MASSEY UNIVERSITY

Geographic Data Visualization with Virtual Reality

Author
Yang JIANG

Supervisor
Dr Arno LEIST

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

in the

Software Engineering
159.888 Computer Science Professional Project

October 14, 2016

Abstract

Geographic data visualization is effective for not only presenting essential information in vast amounts of data but also driving complex analyses. In the last decade, along with the rapid development of science and technology, numbers of very successful virtual globes software appeared and became super popular across all kind of communities. However people always want more, they want to step into this world and interact with it, instead of just watching a picture on the monitor, this technology which becomes overwhelmingly popular and fashionable in current decade is called Virtual Reality (VR). The propose of this project is to explore a feasible scheme for geographic data visualization with VR by implement both front and back end software that support VR device.

Keywords: Visualization, Virtual Reality, Android, Opengl ES, KML

Contents

List of Figures

List of Tables

1 Introduction

Undoubtedly VR has attracted a lot of interest of people in last few years, along with a mess of VR related products, such as Google Cardboard that be able to turn Smartphone to VR device, 3D 360 camera, and 3D VR headset are developed by different manufacturers.

Three-dimensional (3D) representations of objects and spaces are frequently used to improve human understanding (Tuttle, Anderson, and Huff, 2008). Therefore VR is appropriate to be used as geographic data visualization.

1.1 Background and Overview

VR provides an easy, powerful, intuitive way of human-computer interaction. The user can watch and manipulate the simulated environment in the same way we act in the real world, without any need to learn how the complicated user interface works. Therefore many applications like flight simulators, architectural walkthrough or data visualization systems were developed relatively fast (Mazuryk and Gervautz, 1996).

1.2 Literature Review and Limitations

****
What is known?
What is unknown?
Review of research
Identifying gaps

1.3 Aims and Objectives

Due to the equipment costs and lacks of VR softwares, by compare with successful virturl globes, geographic data visualization with VR has not yet as popular as virturl globe.

Our Objectives is to explore and implement a VR project that runs on a minimum equipment costs VR device. By doing the project, we develop a geographic data visualization VR software that includes both front and back end.

The project is expected to show geographic data visualization with VR can be easily used as virturl globe, and it will be widespread in the future.

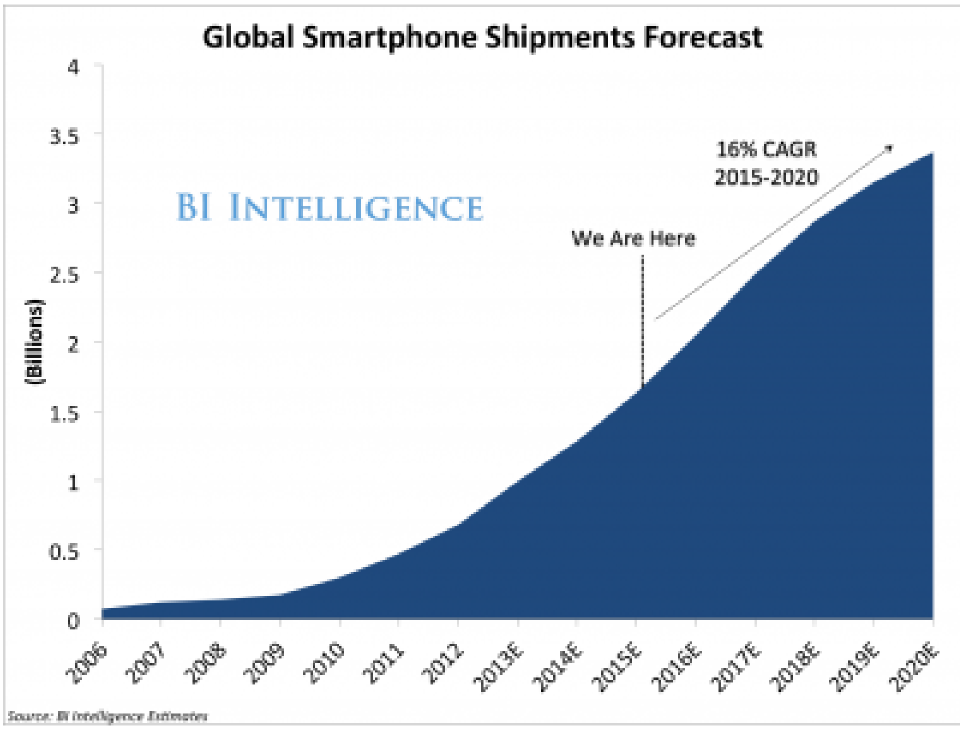
2 Technology

This chapter expose how I am going to discover geographic data visualization in a 3D virtual reality world by presenting some of the main technologies used in this project, along with the reason why they are suited to my aims.

2.1 Android Phone

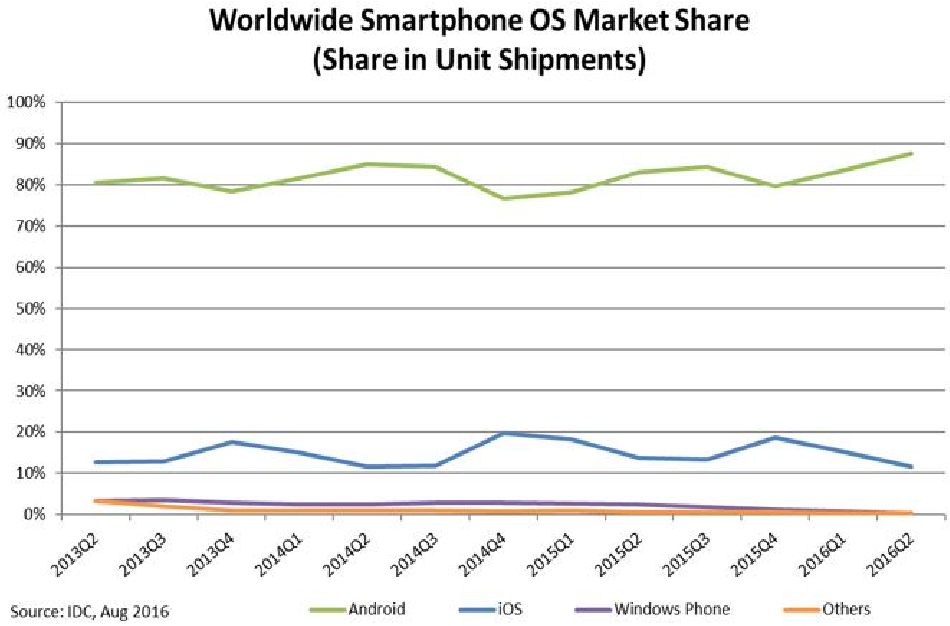
There are reasons we implemented this project on the Android device. We intent to build the virtual reality on a common and familar device in the world. Smartphone is fully deserve the title that not only it has a incredible fast growth trend in the last decade and a good promising prospect, but also they have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful to monitor three-dimensional device movement or positioning.

FIGURE 2.1: Global Smartphone Shipments Forecase (Danova, 2015)



According to data from the International Data Corporation (IDC), Android dominated the smartphone market with a share of 87.6% in the worldwide.

FIGURE 2.2: Smartphone OS Market Share (IDC, 2016)



Moreover, the perfect part is the Google VR SDK (Google, 2016d) for Android supports and the affordable Cardboard product (Google, 2016c) designed for different kind of mobile devices.

2.2 OpenGL ES

Android includes support for high performance 2D and 3D graphics with the Open Graphics Library, specifically, the OpenGL ES API (google, 2016). OpenGL ES is a flavor of the OpenGL specification intended for embedded devices. Android supports several versions of the OpenGL ES API:

TABLE 2.1: OpenGL ES API specification supported by Android

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

2.3 Keyhole Markup Language

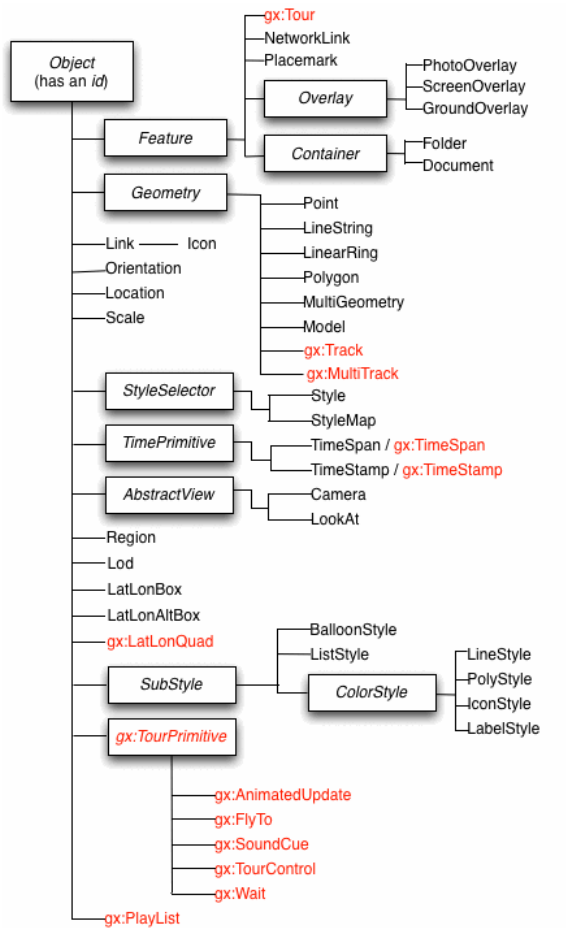
We were looking for a simple markup languages that we can publish and consume data in interoperable formats without the need for technical assistance.

In this section, we present Keyhole Markup Language (KML) (Google, 2016e), a file format to display geographic data ((Note that KML files can be combined with other supporting files such as imagery in a zip archive, producing a KMZ file). It is an international standard maintained by the Open Geospatial Consortium, Inc. (OGC), and also it is supported by many Virtual Globes (VG) applications and other GIS systems and is therefore already becoming a de facto standard. Such as the most wellknown VG application Google Earth that has the largest community, NASA World Wind has more focus toward scientific users, and ArcGIS Explorer that is a lightweight client to the ArcGIS Server (Blower et al., 2007).

From an environmental science point of view, KML is a somewhat limited language. It describes only simple geometric shapes on the globe (points, lines and polygons) and is not extensible. It is, in many respects, analogous to Geography Markup Language (GML) 3.0+ is much more sophisticated and allows the rich description of geospatial features such as weather fronts and radiosonde profiles. So, KML is currently not suitable as a fully-featured, general-purpose environmental data exchange format.

Figure 2.3 shows the KML schema. From the point of view of usability, KML spans a gap between very simple (e.g. GeoRSS) and more complex (GML) formats, that makes it easy for non-technical scientists to share and visualize simple geospatial information which can then be manipulated in other applications if required. Also it makes it easier for user to visualize their data quickly using a single, simple data file. Moreover, its rapidly-growing adoption by scientists, and it is important to be aware of that virtual geographic data visualization (and KML) do not attempt to replace more sophisticated systems.

FIGURE 2.3: KML schema (Google, 2016e)



2.4 Network

The key strengths of virtual reality applications is not only easy-to-use, and intuitive nature, but also the ability to incorporate new data very easily. Therefore, real-time data are very important in the environmental sciences (Blower et al., 2007). To do that, a web server is needed. In this project, we implemented a RESTful web server to support communication with client, along with a file server to synchronize data. In the client side.

Go (often referred to as golang (Google, 2016a)) 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 we were seeing developing software infrastructure (Google, 2012). Also, it growing fast that each month the contributors outside Google is already more then contributors inside the Go team.

We are using Go to build the server, it is well suited for developing RESTful API's. The net/http standard library provides key methods for interacting via the http protocol. On the other hand, since our client is Android phone, we introduced Volley for transmitting network data (Volley is an open sourced HTTP library that makes networking for Android apps easier and most importantly, faster (Google, 2016b)), and jsoup (Java HTML Parser (jsoup, 2016)) for analysing HTML format response.

3 Implementation

This chapter presents more details of the key implementation in the project.

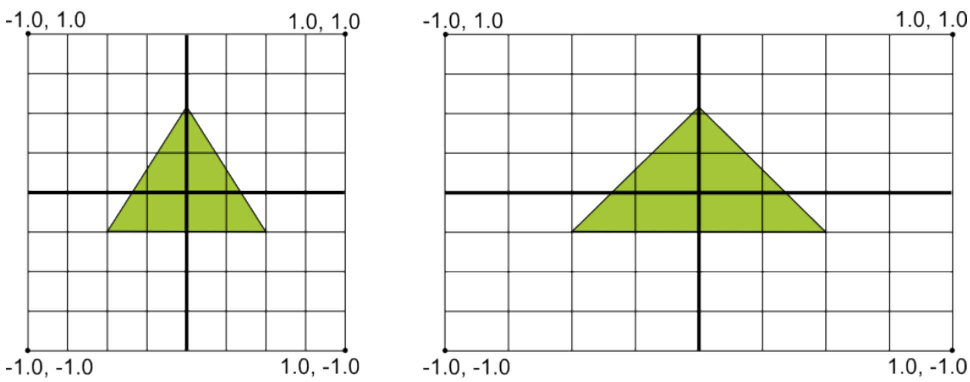
3.1 Google VR SDK

The Google VR SDK repository is free and accessible from *https://github.com/googlevr/gvr-android-sdk*, where we can get the necessary libraries and examples. The SDK libraries are in the libraries directory of the repository as *.aar* files (Google, 2016a). This project has two dependencies on *base* and *common* modules.

3.2 OpenGL ES

OpenGL assumes a square, uniform coordinate system and, by default, happily draws those coordinates onto your typically non-square screen as if it is perfectly square. But the problem is that screens can vary in size and shape:

FIGURE 3.1: Default OpenGL coordinate system (left) mapped to a typical Android device screen (right) (google, 2016)



The illustration above shows the uniform coordinate system assumed for an OpenGL frame on the left, and how these coordinates actually map to a typical device screen in landscape orientation on the right. To solve this problem, you can apply OpenGL projection modes and camera views to transform coordinates so your graphic objects have the correct proportions on any display.

In order to apply projection and camera views, you create a projection matrix and a camera view matrix and apply them to the OpenGL rendering pipeline. The projection matrix recalculates the coordinates of your graphics so that they map correctly to Android device screens. The camera view matrix creates a transformation that renders objects from a specific eye position.

TABLE 3.1: OpenGL Compute

What	How	Where
Model Matrix	translationMatrix * scaleMatrix * rotationMatrix * matrix(1)	CPU
Camera Matrix	Matrix.setLookAtM(positionV, lookAtV, upV)	CPU
View Matrix	eye.getEyeView() * cameraM	CPU
Perspective Matrix	eye.getPerspective(zNear, zFar)	CPU
Projection Matrix	perspectiveM * viewM * modelM	GPU
Vertex′	projectionM * vertex	GPU

3.3 Server

As mentioned in 2.4, Go is well suited and super easy for developing network. A simple localhost file server on port 8080 to serve a directory on disk (/tmp) under an alternate URL path (/files/), use StripPrefix to modify the request URL's path before the FileServer sees it.

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

For RESTful APIs, we introduce a free framework Go-Json-Rest (ant0ine, 2016), 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 easily.

Note that, a file server satisfied all requirement from client at this moment. Alrough the RESTful is setup, but there is no RESTful APIs is actually in use yet.

3.3.1 Assets

Following is the folder structure served by file server:

TABLE 3.2: Assets Structure

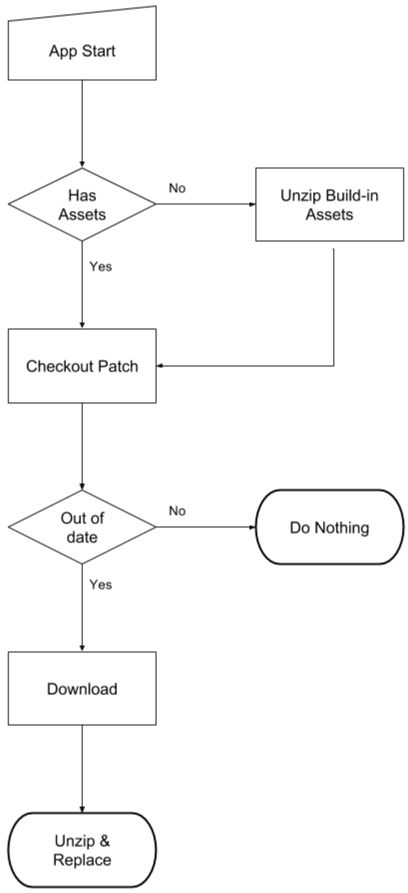
Path	Usage
\assets	Root
\assets\static.zip	Patch (see 3.3.2) compressed file
\assets\static\kml	KML (see 3.4.1) storage
\assets\static\layer	KML storage (see 3.4)
\assets\static\model	OBJ model (see 3.6.3) storage
\assets\static\resource	Image storage

3.3.2 Patch

Patch check is happening everytime when app start. First of all, client checkout the patch file (*\assets\static.zip*) from file server, comparing the*lastModifiedTime* with local patch file, and only continue to download if local patch out of date. Once the patch file is downloaded, replace any existing files.

Note that a build-in default patch is included in the apk (Android app binary) in case of client disconnect from internet during the first time launch time that no available data should be avoid.

FIGURE 3.2: Patch Check



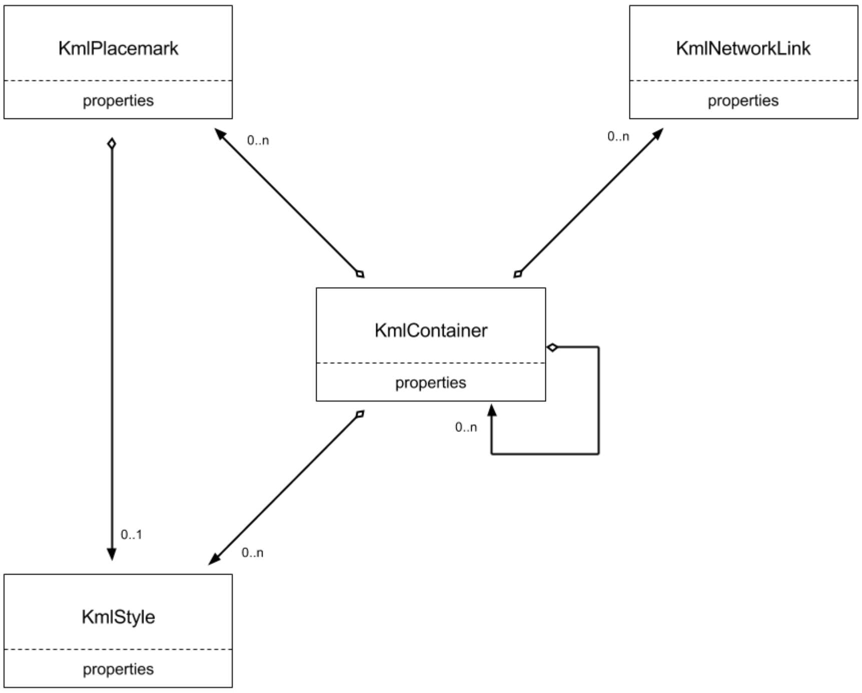
3.4 Scene

A layer list shows available KML files from *\assets\static\layer*, scene will be created according to which KML is selected. The KMLs in *\assets\static\layer* is literally same as KMLs in *\assets\static\kml*, only different is that user can only able to see laye that achieving the idea of KML categorization by KML NetworkLink feature (see 3.4.1). The NetworkLink feature allows a KML file (*\assets\static\layer*) includes one or more KMLs (*\assets\static\kml*).

3.4.1 Keyhole Markup Language

In this project, we only take use of few feature of KML 2.3: Container, Style, Placemark, and NetworkLink. The KML parser we are using is based on the open-source library `android-maps-utils` (Google, 2016b) (NetworkLink is one of the unsupported feature in the library). Main modifications are getting rid of `com.google.android.gms.maps.GoogleMap` dependency, and extending `NetworkLink` feature support in accordance with the current design pattern.

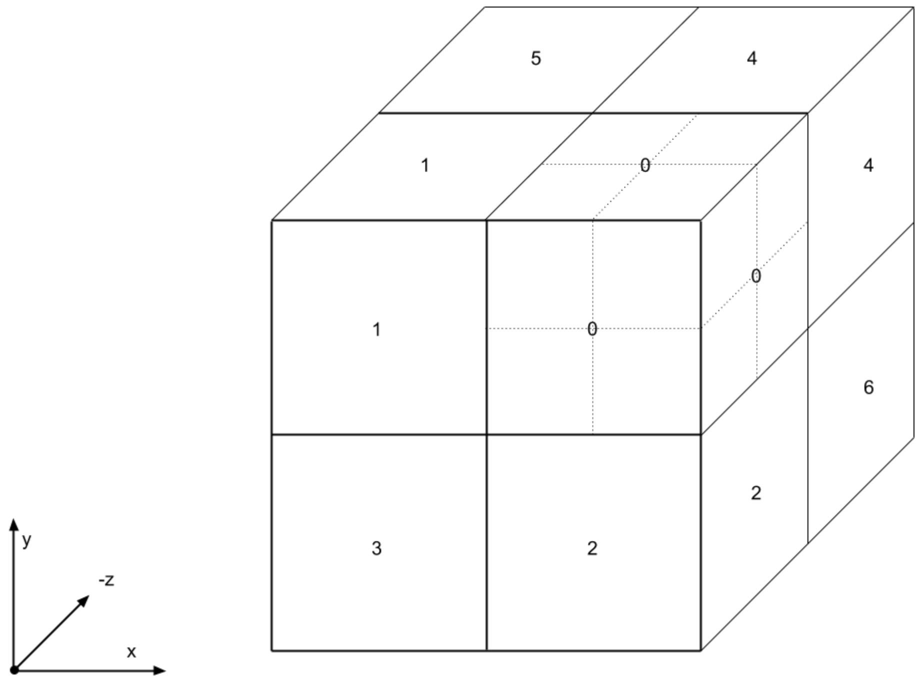
FIGURE 3.3: kML parser simple



3.4.2 Octree

To reduce the ray-object intersection tests, space partitioning is needed. The main requirement is not to use a spatial data structure for an ray intersect with irregular geometry, but to determine what objects are in the same cell to avoid doing an n^2 check on all objects. In this case, it is spherical placemark. Therefore, we don't need overlapped volumes, and contained objects don't need to be cut on volume boundaries. It is actually 3D space partitioning process with a predefined restricted maximum number of objects in the same cell. A simple axis-aligned Octree is fully satisfy in here 3.4.

FIGURE 3.4: Octree split



For each box splitting process, we also generate eight indexes to indicate the relative position inside the box. These indexes are importent for the next partition, where we might need to relink contained objects to the new corresponding box. To insert a object into the box only if the existed contained number of objects is less then the predefined constant value, otherwise splitting the box then relink existed object and insert the new object again.

Integer indexes of box is defined by three boolean valuse that indicates three axis-relative value:

Any position P in the box with known center$O$:

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

TABLE 3.3: Octree Octant

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

Octant solution:

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

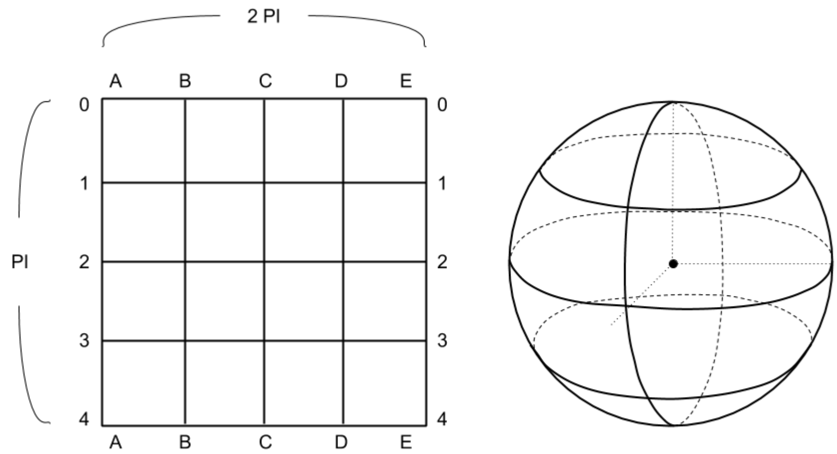
Index solution:

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

3.5 Earth

The Earth is created as a UV Sphere, which somewhat like latitude and longitude lines of the earth, uses rings and segments. Near the poles (both on the Z-axis with the default orientation) the vertical segments converge on the poles. UV spheres are best used in situations where you require a very smooth, symmetrical surface.

FIGURE 3.5: UV sphere mapping

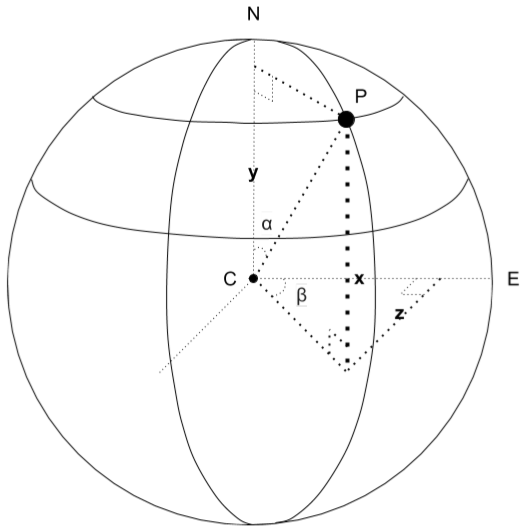


As we can see the mapping from 3.5. Vertex $A0,\ A1,\ A2,\ A3,\ A4$ and $E0,\ E1,\ E2,\ E3,\ E4$ are duplicated, and $A0,\ B0,\ C0,\ D0,\ E0$ converge together as well as $A4,\ B4,\ C4,\ D4,\ E4$. So we can simply define it as a UV sphere has 5 rings and 4 segments. Also be noticed that each ring spans $2\,\pi$ radians, but each segment spans π radians in the sphere mapping.

The total vertex number is:

$$Vertices = Rings \times Segments \tag{3.1}$$

FIGURE 3.6: UV sphere vertex



For each vertex P on sphere from ring r and segment s, we have:

$$v = r \times \frac{1}{rings-1}$$
$$u = s \times \frac{1}{segments-1}$$
$$\measuredangle\alpha = v \times \pi$$
$$\measuredangle\beta = u \times 2\,\pi$$

$\therefore$ P (x, y, z)

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

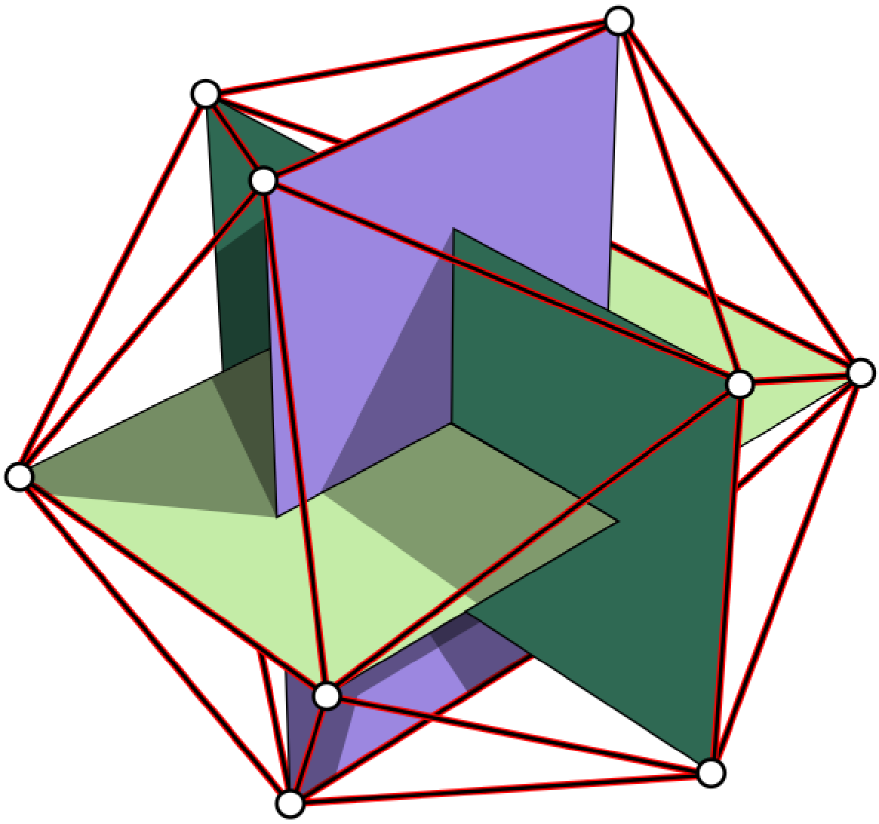
& 2D Texture (x, y) mapping for vertex P is:

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

3.6 Placemarker

Generation of vertices for placemarker is a recursion process of subdividing icosphere. Figure 3.7 shows that the initial vertices of an icosahedron are the corners of three orthogonal rectangles.

FIGURE 3.7: Icosahedron rectangles (Fropuff, 2006)



Rounding icosphere by subdividing a face to an arbitrary level of resolution. One face can be subdivided into four by connecting each edge's midpoint.

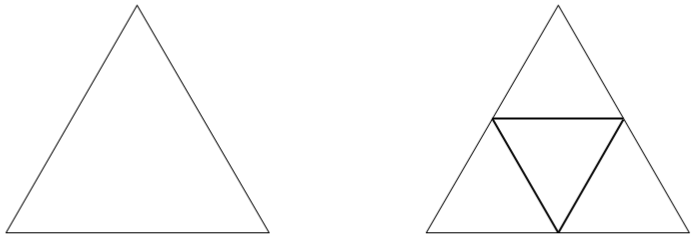


FIGURE 3.8: Icosphere subdivide

Then, push edge's midpoints to surface of the sphere.

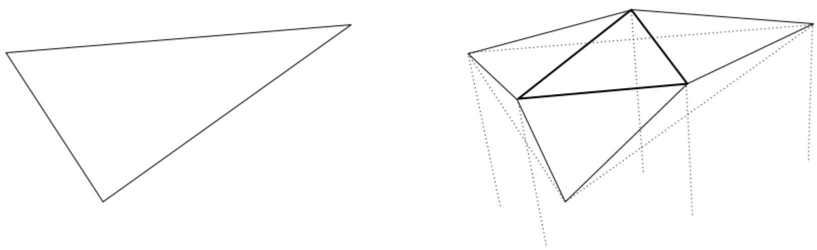


FIGURE 3.9: Icosphere refinement

TABLE 3.4: Rounding Icosphere

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

3.6.1 Geographic Coordinate System

A geographic coordinate system is a coordinate system that enables every location on the Earth to be specified by a set of numbers or letters, or symbols (Wikipedia, 2016c). A common geodetic-mapping coordinates is latitude, longitude and altitude (LLA), which also is the raw location data read from KML.

We introduce ECEF ("earth-centered, earth-fixed") coordinate system for converting LLA coordinates to position coordinates. According to, the z-axis is pointing towards the north but it does not coincide exactly with the instantaneous earth rotational axis. The x-axis intersects the sphere of the earth at 0 latitude and 0 longitude (Wikipedia, 2016b).

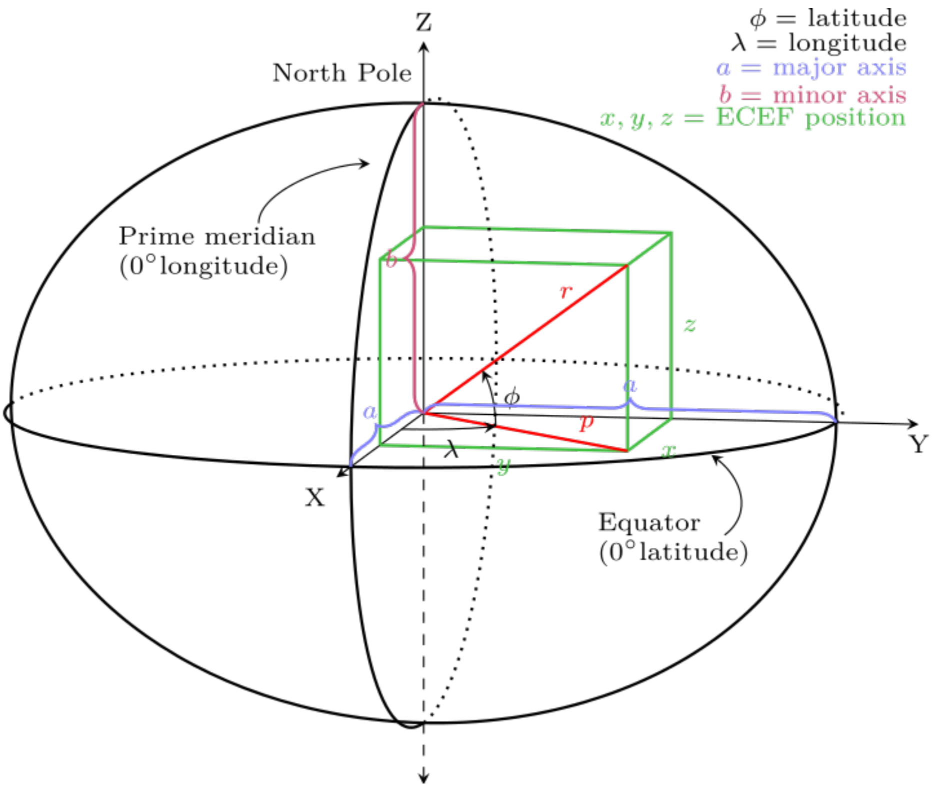


FIGURE 3.10: earth-centered, earth-fixed (Wikipedia, 2016b)

The ECEF coordinates are expressed in a reference system that is related to mapping representations. Because the earth has a complex shape, a simple, yet accurate, method to approximate the earth's shape is required. The use of a reference ellipsoid allows for the conversion between ECEF and LLA (blox, 1999).

A reference ellipsoid can be described by a series of parameters that define its shape and which include a semi-major axis (a), a semi-minor axis (b), its first eccentricity (e_1) and its second eccentricity (e_2) as shown in Table 3.5.

TABLE 3.5: WGS 84 parameters

Parameter	**Notation**	**Value**
Reciprocal of flattening	$1/f$	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	$1-b^2/a^2 = 2\,f - f^2$
Second eccentricity squared	e_2^2	$a^2/b^2 - 1 = f\,(2-f)/(1-f)^2$

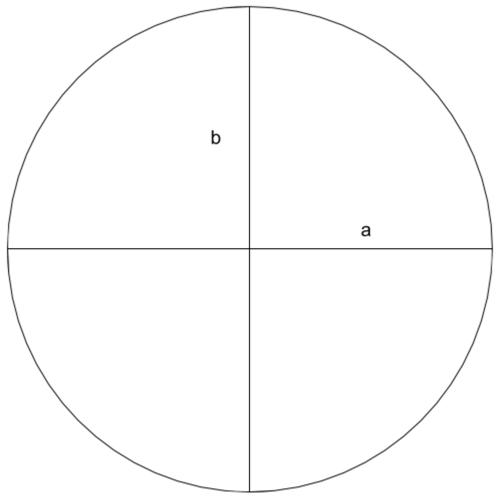


FIGURE 3.11: Ellipsoid Parameters

The conversion from LLA to ECEF is shown below.

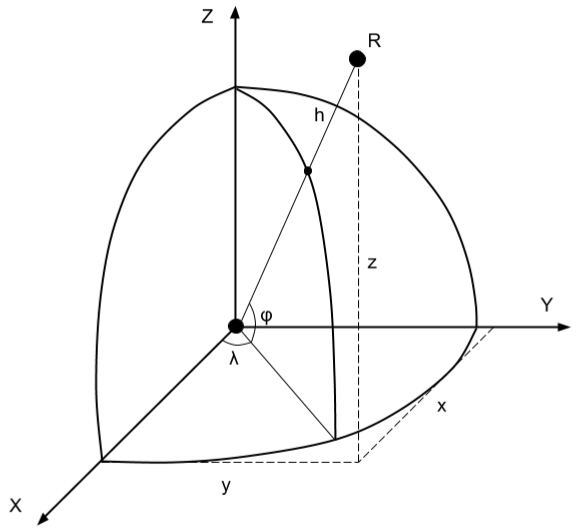


FIGURE 3.12: LLA to ECEF

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

Where

$$
\begin{aligned}
\varphi &= \text{latitude} \\
\lambda &= \text{longitude} \\
h &= \text{height above ellipsoid (meters)} \\
N &= \text{Radius of Curvature (meters), defined as:} \\
&= \frac{a}{\sqrt{1-e^2\ \sin(\varphi)^2}}
\end{aligned}
$$

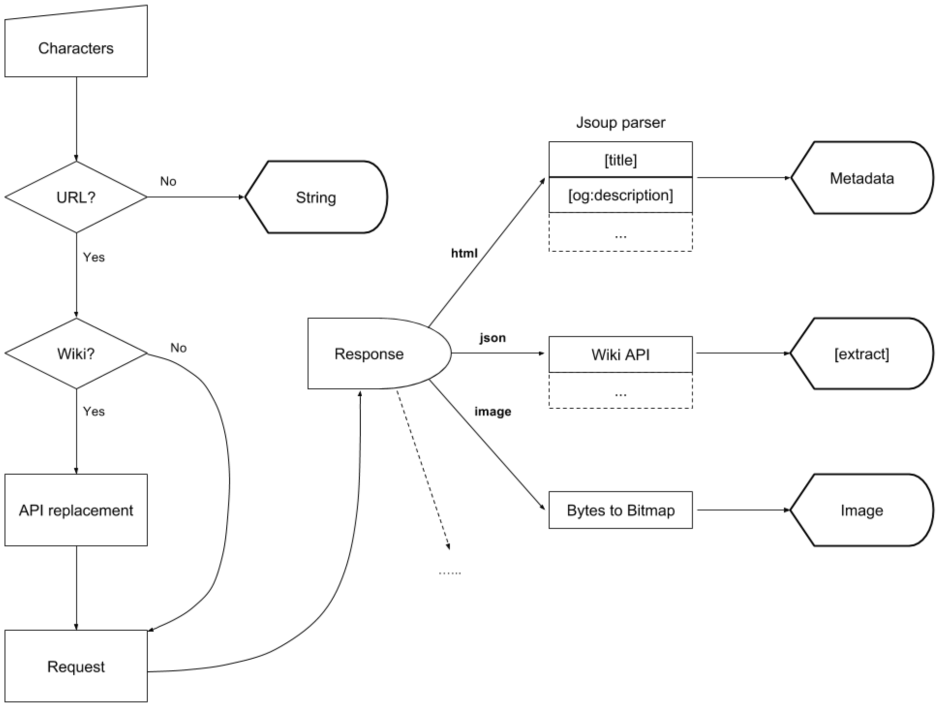
At last, for this project usage, where high accuracy is not required, a equals to b. And also the ECEF coordinate system is y-east, z-north (up), and x points to 0 latitude and 0 longitude, but for project specific, we still need to convert ECEF to x-east, y-north (up), and x points to 0 latitude and 180 longitude.

3.6.2 Description

Description of placemarker requires an appropriate analysis for display. The raw data of description is a set of characters that could be a normal text, an image URL, an URL returns different type of content, or maybe just some meaningless characters.

Althrough the implementation of analysis in this project did not cover every situation, but it is flexible and extendable for more functionality.

FIGURE 3.13: Description Analysis



In order to get an extracted content from a wikipedia page, we can transform the URL to a Wiki-API based open-search url (Wikipedia, 2016a), which will returns a json format raw data that we can easily get what we need from different json tags.

Replace `.wikipedia.org/wiki/`
To `.wikipedia.org/w/api.php?`$APIs$

Where $APIs$ is:

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

For $html$ parser, we introduced jsoup (it is a Java library for working with real-world HTML (jsoup, 2016)), to get the basic information we need, such as $title$, and some other metadata. In this project, I am also use $og : description$ (one of the open graph meta tags (ogp, 2014)) from the html source if it exist.

3.6.3 OBJ Model

A simple and common OBJ format model can be loaded as an extra model for the placemarker. OBJ model can be generated by Blender (Blender, 2016). A simple

OBJ parser is created only support v (vertex indices), vn (vertex normals), fv (face vertex), fvn (face vertex normals), and MTL syntax is ignored (hwshen, 2011).

3.7 Information Display

A textfield is a a rectangle vertices based renderable component to display text on a flat plane. Since it is a GL scene, the actual text will be drawn as a texture. By a constant width and native `android.text.StaticLayout` support, the height of the texture can be calculated.

A menu contains multi-textfield can be seen as an empty textfield based which texture is fill full a pure background color, and several textfields are laid out on the top of it with a certain vertical dimension.

A head rotation matrix (quaternion matrix (Verth, 2013)) is required for locating object in front of camera (mathworks, 2016).

3.8 Camera Movement

In general, there are two sensors can be useful to manager camera movement: ACCELEROMETER (API level 3), LINEAR_ACCELERATION (API level 9) and STEP_DETECTOR (API level 19).

LINEAR_ACCELERATION is same as ACCELERATION which measures the acceleration force in meter per second repeatedly, except linear acceleration sensor is a synthetic sensor with gravity filtered out.

$$
\begin{aligned}
&LinearAcceleration = AccelerometerData - Gravity \\
&v = \int a\, dt \\
&x = \int v\, dt
\end{aligned}
$$

First of all, we take the acclerometer data and remove gravity that is called gravity compensation, whatever is left is linear movement. Then we have to integrate it one to get velocity, integrated again to get position, which is called double integral. Now if the first integral creates drift, double integrals are really nasty that they create horrible drift. Because of these noise, using acceleration data it isn't so accurate, it is really hard to do any kind of linear movement (GoogleTechTalks, 2010).

On the other hand, use step counter from STEP_DETECTOR, and pedometer algorithm for pedestrian navigation, that in fact works very well for this project.

$$
\begin{aligned}
&p_1 = p_0 + v_0 \times dt \\
&v_1 = v_0 + a \times dt
\end{aligned}
$$

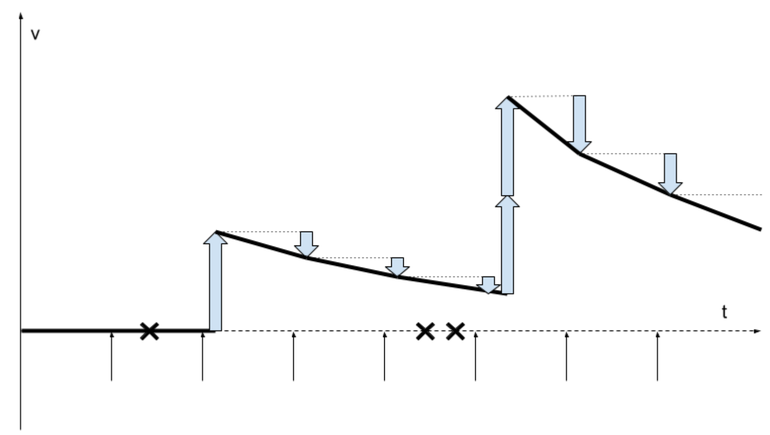
The accuracy of this depends on how precision we can get for changing velocity. Considering that velocity is made of 3-axis directions, the current heading direction is required for a correct velocity calculation. Since the frame life cycle is implemented based on (Google, 2016d), which provide the heading direction in each frame callback. So I collect everything I need from the last frame to new frame, and update both velocity and position for each new frame.

For updating process, first of all,

First of all, damping is required. I reduce velocity by a percentage. It is simply for avoiding that camear taking too long to stop. Damping by percentage can stable and stop the camera in a certain of time that won't be affected by the current camera speed.

Secondly, a constant value in head forwarding direction is been used as a pulse for each step. Because a step is happening instantaneously which implies $a\,dt$ made by each step is actually can be replace by a constant value.

FIGURE 3.14: Camera movement



For each new frame:

$$\begin{aligned}\overrightarrow{V_0} &= \overrightarrow{V_0} \cdot Damping\\ \overrightarrow{P_1} &= \overrightarrow{P_0} + \overrightarrow{V_0} \cdot dt\\ \overrightarrow{V_1} &= \overrightarrow{V_0} + \overrightarrow{Forwarding} \cdot Pulse \cdot Steps\\ Damping &\in [0,\ \ 1]\\ Pulse &\in [0,\ \ \infty)\end{aligned}$$

3.9 Ray Intersection

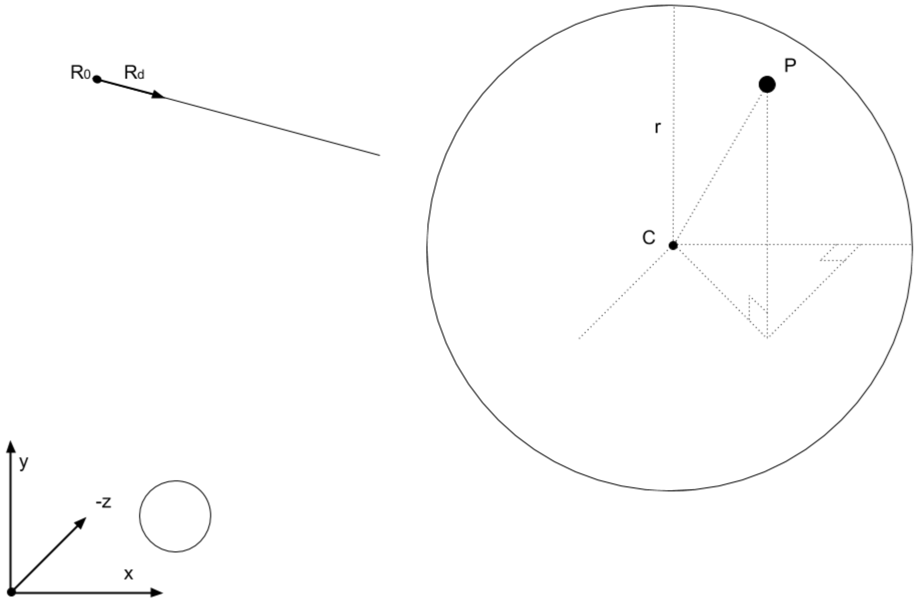
Detect collisions between ray and models is the key to allow user selecting objects in the VR would, which is one of the importent experience for user interaction.

A ray can be describe in a equation with known ray start position $\overrightarrow{R_0}$ and ray direction $\overrightarrow{R_d}$.

$$\overrightarrow{R(t)} = \overrightarrow{R_0} + \overrightarrow{R_d} \cdot t \tag{3.2}$$

3.9.1 Ray-Sphere

FIGURE 3.15: Ray-Sphere intersection



A point P on the surface of sphere should match the equation:

$$(x_p - x_c)^2 + (y_p - y_c)^2 + (z_p - z_c)^2 = r^2 \tag{3.3}$$

If the ray intersects with the sphere at any position P must match the equation 3.2 and 3.3. Therefor the solution of t in the cointegrate equation implies whether or not the ray will intersect with the sphere:

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

It can be seen as a quadratic formula:

$$a\, t^2 + b\, t + c = 0 \tag{3.4}$$

At this point, we are able to solved the t:

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

Then, I take a further step to get rid of formula complexity.

$\because$ Equation 3.3, 3.4

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

&

$$\begin{aligned}
|\overrightarrow{R_d}| &= \sqrt{x_{R_d}^2 + y_{R_d}^2 + z_{R_d}^2} = 1 \\
\overrightarrow{V_{c_R_0}} &= \overrightarrow{R_0} - \overrightarrow{C} = \overrightarrow{\left(x_{R_0} - x_c,\ \ y_{R_0} - y_c,\ \ z_{R_0} - z_c\right)}
\end{aligned}$$

$\therefore$

$$\begin{aligned}
&a = 1 \\
&b = 2 \cdot \overrightarrow{R_d} \cdot \overrightarrow{V_{c_R_0}} \\
&c = \overrightarrow{V_{c_R_0}} \cdot \overrightarrow{V_{c_R_0}} \cdot r^2
\end{aligned}$$

$\because$ The formula for t can also be optimized

$$\begin{aligned}
&\tfrac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a} = -\alpha \pm \sqrt{\beta} \\
&\alpha = \tfrac{1}{2} b \\
&\beta = \alpha^2 - c
\end{aligned}$$

$\therefore$ The final solution for t

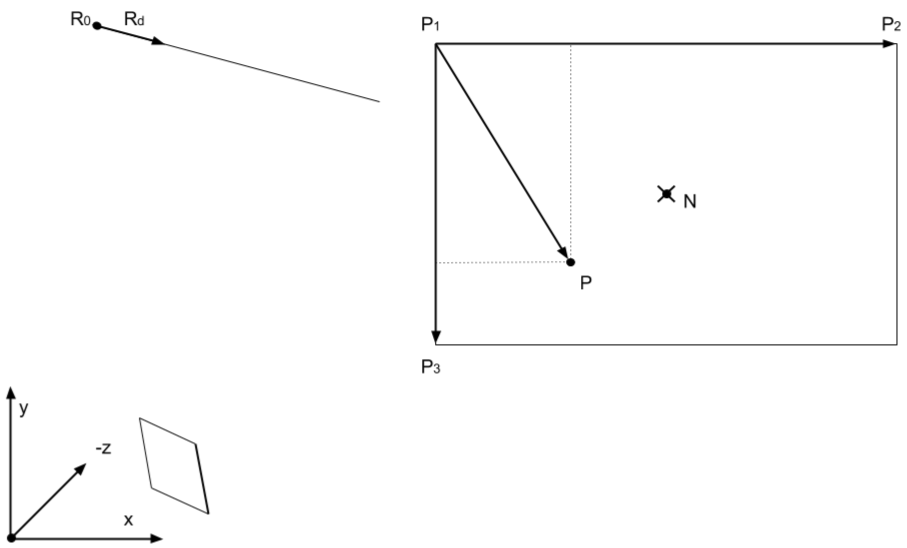
$$t = \begin{cases} -\alpha \pm \sqrt{\beta} & \text{if } \ \beta > 0 \\ -\alpha & \text{if } \ \beta = 0 \\ \varnothing & \text{if } \ \beta < 0 \end{cases}$$

And the collision position for each t is:

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

3.9.2 Ray-Plane

FIGURE 3.16: Ray-Plane intersection



If a point P on the plane and also belongs to the ray, we have quadric equation:

$$\begin{aligned} (\overrightarrow{P} - \overrightarrow{P_1}) \cdot \overrightarrow{N} &= 0 \\ \overrightarrow{P} &= \overrightarrow{R_0} + \overrightarrow{R_d} \cdot t \end{aligned} \tag{3.5}$$

Solution for the t is:

$$t = \begin{cases} \frac{-\overrightarrow{N} \cdot (\overrightarrow{R_0} - \overrightarrow{P_1})}{\overrightarrow{N} \cdot \overrightarrow{R_d}} & \text{if } \overrightarrow{N} \cdot \overrightarrow{R_d} \nsim 0 \\ \varnothing & \text{if } \overrightarrow{N} \cdot \overrightarrow{R_d} \sim 0 \end{cases}$$

At last, we have to verify if the collision is inside of the quadrangle by putting t back to 3.5, (user3146587, 2014) the t is valid only if:

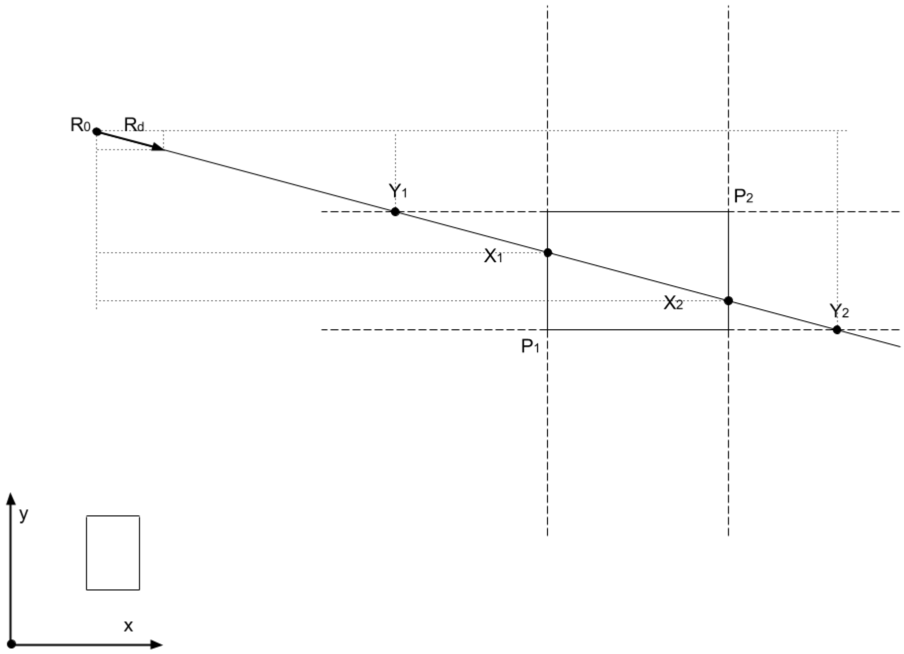
$$\begin{aligned} \mu &= \sqrt{(\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_2} - \overrightarrow{P_1}))} \in [0, \ \ |\overrightarrow{P_2} - \overrightarrow{P_1}|] \\ \nu &= \sqrt{(\overrightarrow{P} - \overrightarrow{P_1}) \cdot (\overrightarrow{P_3} - \overrightarrow{P_1}))}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

3.9.3 Ray-Box

There is a octree implementation 3.4.2 in the VR 3D world that separate the 3D world to invisible 3D boxes that each box contains a certain number of other models. It is to avoid unnecessary ray-object collision detection. In this section, I am going to first explain Ray-Box-2D collision detection (Barnes, 2011), then derive out Ray-Box-3D intersection.

Ray-Box-2D

FIGURE 3.17: Ray-Box-2D intersection



$\because$ Known $R_0,\ R_d,\ P_1,\ P_2$

$$X_1 = \begin{cases} x_{P_1} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_2} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad Y_1 = \begin{cases} y_{P_1} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_2} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

$$X_2 = \begin{cases} x_{P_2} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_1} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad Y_2 = \begin{cases} y_{P_2} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_1} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

$$t_{X_1} = \frac{X_1}{x_{R_d}} \qquad t_{Y_1} = \frac{Y_1}{y_{R_d}}$$
$$t_{X_2} = \frac{X_2}{x_{R_d}} \qquad t_{Y_2} = \frac{Y_2}{y_{R_d}}$$

$\&$ When collision happens, we have formula:

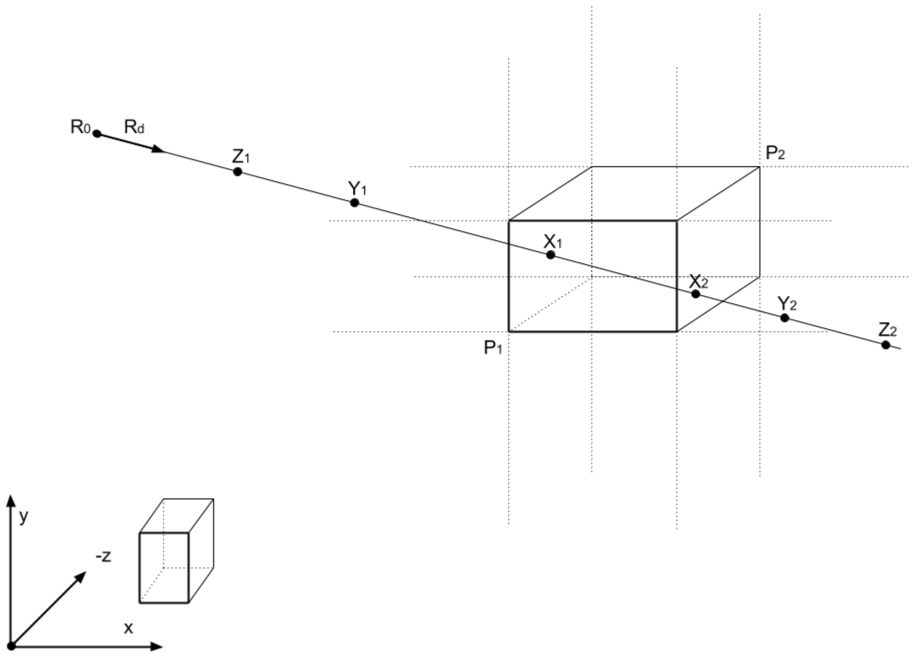
$$t_{X_1} < t_{X_2}$$
$$t_{Y_1} < t_{Y_2}$$

$\therefore$ Which is

$$max(t_{X_1},\ t_{Y_1}) < min(t_{X_2},\ t_{Y_2}) \tag{3.6}$$

Ray-Box-3D

FIGURE 3.18: Ray-Box-3D intersection



$\because$ Known $R_0,\ R_d,\ P_1,\ P_2$

$$X_1 = \begin{cases} x_{P_1} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_2} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad Y_1 = \begin{cases} y_{P_1} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_2} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

$$X_2 = \begin{cases} x_{P_2} - x_{R_0} & \text{if } x_{R_d} > 0 \\ x_{P_1} - x_{R_0} & \text{if } x_{R_d} < 0 \end{cases} \qquad Y_2 = \begin{cases} y_{P_2} - y_{R_0} & \text{if } y_{R_d} > 0 \\ y_{P_1} - y_{R_0} & \text{if } y_{R_d} < 0 \end{cases}$$

$$t_{X_1} = \frac{X_1}{x_{R_d}} \qquad t_{Y_1} = \frac{Y_1}{y_{R_d}}$$
$$t_{X_2} = \frac{X_2}{x_{R_d}} \qquad t_{Y_2} = \frac{Y_2}{y_{R_d}}$$

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

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

$$t_{Z_1} = \frac{Z_1}{z_{R_d}}$$
$$t_{Z_2} = \frac{Z_2}{z_{R_d}}$$

$\&$ When collision happens, we have formula:

$$\begin{cases} t_{X_1} < t_{X_2} \\ t_{Y_1} < t_{Y_2} \\ t_{Z_1} < t_{Z_2} \end{cases}$$

$\therefore$ Which is

$$max(t_{X_1},\ t_{Y_1},\ t_{Z_1}) < min(t_{X_2},\ t_{Y_2},\ t_{Z_2}) \tag{3.7}$$

4 Discussion

****

compare to others. etc. this allows to do similar things, google earth etc...
this, strength, limitation

5 Conclusion

****

outcomes;findings; pass on to ...
2d and 3d env...
vr can .... it explores.....
might apply to other data, not only earth geo d. eg, other natruel sys..

A Appendix A

Write your Appendix content here.

Bibliography

ant0ine (2016). *Go-Json-Rest*. URL: https://github.com/ant0ine/go-json-rest.

Barnes, Tavian (2011). *FAST, BRANCHLESS RAY/BOUNDING BOX INTERSECTIONS*. URL: https://tavianator.com/fast-branchless-raybounding-box-intersections.

Blender (2016). *Blender*. URL: https://www.blender.org/.

Blower, Jonathan David et al. (2007). "Sharing and visualizing environmental data using Virtual Globes". In:

blox, u (1999). *Datum Transformations of GPS Positions*. URL: http://www.nalresearch.com/files/Standard%20Modems/A3LA-XG/A3LA-XG%20SW%20Version%201.0.0/GPS%20Technical%20Documents/GPS.G1-X-00006%20(Datum%20Transformations).pdf.

Danova, Tony (2015). *THE GLOBAL SMARTPHONE MARKET REPORT*. URL: http://www.businessinsider.com.au/global-smartphone-market-forecast-vendor-platform-growth-2015-6?r=US&IR=T.

Fropuff, Mysid (2006). *Icosahedron Golden Rectangles*. URL: https://commons.wikimedia.org/wiki/File:Icosahedron-golden-rectangles.svg.

Google (2012). *Go at Google: Language Design in the Service of Software Engineering*. URL: https://talks.golang.org/2012/splash.article.

– (2016a). *AAR Format*. URL: http://tools.android.com/tech-docs/new-build-system/aar-format.

– (2016b). *android-maps-utils*. URL: https://github.com/googlemaps/android-maps-utils/tree/master/library/src/com/google/maps/android/kml.

– (2016c). *Google Cardboard*. URL: https://vr.google.com/cardboard/.

– (2016d). *Google VR SDK for Android*. URL: https://developers.google.com/vr/android/.

– (2016e). *Keyhole Markup Language*. URL: https://developers.google.com/kml/.

google (2016). *OpenGL ES*. URL: https://developer.android.com/guide/topics/graphics/opengl.html.

Google (2016a). *The Go Programming Language*. URL: https://golang.org/.

– (2016b). *Transmitting Network Data Using Volley*. URL: https://developer.android.com/training/volley/index.html.

GoogleTechTalks (2010). *Sensor Fusion on Android Devices: A Revolution in Motion Processing*. URL: https://www.youtube.com/watch?v=C7JQ7Rpwn2k&feature=youtu.be&t=23m21s.

hwshen (2011). *Guidance to write a parser for .Obj and mtl file*. URL: http://web.cse.ohio-state.edu/~hwshen/581/Site/Lab3_files/Labhelp_Obj_parser.htm.

IDC (2016). *Smartphone OS Market Share*. URL: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

jsoup (2016). *jsoup: Java HTML Parser*. URL: https://jsoup.org/.

mathworks (2016). *Quaternion Rotation*. URL: http://au.mathworks.com/help/aeroblks/quaternionrotation.html.

Mazuryk, Tomasz and Michael Gervautz (1996). "Virtual reality-history, applications, technology and future". In:

ogp (2014). *The Open Graph protocol*. URL: http://ogp.me/.

Tuttle, Benjamin T, Sharolyn Anderson, and Russell Huff (2008). “Virtual globes: an overview of their history, uses, and future challenges”. In: *Geography Compass* 2.5, pp. 1478–1505.

user3146587 (2014). URL: http://stackoverflow.com/questions/21114796/3d-ray-quad-intersection-test-in-java.

Verth, Jim Van (2013). *Understanding Quaternions*. URL: http://www.essentialmath.com/GDC2013/GDC13_quaternions_final.pdf.

Wikipedia (2016a). *API*. URL: https://www.mediawiki.org/wiki/API:Opensearch.

– (2016b). *ECEF*. URL: https://en.wikipedia.org/wiki/ECEF.

– (2016c). *Geographic coordinate system*. URL: https://en.wikipedia.org/wiki/Geographic_coordinate_system.