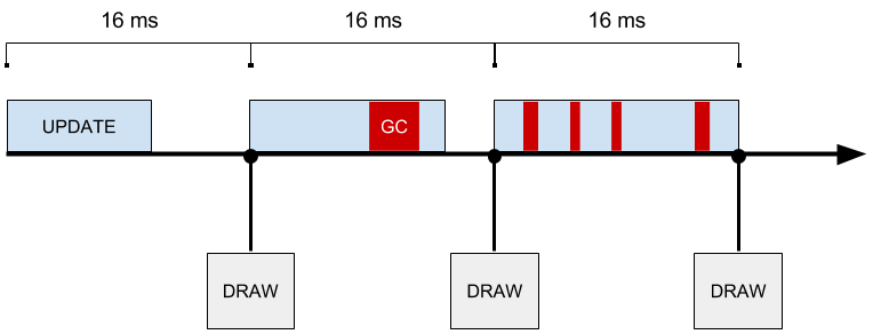


4 Performance

What is great about the Android runtime is that most of the stress of memory reclamation is done for developers. The system will track what developers are doing and when it sees that an object is not needed anymore, it will free it on their behalf. However, this does not exclude performance problems from happening here. When the amount of memory have allocated reaches an upper limit, a Garbage Collection (GC) event will be kicked off to free any resources that might not be needed any longer, freeing up space for future allocations.

Anytime the frame drips about the 16ms barrier, and the users are going to start to notice. Therefore, any code that forces allocated memory to spike above this threshold can cause problems. For instance, memory can become tighter, if the developer is allocating and freeing a large number of objects in a short period of time, the temporary objects again kicking off GC event.

FIGURE 4.1: 16ms per frame



Therefore, a performance testing is important for avoiding nasty GC events. Each GC event that developer can avoid, the application has more time per frame to do interesting things. In order to find out where in the code objects are being created but not released, created and not used, or created new when the developer could have been reusing them from existing objects. Android Studio provides a series of performance testing tools, such as Memory Monitor, Allocation Tracker, Heap Viewer, and the Systrace (it is an Android system trace tool helps developers analyze how the execution of the application fits into the many running systems on an Android device [17]). According to the runtime, the performance analysis is divided into two parts: acyclic process and cyclic process (performance data came from Android phone $Nexus\ 6P$[1]).

4.1 Initialization

In order to avoid UI being block, all of acyclic process in the application are handled in the new threads. In this section, I present the performance of geometric vertices' generation for `Placemark` (Icosphere) and `Earth` (UV Sphere) which only executes one time (or not) when needed. The geometric data from Icosphere generation will be cached once it has been created. This is useful to avoid duplicate creations for same geometric vertices. In the `Placemark` description URL analysis, the same cache solution for image and text.

4.1.1 Icosphere Generator

Based on the varying roundness of the Icosphere, it could evolve into spheres have different vertex count. In general, level 3 Icosphere is enough for illustrating a sphere, and in the application the `Placemark` is created on level 1 Icosphere. As we can see the performance testing result from Table 4.1 (40 data sets each), even for a particular reason that requires a level 5 Icosphere, less than 300ms is acceptable as an one time execution.

TABLE 4.1: Icosphere generation performance

Recursion Level	Vertex Count	Mean Value (ms)	Stand Deviation (ms)
0	12	0.049	0.014
1	42	0.257	0.490
2	162	0.722	0.250
3	642	3.296	0.707
4	2562	19.988	3.050
5	10242	148.927	18.422

[1]**CPU: 2.0GHz octa-core, 64-bit ARM Cortex-A57 & ARM Cortex-A53, 8 cores; GPU: Adreno 430**.

FIGURE 4.2: Icosphere performance sets

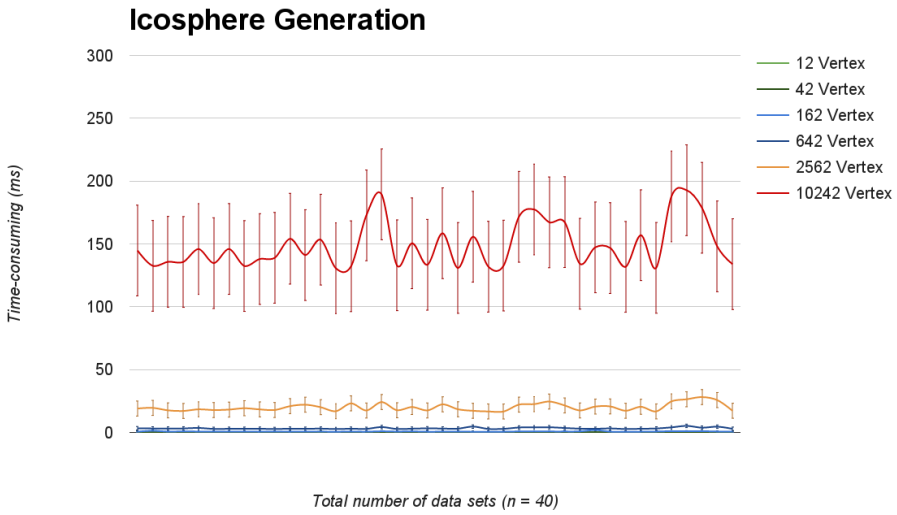
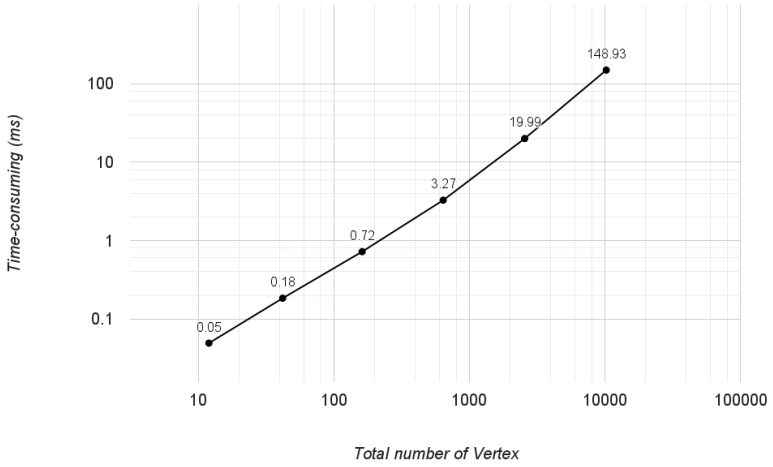


FIGURE 4.3: Icosphere performance loglog



4.1.2 UV Sphere Generator

The roundness of a UV Sphere depends on the partition level on both horizontal (ring) and vertical (segment) which denotes the axes of the 2D texture. The Earth is casually implemented as a $180/180$ UV Sphere in the application. Table 4.3 shows the testing result from 40 data sets each.

TABLE 4.2: UV Sphere generation performance

Partition Level	Vertex Count	Mean Value (ms)	Stand Deviation (ms)
5:5	25	0.100	0.033
10:10	100	0.201	0.072
20:20	400	0.322	0.100
40:40	1600	2.098	0.699
80:80	6400	5.431	1.153
160:160	25600	17.050	2.084

FIGURE 4.4: UV Sphere performance sets

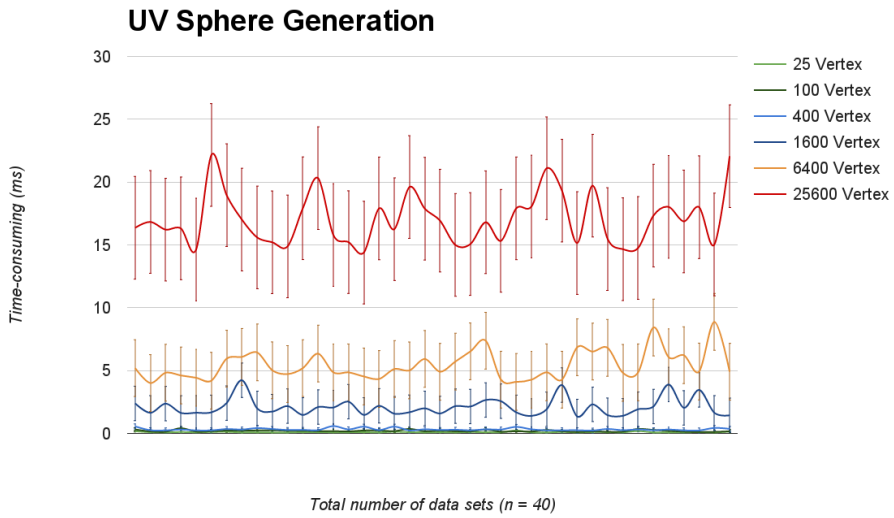
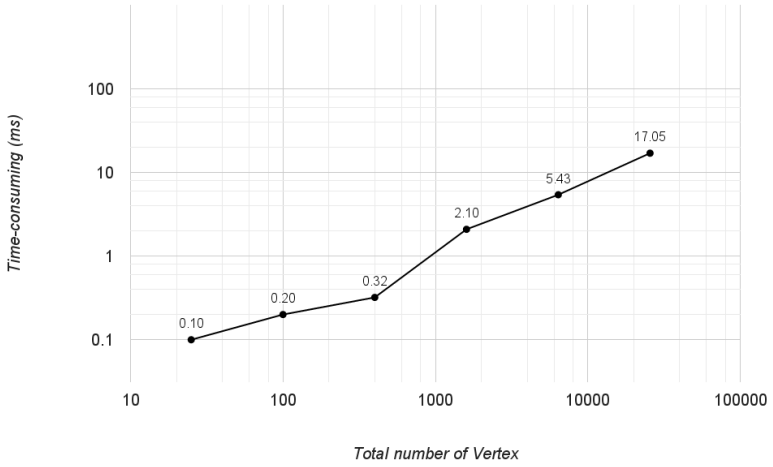
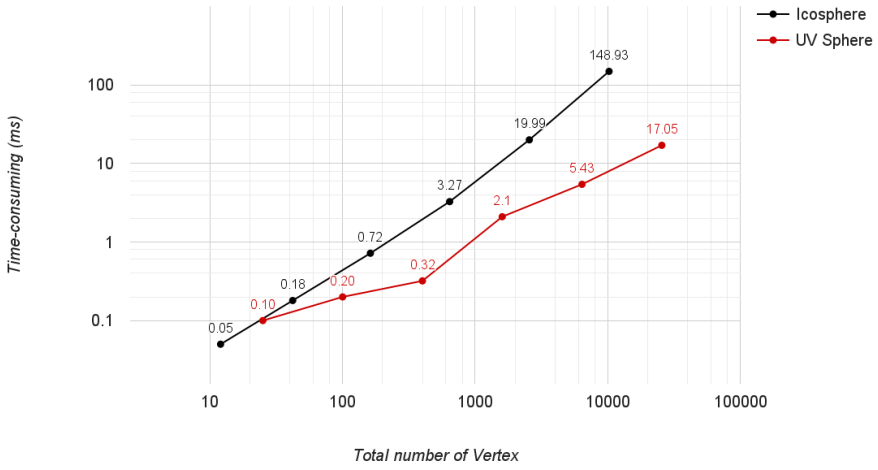


FIGURE 4.5: UV Sphere performance loglog



By comparison to Icosphere, UV sphere generation has better performance when application requires a very smooth, detailed surface.

FIGURE 4.6: Icosphere and UV Sphere



4.1.3 Geographic Data Initialization

Figure **??** is the performance of initializing geographic data, including parsing KML files, transformation coordinate system from LLA to ECEF for each `Placemark`, etc. The number of time-comsuming has linear increase with the size of `Placemark`, see following testing result from 10 data sets each.

TABLE 4.3: UV Sphere generation performance

Placemark Count	Mean Value (ms)	Stand Deviation (ms)
10	29.965	3.235
20	53.441	3.551
40	119.092	5.128
80	229.839	13.266
160	446.349	25.377
320	882.641	32.351

FIGURE 4.7: Geographic data performance sets

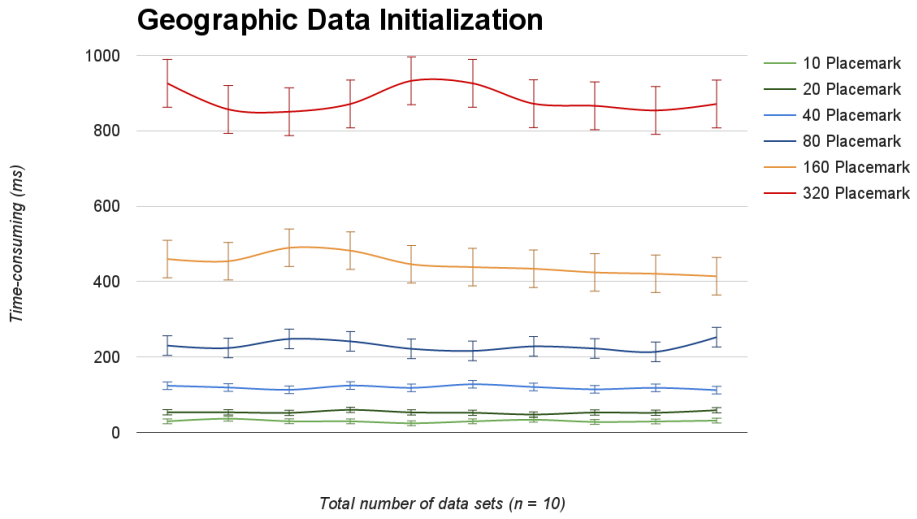
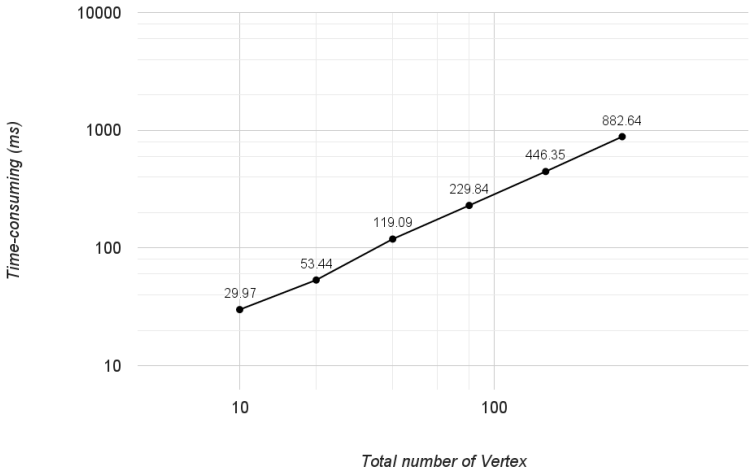


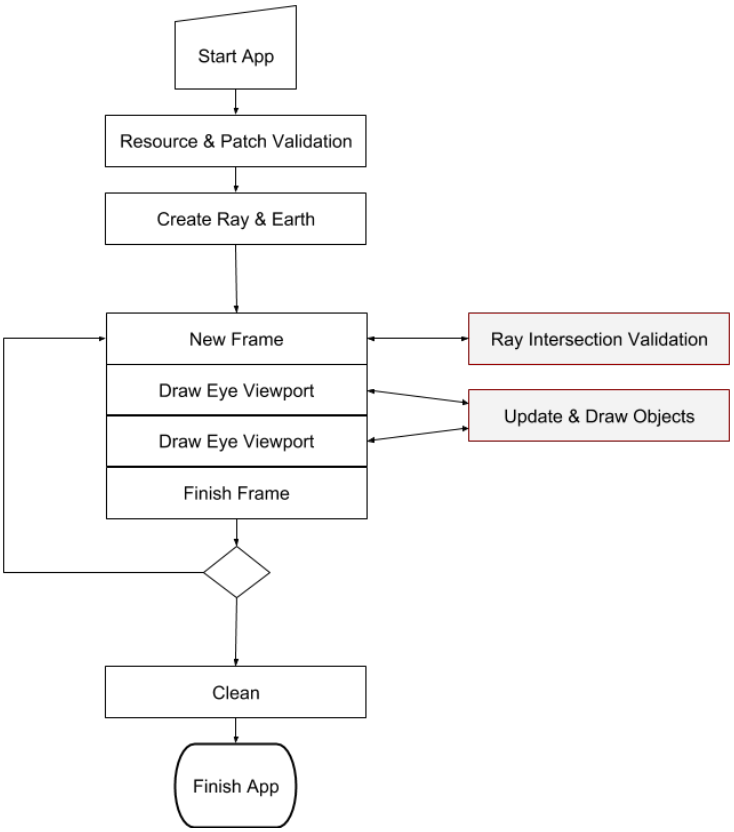
FIGURE 4.8: Geographic data performance loglog



4.2 Runtime

Tasks that require continually repeated during the render loop is the factors influencing the runtime performance. They are ray-model intersection detection, update and draw models 4.9. In this section, I present the tasks that closely related to performance and the memory leak analysis during the loop.

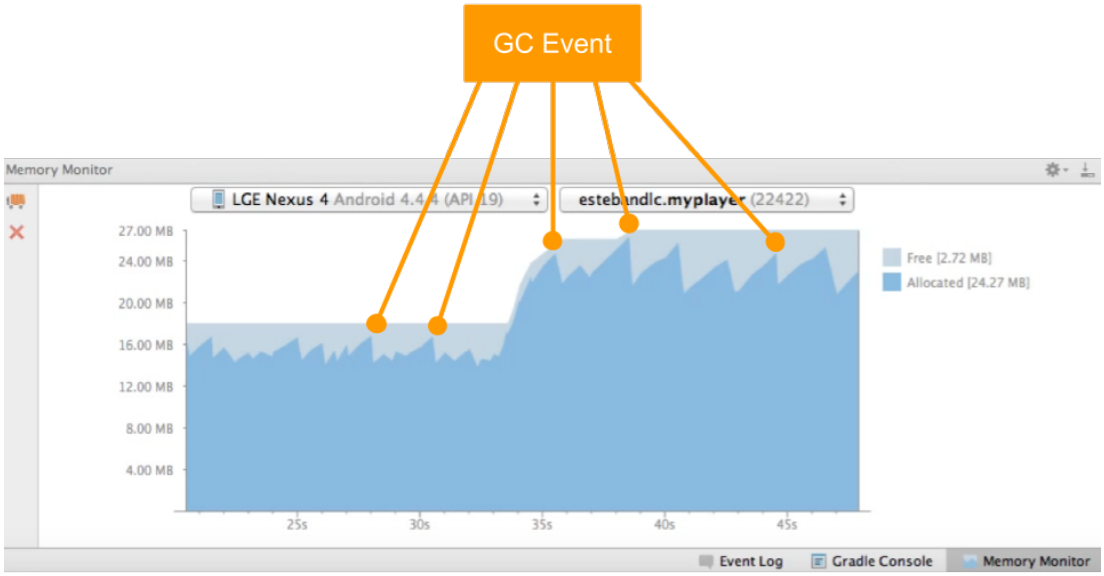
FIGURE 4.9: Runtime task



4.2.1 Memory Leak

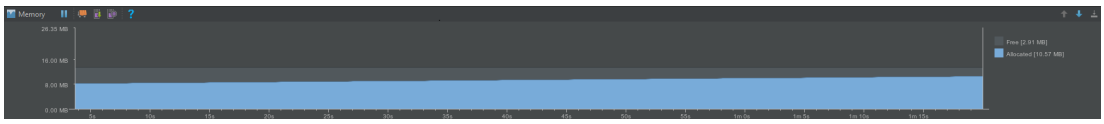
I am using Memory Monitor to evaluate memory leak. As the application allocates and free memory, we will see the allocative amount fluctuate in the graph at the same time. Any time the allocated memory drops by a significant amount, that is a signal that GC event has occurred (see 4.10). These GC events are not generally a noticeable performance problem. However, lots of them occurring over and over and over again in a short period of time can lead to performance issues. Additionally, it more or less created memory leaks, which are objects which the application is no longer using, but the garbage collector fails to recognize them as unused.

FIGURE 4.10: Memory Monitor [8]



In a world where the application is not doing much of anything, there will be a flat memory allocation graph. This is an ideal scenario from a performance perspective. The more time is spending doing GC, the less time for the application to do other stuff. Fortunately, with a careful in dealing with objects, I got a quite flat runtime memory allocation 4.11.

FIGURE 4.11: Memory performance



4.2.2 Placemarks Intersection

There was a significant performance improvement in detecting ray-placemark intersection per frame. In order to avoid doing intersection test on all `Placemark` per frame, I implemented Octree space partition to ignore unnecessary detections (see Chart 4.14).

FIGURE 4.12: Placemark intersection performance - Before

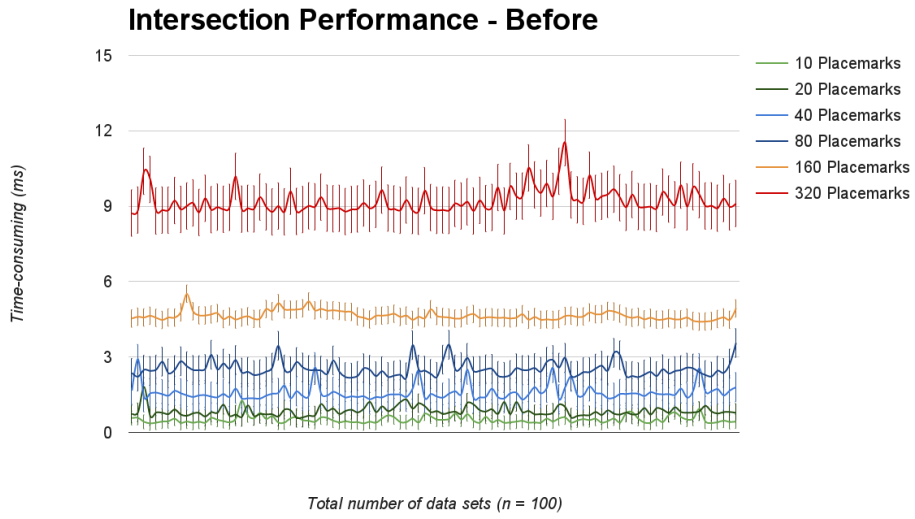


FIGURE 4.13: Placemark intersection performance - After

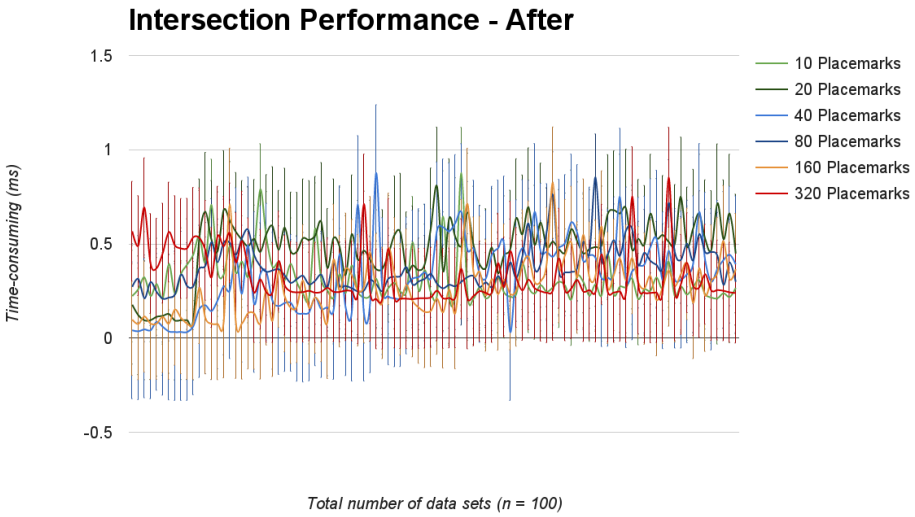
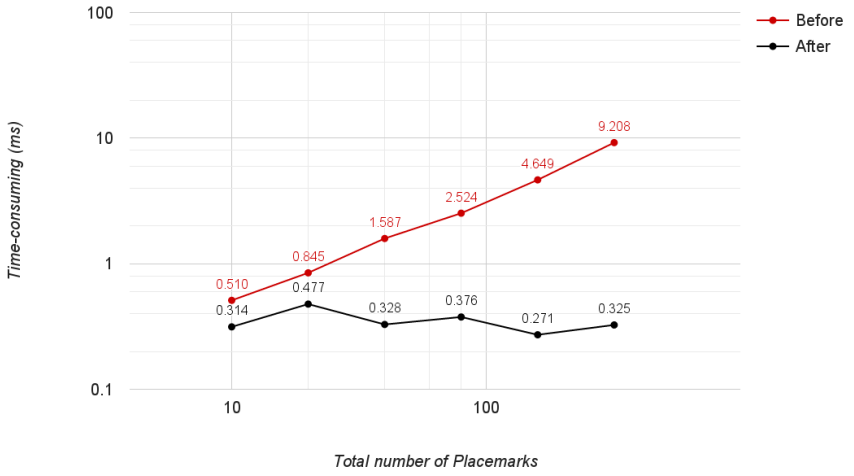


FIGURE 4.14: Placemark intersection compare



After the optimization, the number is negligible even for 1000 `Placemark` (less than 2ms).

4.2.3 Placemarks Update

There was a incredible performance optimization for ray-placemark update process 4.17.

FIGURE 4.15: Placemark update performance - Before

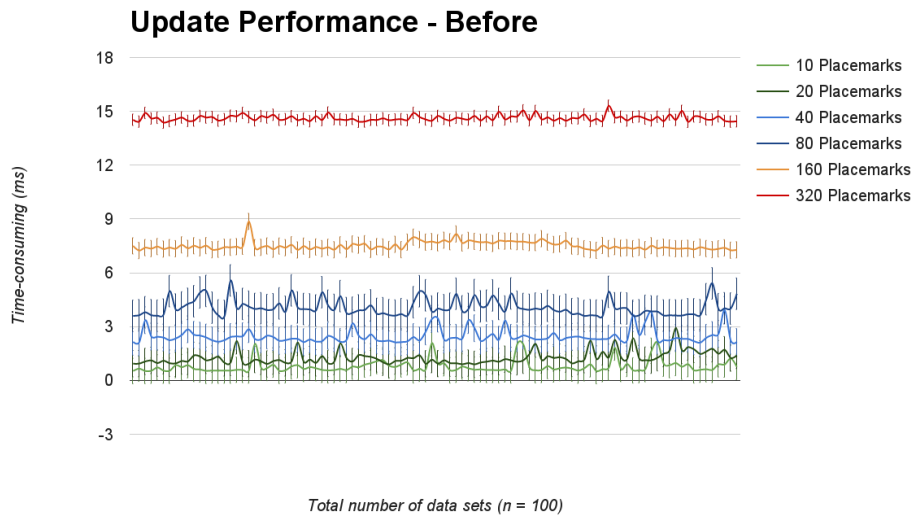


FIGURE 4.16: Placemark update performance - After

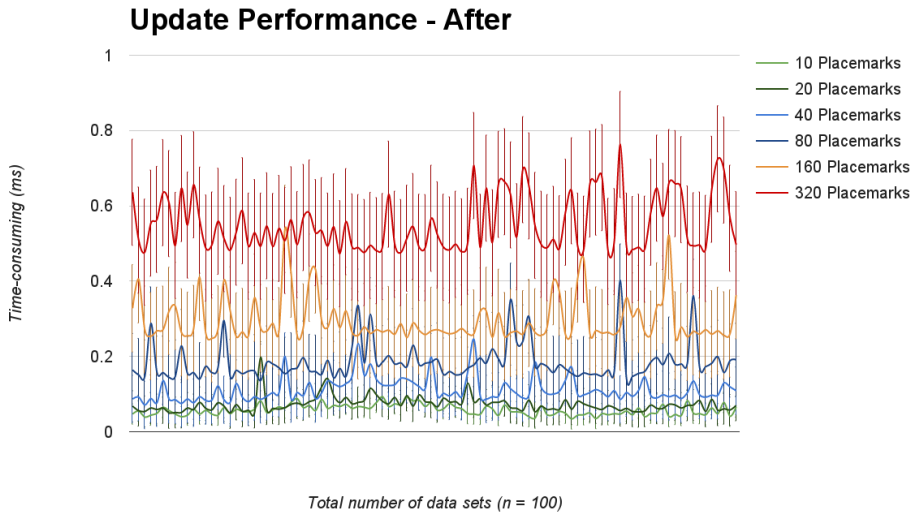
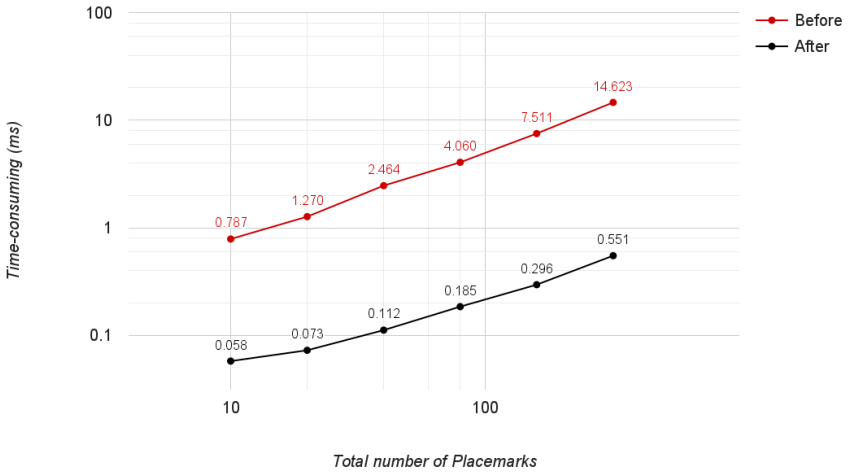


FIGURE 4.17: Placemark update compare



There are mainly 6 type of calculations (see Table 4.4):

TABLE 4.4: OpenGL compute scope

What	How	Scope
Model Matrix	translationM * scaleM * rotationM * identityM(1)	Object specific
Camera Matrix	lookAt(positionV, lookAtV, upV)	Worldwide
View Matrix	eye.viewM * cameraM	Eye specific
Perspective Matrix	eye.perspective(zNear, zFar)	Eye specific
ModelView Matrix	viewM * modelM	Object specific
Projection Matrix	perspectiveM * modelViewM	Object specific
Vertex'	projectionM * vertex	Vertex specific

Before the optimization, based on the `viewM` and `perspectiveM` from each eye, computing the `modelM`, `modelViewM` and `projectionM` in CPU for each object per frame. Then pass the `modelViewM` and `projectionM` to GPU for further calculation, such as lighting and position (`Vertex'`). It creates huge performance problem, due to a large amount of computation can be omitted. There is no need to re-calculate an object's `modelM` per frame if the object has the same position, scale, and rotation. It is the same for `modelViewM` and `projectionM`. Therefore, after the optimization, application only re-calculate the `modelM` for objects which needed, and all objects pass `viewM` and `perspectiveM` to GPU for the rest of calculation. Although it increases the workload in GPU as well as duplicated calculation, it has improved performance for the application. Dealing with the duplicated calculation in GPU caused by the optimization has been postponed to future work, because it related to optimization of drawing massive `Placemark` which has been postponed as well (see the next Section 4.2.4).

TABLE 4.5: OpenGL compute optimization

What	Before	After
Model Matrix	Each object (CPU)	Objects are needed (CPU)
Camera Matrix	When moving (CPU)	When moving (CPU)
View Matrix	Each eye (CPU)	Each eye (CPU)
Perspective Matrix	Each eye (CPU)	Each eye (CPU)
ModelView Matrix	Each object (CPU)	Each Vertex (GPU)
Projection Matrix	Each object (CPU)	Each Vertex (GPU)
Vertex'	Each Vertex (GPU)	Each Vertex (GPU)

4.2.4 Placemarks Draw

The performance of draw all `Placemark` has not been adequate, due to the number of times calling `glDrawElements` API, which is an expensive call and it is called during each `Placemark`

draw process. Anytime the frame takes more than 16ms barrier, and the user will notice it. Because the two eye's viewports are both need to be updated and re-draw, therefore the update and draw process only has 8ms in total to do what needs to be done. Although the performance of ray-object intersection and `Placemark` update are satisfied, but the performance issues in draw process are significant. The 0.048ms average time-consuming and 0.046ms stand deviation indicate that it not only has an unstable value but also it could reach 8ms by just call the API 170 times (See Chart 4.18).

FIGURE 4.18: glDrawElements performance

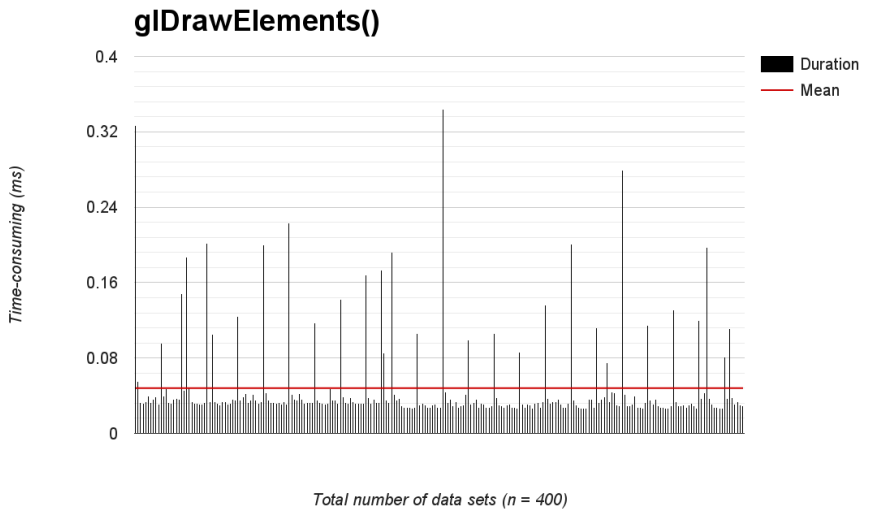
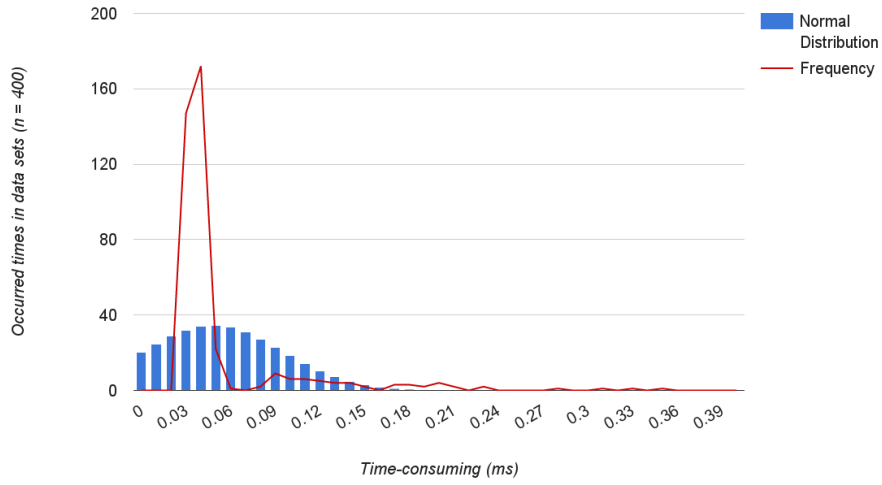


FIGURE 4.19: glDrawElements performance normal distribution



The solution is straightforward using one `glDrawElements` call for all `Placemark`. Unfortunately, it has not been optimized yet due to a limited time. In general, using interleaved attributes will do the trick.

5 Discussion

In this paper, I implemented an immersive virtual reality Android application includes a remote database server and a graphic display system for geographic data visualization. Popping a smartphone into the Google Cardboard container and then strapping it to people's head may sound like funny, but it works, and it is a super low-cost way (15 US dollars) to experience virtual reality compare to other particular virtual reality devices. After all, Android phone contains all the necessary sensors and positioning systems to track device movements which contribute to a development of complex virtual reality application.

KML format data source has been used for decorating geographic data in the application. A KML format file not only provides basic needs for a decoration of geographic data but also has the ability to contain remote files by URL. It guarantees a real-time synchronous 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. The human readable plain text is beneficial to publish and consume data in interoperable formats without the needs of technical assistance. Furthur more, KML has been adopted by more and more people and scientists across the different industry. Thus, there is a mass of geographic data resources created by KML have been shared within geographic communities. For instance, existing virtual globe software Google Map and Google Earth.

The application has an easy-to-use, intuitive nature working mode. On the one hand, a user can make a six degrees of freedom (DOF) movement. Although due to the sensor fusion creates a huge drift during the nasty double integration process, I alternatively using the Step sensor (pedometer) as the pedestrian navigation instead of Linear Acceleration sensor. It allows the user to move forward in the current heading that satisfies user for navigating through all scene. On the other hand, the application enables the ability for the user to intuitively interact with the scene for a better understanding of information. For example, select a `Placemark` and view the details of the `Placemark` on a popup message board; display a `Placemark` related OBJ model, or any further information from a URL, such as an image, summarized Wikipedia, or plain text.

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%) [24]. The immersive virtual reality certainly improved the feedback of sight sense and hearing. Although

spatial sound is not included in the application yet, but by the given the existing Spatial Audio technology (such as [16]), it can easily use the spatial audio as a simultaneous response for "fooling" the hearing sense.

There is a limitation of gesture recognition and perception technology, in particular with Android smartphones. As the modern phone, they are now being designed to be less physical key as possible. Therefore a user can only trigger a click (a touching) or focus on somewhere (staring at a target). On the other hand, PC-based pseudo-3D virtual globes are manipulated by the mouse and keyboard which allows more straightforward interactive actions.

The application guarantees 60 FPS when the there is less than 150 `Placemark` exist in the scene. Although this number is not satisfied as a complex geographic visualization application, it has huge potential for releasing the restricted performance issues. As we can see from the performance testing result 4, application it has a smooth memory usage, and the optimized ray-model intersection and model updates are extremely successful. Once an future optimization of `glDrawElements` is implemented, there will be no pressure for thousands of objects existing at the same time.

The application satisfy basic needs as an immersive virtual reality application for geographic data visulization, along with a remote file server and RESTful API support. However, it still has many unfinished optimizations and features due to a limited time. As we can see from the KML schema 2.3, application only supports a small part of KML facility.

One of the important features in geographic data visualization is Level Of Detail (LOD) render based on the distance from the eyes to the target area. Detail textures could be separately prepared, and attached as the circumstances may require. It can also provide a solution to visualize a large amount of overlapping data (enable layers on distance). Additionally, along with more geometric shapes support in the application (such as lines and polygons), LOD allows the user to see the details not only geographical map but also the architectural structure on different floors.

A key requirement in environmental scientists is to be able to visualize four-dimensional data (i.e. time-dependent three-dimensional data). It does not just visualize the environment data from different data files which created from the different period of time, or a fake real-time data visualization by refreshing in a certain frequently rate. But, the ability to visualize dynamic graphic animation on a flexible timeline. It improves user understanding of environmental data to a higher level. In other words, an animation transform from one piece of time-dependent data to another, just like watching the 3D movie.

After all, in the immersive virtual reality visualization point of view, there is always room for improvement. When it related to human intuitive nature system, the user experience will always be not real enough compare to the real world.

6 Conclusion

The immersive virtual reality provides a highly integrated easy-to-use, intuitive system. It's efficient and simultaneous property provides an attractive way for 3D geographic data visualization. By comparison to virtual globes, geographic data visualization with immersive virtual reality device allows to do similar things, but, due to a limitation of human-machine interaction, gesture recognition, and perception, virtual reality device is not yet able to do everything that the pseudo-3D virtual globes can do. However, it has the potential to do more than people expected in the future.

Android SDK for virtual reality development is well supported. By making use of Google Cardboard and Android smartphone is not only the most low-cost and convenient way to experience immersive virtual reality but also easy to develop and spread the virtual reality related application and product. However, Android has a precise limitation on short distance navigation. The most logical way to calculate the movement in an immersive virtual reality environment is to first use Gyroscope to measures angular velocity relevant 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 on gravity, a correction due to the magnetic north from Compasses is also required. However, it is 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 in the application based on the Step sensor and a certain algorithm for velocity.

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:

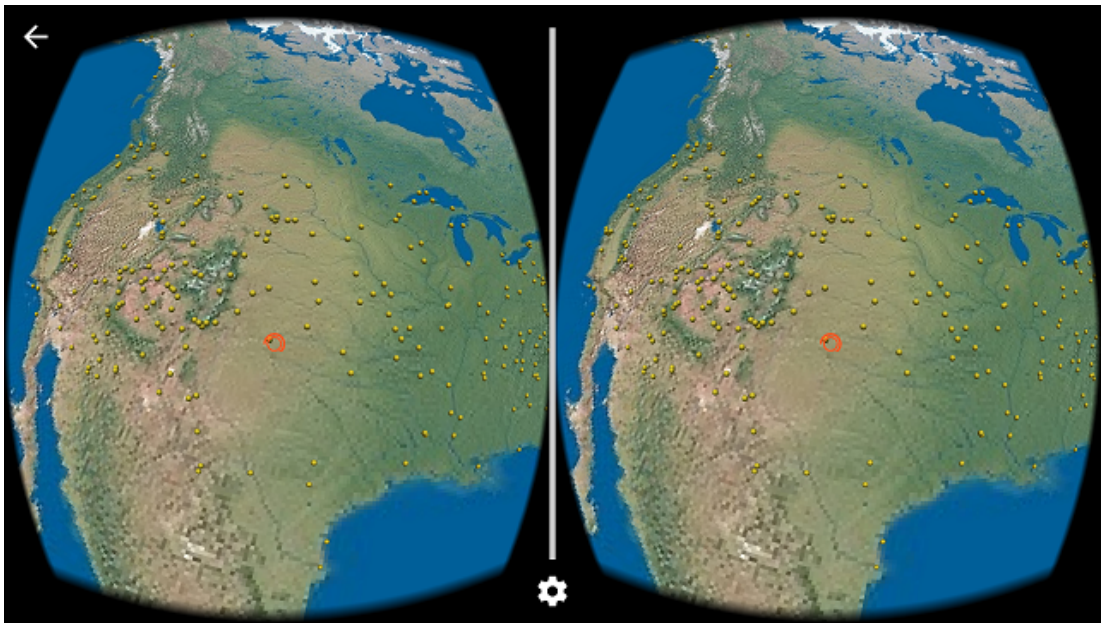
https://github.com/jiangyang5157/virtual-reality

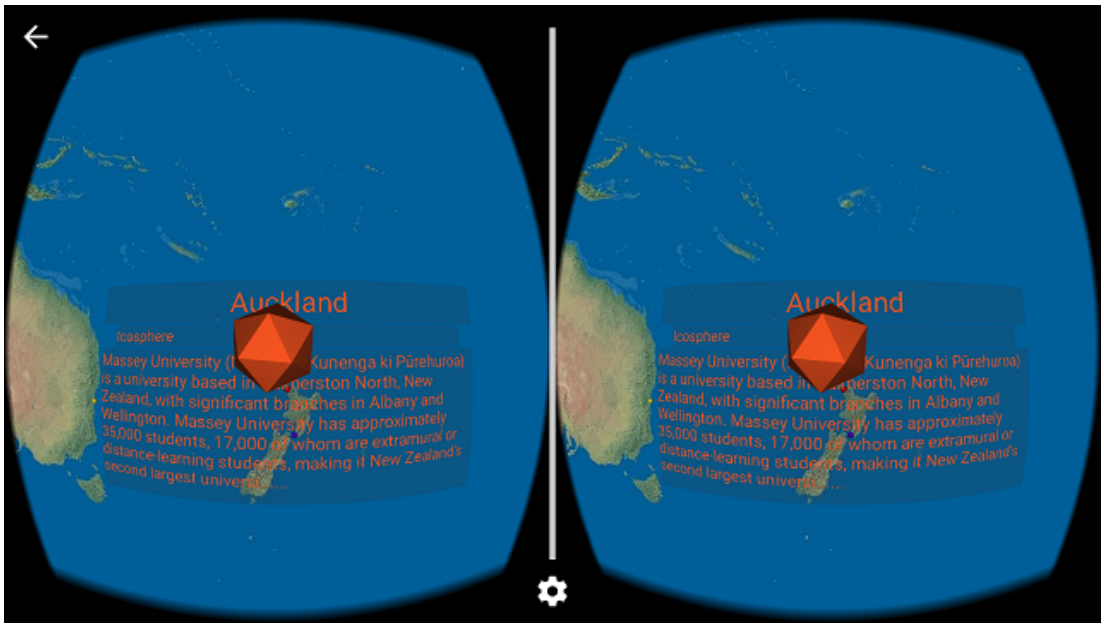
https://github.com/jiangyang5157/tookit

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

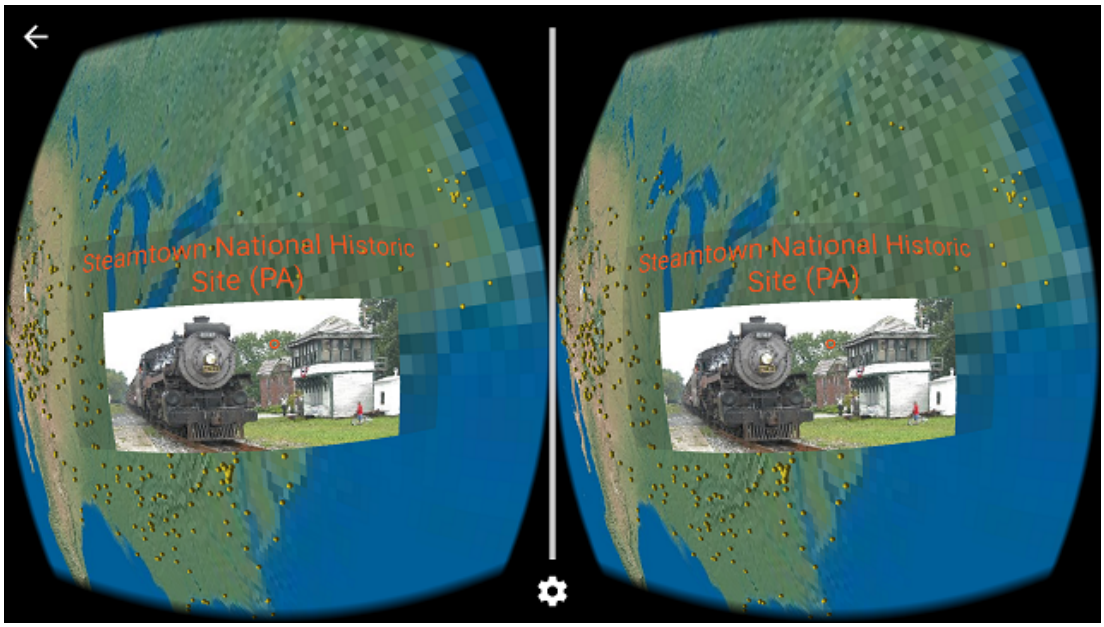
https://github.com/jiangyang5157/massey-master-thesis-2016

B Screenshots




Auckland
Icosphere
Massey University (
Kunenga ki Pūrehuroa)
is a university based in
perston North, New
Zealand, with significant branches in Albany and
Wellington. Massey University has approximately
35,000 students, 17,000 of whom are extramural or
distance-learning students, making it New Zealand's
second largest univers
Auckland
Icosphere
Massey University (
Kunenga ki Pūrehuroa)
is a university based in
perston North, New
Zealand, with significant branches in Albany and
Wellington. Massey University has approximately
35,000 students, 17,000 of whom are extramural or
distance-learning students, making it New Zealand's
second largest univers


New South
city in Australia and
Located on Australia's east coast, the
polis surrounds the world's largest natural
bour, and sprawls towards the Blue Mountains to
the west. Residents of Sydney are known as
"Sydneysiders". Sydney is the.....
New South
city in Australia and
Located on Australia's east coast, the
polis surrounds the world's largest natural
bour, and sprawls towards the Blue Mountains to
the west. Residents of Sydney are known as
"Sydneysiders". Sydney is the.....


Steamtown National Historic Site (PA)
Steamtown National Historic Site (PA)

C KML Sample

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Style id="placemark_1">
  <IconStyle> <color>7f0000ff</color> </IconStyle>
</Style>
<Style id="placemark_2">
  <IconStyle> <color>ffff0055</color> </IconStyle>
</Style>
<Folder>
<Placemark>
  <name>Auckland</name>
  <styleUrl>#placemark_1</styleUrl>
  <description>
    https://en.wikipedia.org/wiki/Massey_University
  </description>
  <Point> <coordinates>174.76192,-36.84845</coordinates> </Point>
  <ExtendedData>
    <Data name="title">
      <value>Icosphere</value>
    </Data>
    <Data name="obj">
      <value>
        http://127.0.0.1:8080/assets/static/model/icosphere.obj
      </value>
    </Data>
  </ExtendedData>
</Placemark>
<Placemark>
  <name>Wellington</name>
  <styleUrl>#placemark_2</styleUrl>
  <description>
    https://en.wikipedia.org/wiki/monkey
  </description>
```

```
  <Point> <coordinates>174.77623,-41.28646</coordinates> </Point>
  <ExtendedData>
    <Data name="title">
      <value>Monkey</value>
    </Data>
    <Data name="obj">
      <value>
        http://127.0.0.1:8080/assets/static/model/monkey.obj
      </value>
    </Data>
  </ExtendedData>
</Placemark>
</Folder>
<Folder>
  <NetworkLink>
    <Link>
      <href>
        http://127.0.0.1:8080/assets/static/kml/example_simple2.kml
      </href>
    </Link>
    <ExtendedData>
      <Data name="sync">
        <value>1</value>
      </Data>
    </ExtendedData>
  </NetworkLink>
</Folder>
</Document>
</kml>
```

D OBJ Sample

```
# 8 vertex cube
# front U-1234
v -1 1 1
v -1 -1 1
v 1 -1 1
v 1 1 1
# back U-5678
v -1 1 -1
v -1 -1 -1
v 1 -1 -1
v 1 1 -1
# vertex normal
# front
vn 0.0 0.0 1.0
# back
vn 0.0 0.0 -1.0
# top
vn 0.0 1.0 0.0
# bottom
vn 0.0 -1.0 0.0
# left
vn -1.0 0.0 0.0
# right
vn 1.0 0.0 0.0
# faces v//vn
# front face
f 1//1 2//1 3//1
f 1//1 3//1 4//1
# back face
f 8//2 7//2 6//2
f 8//2 6//2 5//2
# top
f 5//3 1//3 4//3
f 5//3 4//3 8//3
```

```
# bottom face
f 7//4 3//4 2//4
f 7//4 2//4 6//4
# left face
f 5//5 6//5 2//5
f 5//5 2//5 1//5
# right face
f 4//6 3//6 7//6
f 4//6 7//6 8//6
```

Bibliography

[1] 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.

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

[3] 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.

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

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

[6] 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.

[7] ——, (2012). Go at google: Language design in the service of software engineering, [Online]. Available: https://talks.golang.org/2012/splash.article.

[8] ——, (2015). Android performance patterns: Tool - memory monitor, [Online]. Available: https://developer.android.com/studio/profile/systrace.html.

[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). System trace, [Online]. Available: https://www.youtube.com/watch?v=7ls28uGMBEs.

[18] ——, (2016). The go programming language, [Online]. Available: https://golang.org/.

[19] ——, (2016). Transmitting network data using volley, [Online]. Available: https://developer.android.com/training/volley/index.html.

[20] 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.

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

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

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

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

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

[26] 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.

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

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

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

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

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

[32] 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.

[33] 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.

[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). Fragment shader, [Online]. Available: https://www.opengl.org/wiki/Fragment_Shader.

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

[40] ——, (2016). Geographic information system, [Online]. Available: https://en.wikipedia.org/wiki/Geographic_information_system.

[41] ——, (2016). Opengl shading language, [Online]. Available: https://en.wikipedia.org/wiki/OpenGL_Shading_Language.

[42] ——, (2016). Rotation matrices, [Online]. Available: https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles.

[43] ——, (2016). Vrml, [Online]. Available: https://en.wikipedia.org/wiki/VRML.

[44] ——, (2016). Wavefront obj file, [Online]. Available: https://en.wikipedia.org/wiki/Wavefront_.obj_file.

[45] ——, (2016). World wide web, [Online]. Available: https://en.wikipedia.org/wiki/World_Wide_Web.

[46] ——, (2016). X3d, [Online]. Available: https://en.wikipedia.org/wiki/X3D.