MASSEY UNIVERSITY

Geographic Data Visualization

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 2, 2016

Abstract

********** overview
propose
What was done?
Why was it done?
How was it done?
What was found?
What is the significance of the findings in short?

Visualization has proven effective for not only presenting essential information in vast amounts of data but also driving complex analyses.

(Tuttle, Anderson, and Huff, 2008)
Three-dimensional (3D) representations of objects and spaces are frequently used to improve human understanding.
The pseudo-3D nature of virtual globes allows people to interact in an environment that they naturally comprehend and that makes the data and information presented easier to understand.
Transportability relates to the fact that virtual globes are based on digital data and, therefore, can be transported as easily as any other digital information.
Scalability is a powerful aspect of virtual globes.
Interactivity is a crucial part of the power of virtual globes. Users are in control of the experience and can adjust the view, scale, data layers, and more.
Choice of topics is an important difference from traditional representations of geospatial data. Virtual globes have the power of combining multiple topics in one media.
Currency is the ability to easily adjust the data available to any given time period. The data can be static or have an update time ranging from years to real time.

The rapid development of virtual globes and their supporting infrastructure of imagery and servers represent a convergence of mainstream information technology and geography. Geography is increasingly becoming a central organizing principal for managing, analyzing, and visualizing the world's information.

(Mazuryk and Gervautz, 1996)
Virtual Reality (VR), sometimes called Virtual Environments (VE) has drawn much attention in the last few years.
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).
At the beginning of 1990s the development in the field of virtual reality became much more stormy and the term Virtual Reality itself became extremely popular.

Undoubtedly VR has attracted a lot of interest of people in last few years. Being a new paradigm of user interface it offers great benefits in many application areas. It 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 (and often clumsy) user interface works. Therefore many applications like flight simulators, architectural walkthrough or data visualization systems were developed relatively fast.
The navigation through the huge amount of data visualized in three-dimensional space is almost as easy as walking.

Three-dimensional objects have six degrees of freedom (DOF): position coordinates (x, y and z offsets) and orientation (yaw, pitch and roll angles for example).

(Blower et al., 2007) Virtual globe technology holds many exciting possibilities for environmental science. These easyto-use, intuitive systems provide means for simultaneously visualizing four-dimensional environmental data from many different sources, enabling the generation of new hypotheses and driving greater understanding of the Earth system. Through the use of simple markup languages, scientists can publish and consume data in interoperable formats without the need for technical assistance. In this paper we give, with examples from our own work, a number of scientific uses for virtual globes, demonstrating their particular advantages. We explain how we have used Web Services to connect virtual globes with diverse data sources and enable more sophisticated usage such as data analysis and collaborative visualization. We also discuss the current limitations of the technology, with particular regard to the visualization of subsurface data and vertical sections.
Recently there have been great advances in the development of "virtual globes": software applications that display a three-dimensional representation of the entire Earth, usually based on satellite imagery, upon which new information can be superimposed.
Google Earth is perhaps the most wellknown VG application currently available. It is a free but closed-source application that is available for Windows, Linux and Mac OSX. The primary method for visualizing data in Google Earth is to create KML files, although there is limited support for other methods. It is aimed at the general public, primarily as a search tool, but has attracted a very large community of people who have used the application for a very wide range of purposes.
NASA World Wind [14] is an open-source VG application that is written in .NET and is therefore only supports Windows operating systems. Future releases will be Java-based and hence cross-platform. World Wind provides access to a wide range of NASA satellite imagery. Data can be imported through tile servers, OGC Web Services and there is limited support for KML. Its focus is toward scientific users, so World Wind has a more specialist community than that of Google Earth.
ArcGIS Explorer [1] can be used as a standalone (free, Windows-only) VG but it is best seen as a lightweight client to the (nonfree) ArcGIS Server. It can import data in a very wide range of GIS formats (including KML) and can perform data analysis on the client (using plug-ins and the associated .NET SDK) or the server (through the interface to ArcWeb Services). At the time of writing this is a very new product and so has not had time to build a large community.
This is because (1) Google Earth is the only VG of the above shortlist that currently works across platforms and thus has the widest potential scientific audience, (2) Google Earth currently possesses the largest community, and (3) Google Earth supports the latest features of KML, many of which are particularly relevant for scientific data.

KML is a simple XML language that is supported by many virtual globes.
Although historically tightly connected with Google Earth, discussions are underway to standardize KML the Open Geospatial Consortium (OGC) [19]. KML is supported by a number of virtual globes and other GIS systems and is therefore already becoming a de facto standard. (Note that KML files can be combined with other supporting files such as imagery in a zip archive, producing a KMZ file. In this paper we intend "KML" to represent both KML and KMZ for clarity.)
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 2.0 [7]. GML 3.x is much more sophisticated and allows the rich description of geospatial features such as weather fronts and radiosonde profiles [22].

Annotations of KML features are intended to be human-readable plain text or simple HTML, not machine-readable XML.
For the above reasons, KML is currently not suitable as a fully-featured, general-purpose environmental data exchange format. Its rapidly-growing adoption by scientists, however, shows that it is finding an important niche. It is important to remember that virtual globes (and KML) do not attempt to replace more sophisticated systems. They make it easy for non-technical scientists to share and visualize simple geospatial information, which can then be manipulated in other applications if required. The objections raised in the previous two paragraphs can actually be seen as advantages from the point of view of usability because they make it easier for users to visualize their data quickly using a single, simple data file. KML is useful because it spans a gap between very simple (e.g. GeoRSS [8]) and more complex (GML 3.x) formats. It is a much more useful interchange format than imagery alone since it holds georeferencing information and allows for the inclusion of links to related information (e.g. the full data archive).

Simultaneous visualization of data from different sources
A key strength of virtual globes is that it is very easy to download data over the Web from a number of different providers and visualize them simultaneously to look for relationships that can be investigated later in a more quantitative fashion. Real-time data are very important in the environmental sciences.
(Near) real-time data can be displayed in Google Earth using the NetworkLink facility in KML. This facility allows all or part of a KML dataset to be refreshed at a fixed rate or based on the expiration time given in the HTTP header. A user can therefore download a fixed KML file containing the NetworkLink and Google Earth will automatically refresh the link, ensuring that the user always sees the latest information.
The key strengths of virtual globe applications are their easy-to-use, intuitive nature and the ability to incorporate new data very easily.
A key requirement for environmental scientists is to be able to visualize four dimensional data (i.e. time-dependent three-dimensional data).

Keywords: Keywords, for, your, thesis

Contents

List of Figures

List of Tables

1 Introduction

****
Why am I doing it?

1.1 Background and Overview

****
vr

1.2 Literature Review

****
What is known?
What is unknown?

1.2.1 Current Landscape

****

Review of research

1.2.2 Respective Limitations

****
Identifying gaps

1.3 Aims and Objectives

****
What do I hope to discover?
What is expected in a thesis?

2 Technology

****
How am I going to discover it?
.... why is best suited to my aims, because ...

2.1 Mobile Device

2.1.1 Sensor

2.1.2 Google Cardboard

2.1.3 Android

2.2 OpenGL ES

****
OpenGL ES 2.0, 3.0, 3.1, 3.1 ext

2.3 Data Format

In this

2.3.1 KML Scene

****
3D scene data format: KML, VRML, COLLADA, glTF

2.3.2 OBJ Model

2.4 File Server

3 Implementation

3.1 Google VR SDK

3.2 OpenGL ES

****
Pass Perspective matrix, View matrix, and Model matrix to shader to avoid calculate MV or MVP in cpu
Perspective: eye.getPerspective(float zNear, float zFar)
View: eye.getEyeView() * camera.matrix
camera.matrix: build by position, lookAt, up
Model: translation * scale * rotation * mat(1)

3.3 Scene

3.3.1 KML

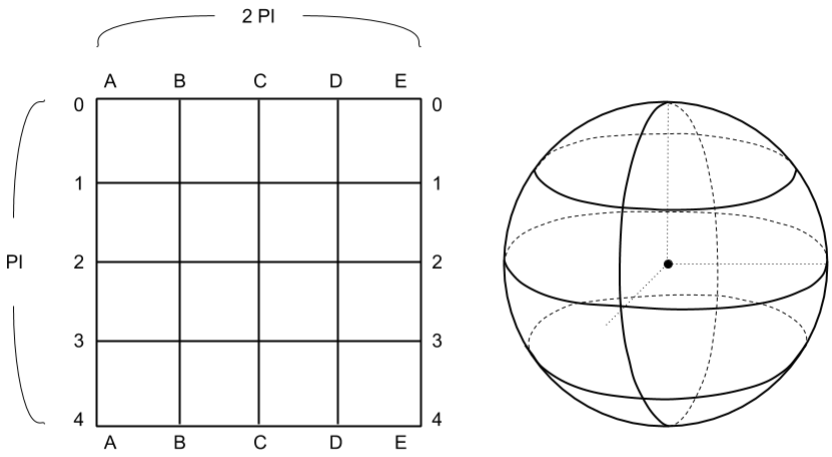
****
kml parser
graph and extended or chanegs

3.3.2 Octree

3.4 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 3.1: UV sphere mapping

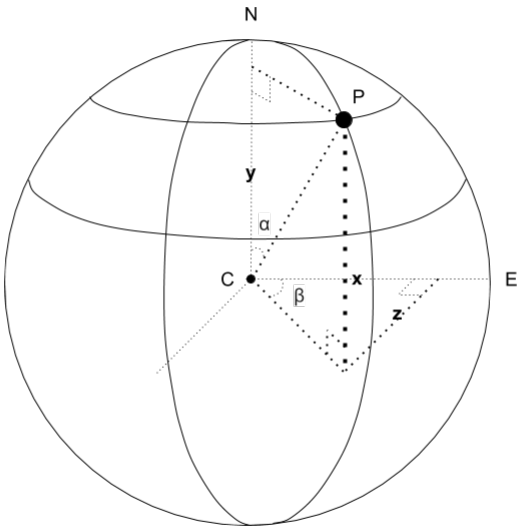


As we can see the mapping from 3.1. Vertex $A0,\ A1,\ A2,\ A3,\ A4$ and $E0,\ E1,\ E2,\ E3,\ E4$ are duplicated, and $A0,\ B0,\ C0,\ D0,\ E0$ converge together as well as $A4,\ B4,\ C4,\ D4,\ E4$. So we can simply define it as a UV sphere has 5 rings and 4 segments. Also be noticed that each ring spans $2\,\pi$ radians, but each segment spans π radians in the sphere mapping.

The total vertex number is:

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

FIGURE 3.2: UV sphere vertex



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

$$
\begin{aligned}
v &= r \times \tfrac{1}{rings-1} \\
u &= s \times \tfrac{1}{segments-1} \\
\measuredangle\alpha &= v \times \pi \\
\measuredangle\beta &= u \times 2\,\pi
\end{aligned}
$$

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

$$
\begin{aligned}
x &= (\sin(\alpha) \times radius) \times \cos(\beta) \\
y &= \cos(\alpha) \times radius \\
z &= (\sin(\alpha) \times radius) \times \sin(\beta)
\end{aligned}
$$

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

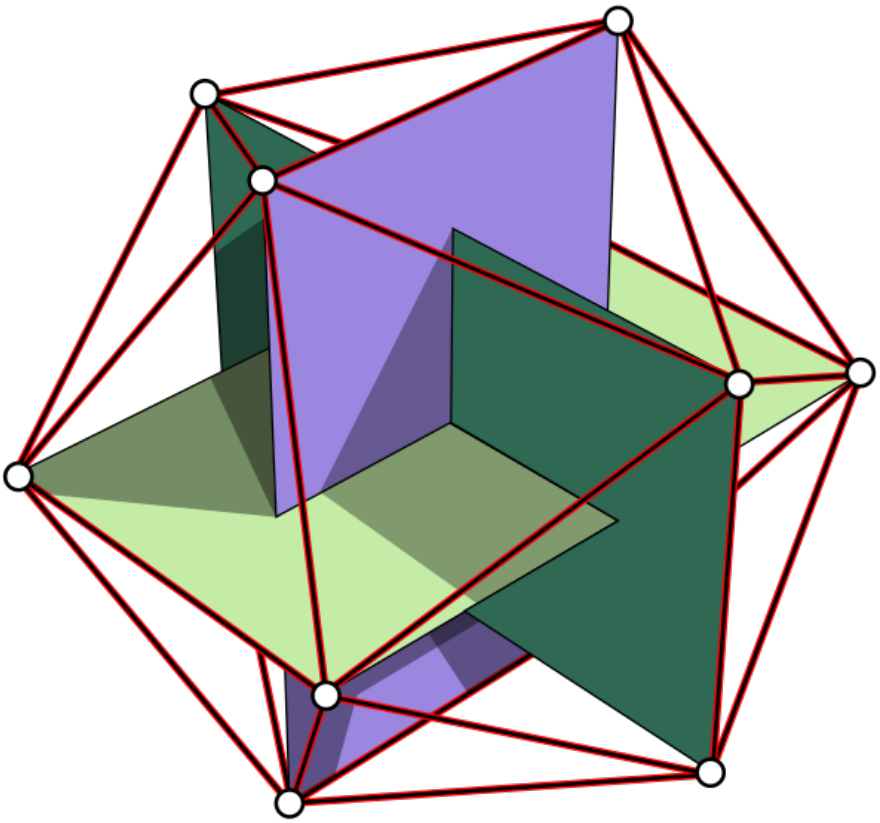
$$
\begin{aligned}
x &= u \\
y &= v
\end{aligned}
$$

3.5 Placemarker

3.5.1 Icosphere

Generation of vertices for placemarker is a recursion process of subdividing icosphere. Figure 3.3 shows that the initial vertices of an icosahedron are the corners of three orthogonal rectangles.

FIGURE 3.3: 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.

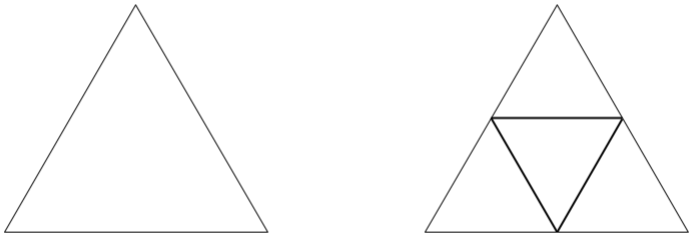


FIGURE 3.4: Icosphere subdivide

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

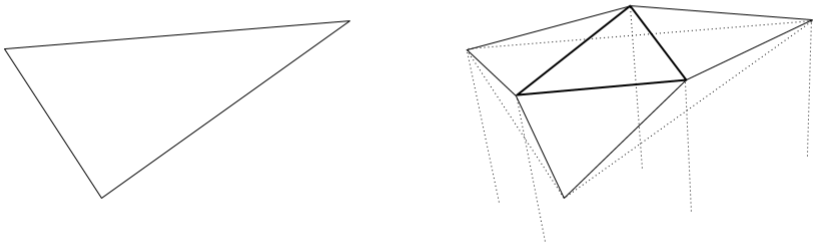


FIGURE 3.5: Icosphere refinement

TABLE 3.1: 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

3.5.2 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).

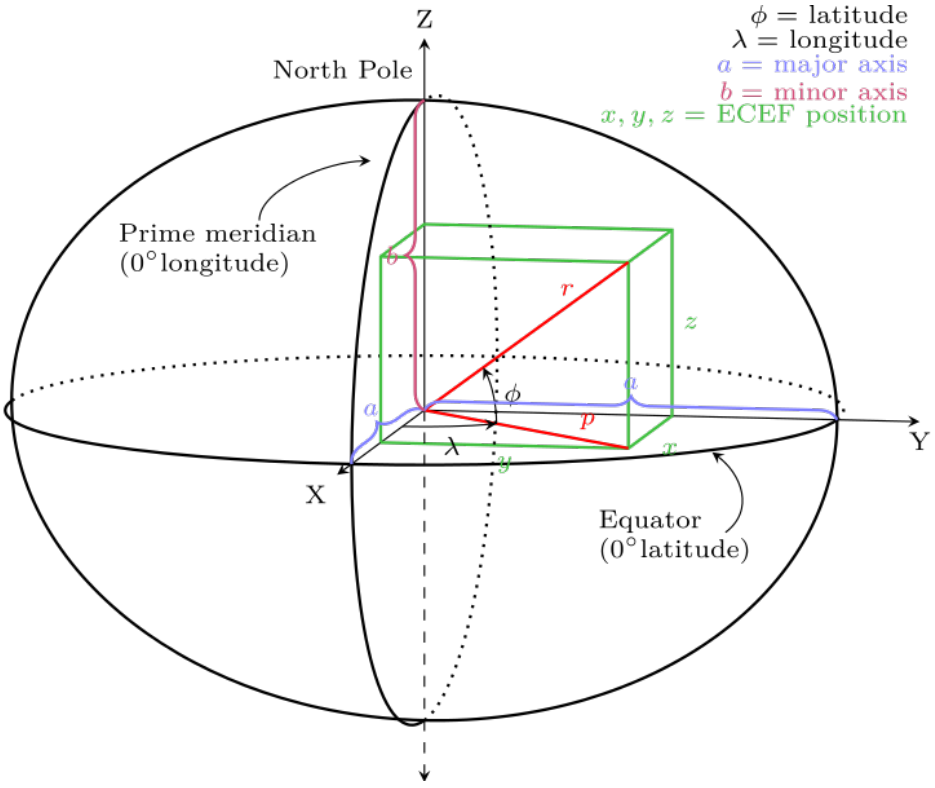


FIGURE 3.6: earth-centered, earth-fixed (Wikipedia, 2016b)

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 3.2.

TABLE 3.2: 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	$a^2/b^2-1 = f\,(2-f)/(1-f)^2$

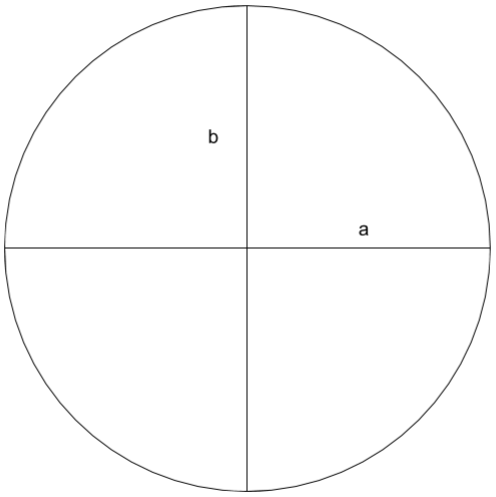


FIGURE 3.7: Ellipsoid Parameters

The conversion from LLA to ECEF is shown below.

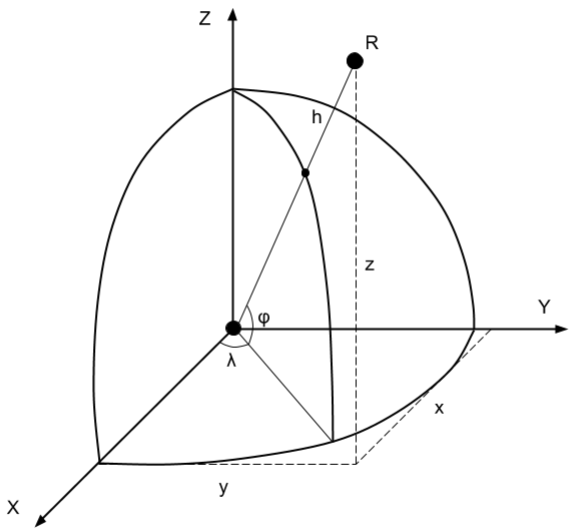


FIGURE 3.8: LLA to ECEF

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

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.

3.5.3 OBJ Model

A simple and common OBJ format model can be loaded as an extra model for the placemarker.

3.6 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) .

****

quaternions

3.7 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.

$$
\begin{aligned}
&Linear Acceleration = Accelerometer Data - Gravity \\
&v = \int a \, dt \\
&x = \int v \, dt
\end{aligned}
$$

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.

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

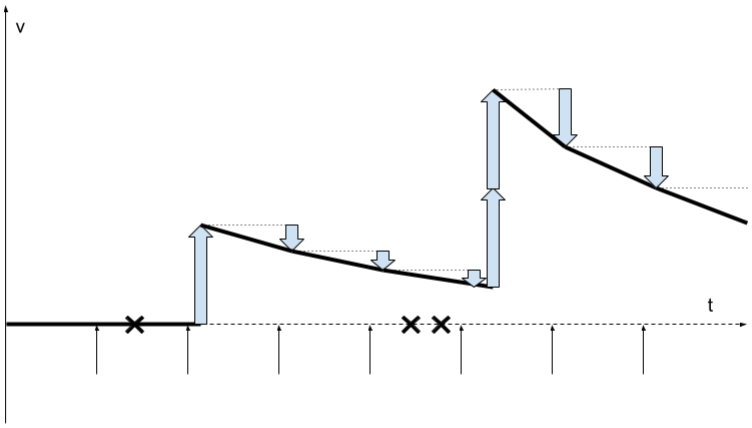
The accuracy of this depends on how precision we can get for changing velocity. Considering that velocity is made of 3-axis directions, the current heading direction is required for a correct velocity calculation. Since the frame life cycle is implemented based on (Google, 2016), which provide the heading 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 3.9: Camera movement



For each new frame:

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

3.8 Ray Intersection

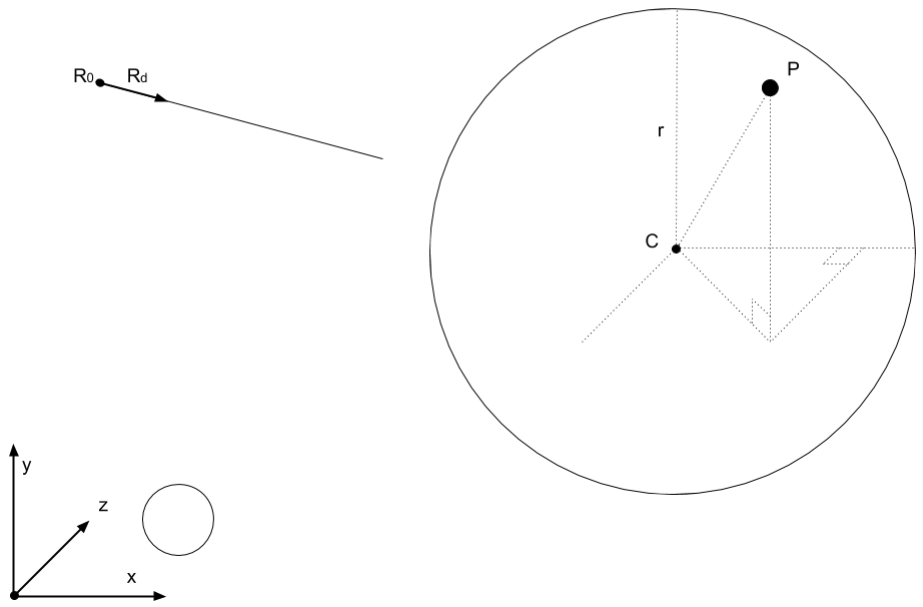
Detect collisions between ray and models is the key to allow user selecting objects in the VR would, which is one of the importent experience for user interaction.

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

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

3.8.1 Ray-Sphere

FIGURE 3.10: 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:

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

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

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

&

$$|\overrightarrow{R_d}| = \sqrt{x_{R_d}^2 + y_{R_d}^2 + z_{R_d}^2} = 1$$
$$\overrightarrow{V_{c\_R_0}} = \overrightarrow{R_0} - \overrightarrow{C} = \overrightarrow{\left(x_{R_0} - x_c,\ \ y_{R_0} - y_c,\ \ z_{R_0} - z_c\right)}$$

$\therefore$

$$\begin{cases} a = 1 \\ b = 2 \cdot \overrightarrow{R_d} \cdot \overrightarrow{V_{c\_R_0}} \\ c = \overrightarrow{V_{c\_R_0}} \cdot \overrightarrow{V_{c\_R_0}} \cdot r^2 \end{cases}$$

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

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

$\therefore$ The final solution for t

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

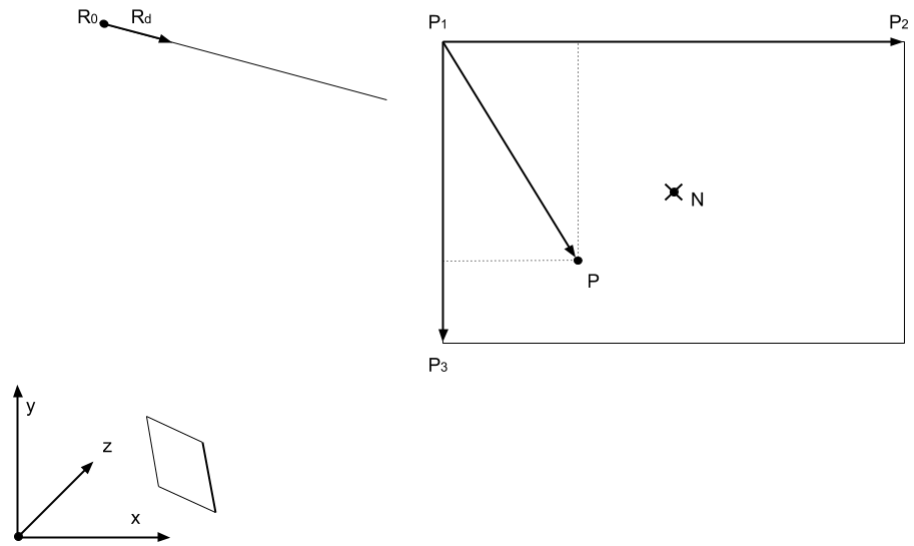
And the collision position for each t is:

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

3.8.2 Ray-Plane

(user3146587, 2014)

FIGURE 3.11: Ray-Plane intersection



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

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

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, and the t is valid only if:

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

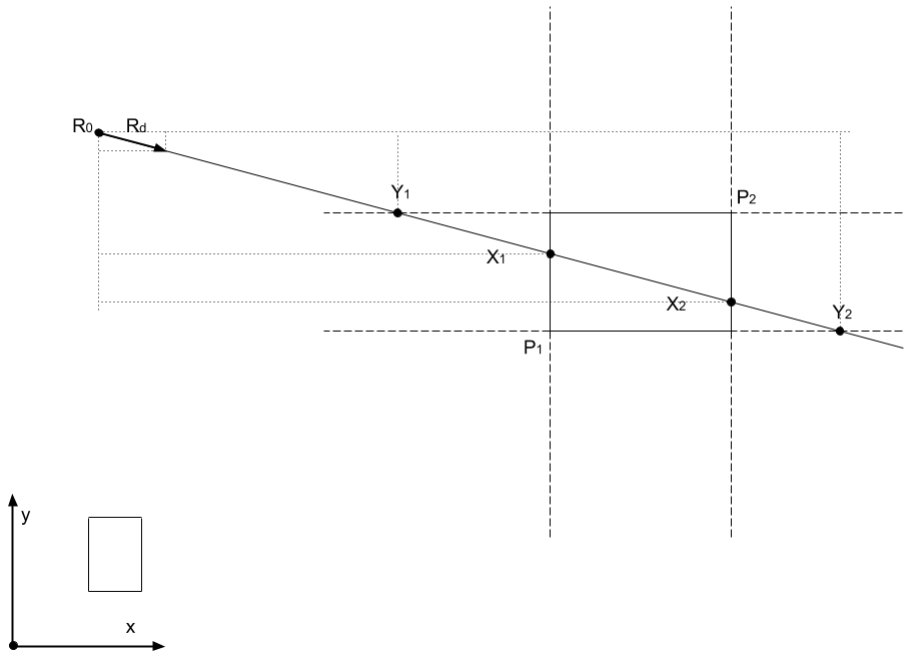
3.8.3 Ray-Box

(Williams et al., 2005)
(Barnes, 2011)
(Scratchapixel, 2014)

There is a octree implementation 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, then derive out Ray-Box-3D intersection.

Ray-Box-2D

FIGURE 3.12: 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

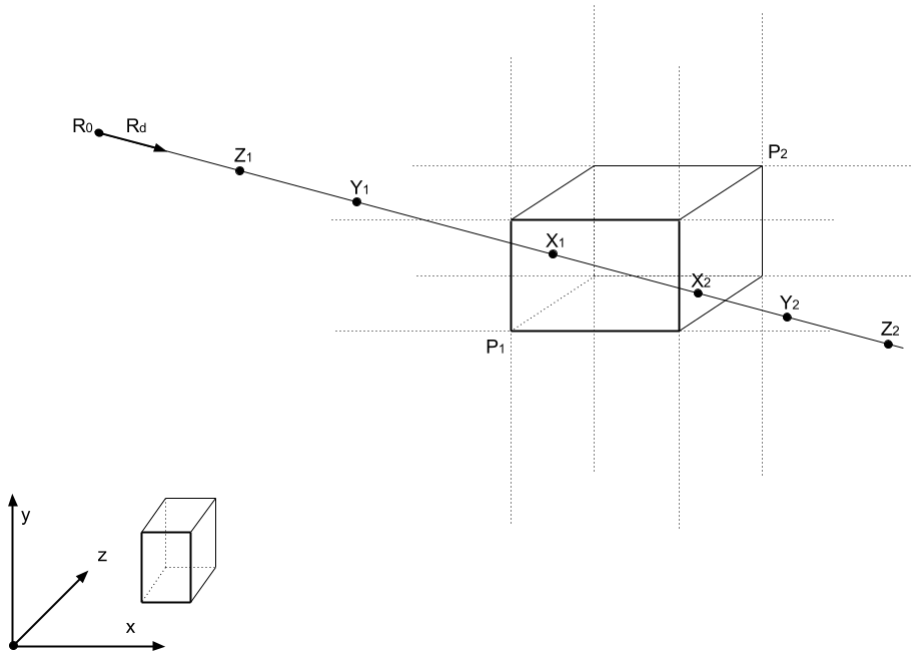
$$\begin{cases} t_{X_1} < t_{X_2} \\ t_{Y_1} < t_{Y_2} \end{cases}$$

$\therefore$ Which is

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

Ray-Box-3D

FIGURE 3.13: 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

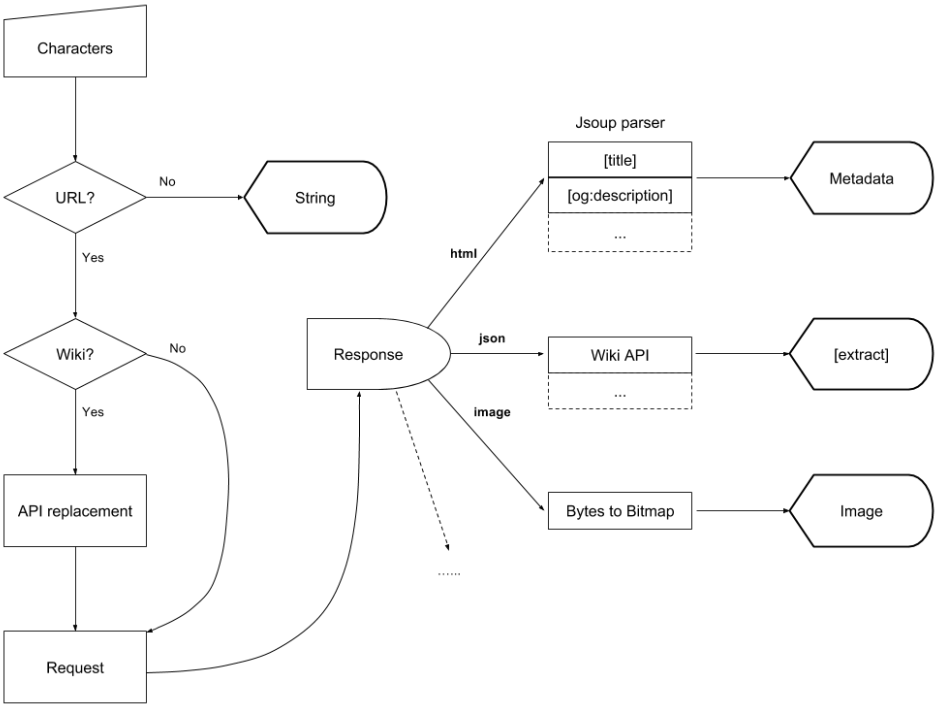
$$max(t_{X_1},\ t_{Y_1},\ t_{Z_1}) < min(t_{X_2},\ t_{Y_2},\ t_{Z_2}) \tag{3.7}$$

3.9 Description Analysis

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 3.14: 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.

3.10 File Server

****
Web server - golang - https://golang.org/pkg/net/http/
http.FileServer
https://github.com/ant0ine/go-json-rest
Go-Json-Rest is a thin layer ("Keep it simple, stupid") on top of net/http that helps building RESTful JSON APIs easily.
Volley - an HTTP library that makes networking for Android apps easier and most importantly, faster.
https://developer.android.com/training/volley/index.html

3.10.1 Assets

****
static, layer, kml, resource,model

3.10.2

****
Patch

3.10.3 Port Forwarding

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

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

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.

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

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

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.

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/.

Scratchapixel (2014). *Ray-Box Intersection*. URL: http://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection.

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.

Williams, Amy et al. (2005). "An efficient and robust ray-box intersection algorithm". In: *ACM SIGGRAPH 2005 Courses*. ACM, p. 9. URL: http://dl.acm.org/citation.cfm?id=1198748.