Exploitation de la réalité augmentée sur plateforme mobile Android dans le domaine touristique

Projet Optionnel à l'EPFL

Oskar Hallström

Abstract

L'objectif du projet est d'explorer un certain nombre de techniques associées à la réalité augmentée qui seront appliquées dans le futur au domaine touristique. Il s'agit principalement de superposer, en temps réel, des informations et des objets à un paysage capté par la caméra d'un smartphone. Le démonstrateur superposera des noms d'objets et des éléments graphiques qui indiquent l'emplacement exacte de chaque objet sur l'écran. Pour savoir où les objets sont localisés, cette application nécessite une base de données de lieux. La principale phase doit permettra de détecter des coordonnées GPS, l'orientation de la caméra et calculer, depuis la base de données, si l'objet apparaît dans le champ de vision. L'application sera développée sur une plateforme Android avec le langage Kotlin.

Veuillez noter que la page de couverture est écrite en français, alors que le reste du rapport est écrit en anglais.

•••

Please note that the cover page is written in French, while the rest of the report is written in English.

Table of contents

1 Introduction

1.1 Augmented reality in a nutshell

The main idea of augmented reality is to add digital elements to experiences of the real world [1]. Often these elements are graphical content, but they can also have other characteristics, such as for instance being in the form of audio effects. The added digital elements are intending to augment the user's perception of reality – hence the name augmented reality.

1.1.1 Augmented reality today

In the following section two examples of uses of augmented reality will be mentioned.

1.1.1.1 HADO

HADO is an augmented reality sport that has been played by millions. Equipped with head-mounted displays and armband sensors, the players can throw digital energy balls and protect themselves with shields. [2]

1.1.1.2 IKEA Place

IKEA Place is an application that allows the user to see how IKEA furniture fits their home. The user can try different colors, rotate and move the furniture around as well as see what different combinations of IKEA furniture looks like. [3]

2 Analysis

2.1 Used tools

In the following section the choices of the used tools are discussed.

2.1.1 Smartphone

In order to create an application that can give an augmented reality experience on the go, a platform which is portable, has a camera and is able to add graphical elements to the camera view is needed. The smartphone, with over 6 378 million users in the world in 2021 [4], fulfills all these requirements. There is no other currently existing device that better could provide this mobile augmented reality experience to a bigger number of users.

2.1.1.1 Android

The most common mobile operating system is Android, which held a market share of 72,84 % in June 2021 [5]. With a wide range of resources and tools available for developers, Google facilitates the development of Android applications. These two facts, combined with the fact that Android is the operating system running on the smartphone device owned by the writer of this report, is why Android is chosen for this project.

2.1.1.2 Sensors, system services and other necessary application input

In the following section aspects of the application that are dependent on certain data readings will be discussed. For this application these aspects correspond to the geographical position, the orientation of the device and lastly the device's camera input to the application. In the discussion of the two first aspects, an emphasis will be put on the sensors and system services needed for retrieval of this data.

2.1.1.2.1 Compass orientation

For the application to be able to display the right graphical contents on the screen, it must know the direction in which the camera points. There are several possible solutions for the retrieval of the compass orientation. However, if the compass should work even when not parallel to the ground, a combination of a magnetic field sensor and an accelerometer is needed. Together the readings from these two sensors can form a rotation matrix from which the compass orientation can be calculated.

2.1.1.2.2 Location

A central aspect of an application of this guiding nature is the location of the user. To serve the user with information about its surroundings, Android's location system service is used.

2.1.1.2.3 Camera input

For a graphical augmented reality smartphone application, camera input is essential. Consequently, the application needs to interact with the smartphone's camera provider.

2.1.2 Kotlin

Since 2019, Android mobile development has been Kotlin-first. Over 60% of the 100 apps on the Play Store use Kotlin and there is a big community accessible for the Kotlin developers. Lastly, Kotlin also supports iOS. [6]

Given Kotlin's dominance for Android development, it is reasonable to use it for the project. For the research aspect of this project, in which several different solutions will be explored, Kotlin's big community will be valuable. With the possible future extension of making this application available to iPhones, it is favorable that Kotlin already supports iOS.

2.2 Possible challenges and their implications for the project

A reoccurring point of discussion for this project is the level of abstraction to be used. On one side of the spectrum there are solutions such as Google's ARCore that provides APIs for all essential AR features [7], in contrast to the opposite side of the spectrum which could be considered being low-level programming. The latter solution, low-level programming, is clearly not feasible in the scope of this project, whereas the option of using ARCore would not be sufficiently challenging for a project of this type. Through reflection as well as discussion with the supervising professor, appropriate compromises could be found.

The independent nature of this project is also a source to challenges. Once a new problem arises, it usually takes considerably longer time to resolve it independently than it would have taken if other group members were available to give new perspectives and approaches.

3 Methodology

In the following section the research phase will be discussed briefly, while each sprint in the development phase will be discussed in detail. The following section aims to give an overview of the necessary stages to complete in order to get a functioning proof of concept for this application. The first part is a research phase, in which internet resources are used to form a general understanding of how the project should be carried out. The latter part consists of the steps of the actual development of the application. The latter part is based on the former one.

3.1 Research phase

The research phase was carried out through extensive reading of online material, such as articles, resources from Kotlin and Android/Google, source code as well as posts on Stack Overflow. The findings from this phase lay the foundation for the development phase and will be mentioned in corresponding development phase when necessary.

The research phase is estimated to correspond to 3 days of work. If considering all supervision meetings as part of the research phase, the work is estimated to correspond to 4 days of work.

The research phase is summarized by the following list:

1. General research about augmented reality
2. Research about existing tools for development of augmented reality applications
3. Research about existing tools for development of smartphone applications
4. Research about integration of application specific tools (sensors et cetera) including how to retrieve the needed data
5. Research about how to calculate when the camera points towards an object using retrieved data
6. Research about integration of graphical elements
7. Research about Kotlin and its practices

3.2 Development phase

The development phase is summarized by the following list of sprints:

1. Setup of development environment (including Android Studio, emulators et cetera) and development of a “Hello World” application
2. Addition of the function that calculates the compass angle
3. Addition of the camera view to the application
4. Implementation of the retrieval of the user’s current position
5. Implementation of a text display that writes out the objects displayed on the screen
6. Implementation of a graphical element that distinguishes one object from another if several objects are present on the screen

The execution of the above-mentioned sprints is estimated to have taken 18 days in total. In the following section the execution will be broken down into details.

3.2.1 Setup of development environment and development of a “Hello World” application

Android studio provides the user with a “Hello World” application. Consequently, the challenge is not to create the application itself, but to make the development environment work correctly so the app is runnable.

While setting up Android Studio some problems arose. Even though Android Studio could be downloaded and started without problem, it was impossible to run an application. After some research online it turned out that there was a problem with the latest version of Android Studio. The solution was to manually reset Android 12 (API 31) to Android 11 (API 30) by manually downloading API 30 and changing some contents in different gradle files.

It was also not possible to run an emulator on the computer used for development, despite several efforts to find a way of making the emulator work. This was probably due to the emulator options given by Android Studio being incompatible with Apple’s new M1 processor and operative system Big Sur. It was therefore necessary to download the application to the smartphone in order to run it.

Later in the project, an update of Android Studio was released. Once again new problems arose. After some research and different efforts, it turned out that some dependencies in the build script had to be changed in order for the new update to work. This also solved the previous emulator problem, which made the initial development of graphical elements easier since they could be tested on an emulator instead of using the smartphone each time.

Similar problems arose when a faulty refactoring of the name was made for the project package, which required some resets and updates. Once again, some hours of work were needed to make the app runnable again. In total this sprint corresponds to around 2 days of work.

3.2.2 Addition of the function that calculates the compass angle

For the implementation of the compass calculations, an extensive reading of the sensor specifications was made. Some sensors could be filtered away rather easy. For the remaining sensors several trigonometry and linear algebra calculations was tried in order to find a working solution.

After research and different tests three different ways of implementing the compass were found successful. However, the first implementation only worked in the setting in which the smartphone is parallel to the ground. For the application, the smartphone is held perpendicular to the ground, so this solution had to be discarded. The second and third solution worked even when the smartphone was held perpendicular to the ground and were therefore acceptable. During tests of the application, the third implementation appeared to give a more accurate result and it was consequently the solution that was decided to be used for the final proof of concept.

The two challenges for this sprint were to implement the right sensors and find out how they could be used to calculate the compass angle. To implement the sensors, studies on

different examples on the internet were carried out. Once the signals from the sensors were readable, the implementation of the calculations was studied in detail. During the process several recalculations were needed due to errors in calculations and the modelling. To find a working solution in horizontal mode (parallel to the ground) required reasonable efforts, whereas to find a working solution in vertical standing mode was a lot more challenging. The testing, calculations, implementation and corrections of the compass angle is approximated to correspond to 4 days of work.

3.2.2.1 Implementation 1 - using rotation vector sensor

The device's orientation is represented by a rotation vector through a combination of an angle and an axis. The used coordinate system consists of a x axis that points east, an y axis that points north and lastly a z axis which corresponds to a normal vector of the earth's surface. [8]

The rotation around the z axis, which corresponds to the compass angle, is calculated using Rodrigues' rotation formula.

Given the rotation axis **a**,

$$\boldsymbol{a} = (a_x, a_y, a_z)^T$$

the result of the rotation of the device, with an angle of θ radians, can be described by:

$$\boldsymbol{v}_{rot} = R\boldsymbol{v} \; where \; \boldsymbol{v} = (0,1,0)^T \; and$$

$$R = \begin{pmatrix} 1 + (\cos\theta - 1) * ({a_y}^2 + {a_z}^2) & (1 - cos\theta) * a_x * a_y - sin\theta * a_z & (1 - cos\theta) * a_x * a_z + sin\theta * a_y \\ (1 - cos\theta) * a_x * a_y + sin\theta * a_z & 1 + (\cos\theta - 1) * ({a_x}^2 + {a_z}^2) & (1 - cos\theta) * a_y * a_z - sin\theta * a_x \\ (1 - cos\theta) * a_x * a_z - sin\theta * a_y & (1 - cos\theta) * a_y * a_z + sin\theta * a_x & 1 + (\cos\theta - 1) * ({a_x}^2 + {a_y}^2) \end{pmatrix}$$

The compass needle corresponds to the vector $\boldsymbol{v}_{rot}$ projected onto the xy-plane. If no rotation has occurred, the compass needle points straight north, hence $\boldsymbol{v} = (0,1,0)^T$.

In order to find the compass angle α, the following calculations are used:

$$\begin{cases} \sin\left(\frac{\pi}{2} - \alpha\right) = \dfrac{\boldsymbol{v}_{rot} \bullet (0,1,0)^T}{|\boldsymbol{v}_{rot} \bullet (1,1,0)^T|} \\ \cos\left(\frac{\pi}{2} - \alpha\right) = \dfrac{\boldsymbol{v}_{rot} \bullet (1,0,0)^T}{|\boldsymbol{v}_{rot} \bullet (1,1,0)^T|} \\ \alpha \in [0, 2\pi[\end{cases} \Leftrightarrow \alpha = \begin{cases} \dfrac{\pi}{2} - \arctan\left(\dfrac{\boldsymbol{v}_{rot} \bullet (0,1,0)^T}{\boldsymbol{v}_{rot} \bullet (1,0,0)^T}\right) if \; \boldsymbol{v}_{rot} \bullet (1,0,0)^T > 0 \\ \dfrac{3\pi}{2} - \arctan\left(\dfrac{\boldsymbol{v}_{rot} \bullet (0,1,0)^T}{\boldsymbol{v}_{rot} \bullet (1,0,0)^T}\right) if \; \boldsymbol{v}_{rot} \bullet (1,0,0)^T < 0 \\ 0 \; if \; \boldsymbol{v}_{rot} \bullet (1,0,0)^T = 0 \; and \; \boldsymbol{v}_{rot} \bullet (0,1,0)^T > 0 \\ \pi \; if \; \boldsymbol{v}_{rot} \bullet (1,0,0)^T = 0 \; and \; \boldsymbol{v}_{rot} \bullet (0,1,0)^T < 0 \end{cases}$$

where

$$\boldsymbol{v}_{rot} = R\boldsymbol{v} = \begin{pmatrix} (1 - cos\theta) * a_x * a_y - sin\theta * a_z \\ 1 - (\cos\theta - 1) * ({a_x}^2 + {a_z}^2) \\ 1 - (\cos\theta - 1) * ({a_x}^2 + {a_z}^2) \end{pmatrix} \; and \; consequently$$

$$\begin{cases} \boldsymbol{v}_{rot} \bullet (1,0,0)^T = (1 - cos\theta) * a_x * a_y - sin\theta * a_z \\ \boldsymbol{v}_{rot} \bullet (0,1,0)^T = 1 + (\cos\theta - 1) * ({a_x}^2 + {a_z}^2) = \cos\theta - (\cos\theta - 1) * {a_y}^2 \end{cases}$$

However, the values retrieved by the rotation vector sensor are the following:

$$values[0] = a_x * \sin\left(\frac{\theta}{2}\right), values[1] = a_y * \sin\left(\frac{\theta}{2}\right), values[2] = a_z * \sin\left(\frac{\theta}{2}\right) \; and \; values[3] = \cos\left(\frac{\theta}{2}\right)$$ [8]

To calculate $\boldsymbol{v}_{rot} \bullet (1,0,0)^T \; and \; \boldsymbol{v}_{rot} \bullet (0,1,0)^T$ using only sensor data, the expressions that compose $\boldsymbol{v}_{rot} \bullet (1,0,0)^T \; and \; \boldsymbol{v}_{rot} \bullet (0,1,0)^T$ has to be rewritten.

Firstly, the compositions only containing components of the vector **a** is calculated:

$$\begin{cases} 1) \left|\sin\left(\frac{\theta}{2}\right)\right| = \sqrt{1 - \cos\left(\frac{\theta}{2}\right)^2} \; is \; used \; for \; 2) \; \& \; 3) \\ 2) \; a_x * sign(\sin\left(\frac{\theta}{2}\right)) = \frac{a_x * \sin\left(\frac{\theta}{2}\right)}{\left|\sin\left(\frac{\theta}{2}\right)\right|} \\ 3) \; a_y * sign(\sin\left(\frac{\theta}{2}\right)) = \frac{a_y * \sin\left(\frac{\theta}{2}\right)}{\left|\sin\left(\frac{\theta}{2}\right)\right|} \end{cases} \qquad 2) \; \& \; 3) \; gives \begin{cases} a_x * a_y = a_x * sign\left(\sin\left(\frac{\theta}{2}\right)\right) * a_y * sign\left(\sin\left(\frac{\theta}{2}\right)\right) \\ {a_y}^2 = (a_y * sign\left(\sin\left(\frac{\theta}{2}\right)\right))^2 \end{cases}$$

The remaining terms that must be rewritten so that they are expressed using sensor data are now $\cos\theta$ and $sin\theta * a_z$.

$$\cos\theta = 2 * \cos\left(\frac{\theta}{2}\right)^2 - 1, \qquad sin\theta * a_z = 2 * a_z * \sin\left(\frac{\theta}{2}\right) * \cos\left(\frac{\theta}{2}\right)$$

In conclusion, $\boldsymbol{v}_{rot} \bullet (1,0,0)^T \; and \; \boldsymbol{v}_{rot} \bullet (0,1,0)^T$ and consequently the compass angle can be calculated using sensor data in the following way:

$$\begin{cases} \boldsymbol{v}_{rot} \bullet (1,0,0)^T = \left(2 - 2 * \cos\left(\frac{\theta}{2}\right)^2\right) * \frac{a_x * \sin\left(\frac{\theta}{2}\right)}{\sqrt{1 - \cos\left(\frac{\theta}{2}\right)^2}} * \frac{a_y * \sin\left(\frac{\theta}{2}\right)}{\sqrt{1 - \cos\left(\frac{\theta}{2}\right)^2}} - 2 * a_z * \sin\left(\frac{\theta}{2}\right) * \cos\left(\frac{\theta}{2}\right) \\ \boldsymbol{v}_{rot} \bullet (0,1,0)^T = 2 * \cos\left(\frac{\theta}{2}\right)^2 - 1 - \left(2 * \cos\left(\frac{\theta}{2}\right)^2 - 2\right) * \left(\frac{a_y * \sin\left(\frac{\theta}{2}\right)}{\sqrt{1 - \cos\left(\frac{\theta}{2}\right)^2}}\right)^2 \end{cases}$$

3.2.2.2 Implementations using accelerometer and magnetic field sensor

Using readings from the accelerometer and the magnetic field sensor, the sensor manager can calculate the rotation matrix through the usage of the getRotation()-function. As in the previous case, the rotation matrix is an identity matrix when the device is parallel to the ground and pointing north. The used coordinate system also consists of a x axis that points east, an y axis that points north and lastly a z axis that corresponds to a normal vector of the earth's surface. [9]

3.2.2.2.1 Implementation 2

Using the already acquired rotation matrix, the sensor manager can compute the device's orientation in terms of Azimuth, Pitch and Roll through the usage of the getOrientation()-function. Azimuth corresponds to the rotation about the -z axis and is consequently the angle that corresponds to the compass angle. [10]

3.2.2.2.2 Implementation 3

The approach explained in the following part is inspired by a post on stack exchange. [11]

With this solution the rotation is partitioned into two different parts. First, a rotation R_1 about axis **a** in the xy-plane. The axis **a** is a normalized vector with the angle γ radians in the xy-plane.

$$\boldsymbol{a} = (\cos\gamma\,, \sin\gamma\,, 0)^T$$

The rotation about **a** is θ radians. By using Rodrigues' rotation formula, the following matrix is obtained:

$$R_1 = \begin{pmatrix} 1 + (\cos\theta - 1) * \sin\gamma^2 & (1 - cos\theta) * \cos\gamma * \sin\gamma & sin\theta * \sin\gamma \\ (1 - cos\theta) * \cos\gamma * \sin\gamma & 1 + (\cos\theta - 1) * \cos\gamma^2 & -sin\theta * \cos\gamma \\ -sin\theta * \sin\gamma & sin\theta * \cos\gamma & cos\theta \end{pmatrix}$$

The second rotation R_2 is a rotation about the -z axis that corresponds to the azimuth angle α. Once again Rodrigues' rotation formula is used to obtain the rotation matrix:

$$R_2 = \begin{pmatrix} \cos\alpha & sin\alpha & 0 \\ -sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The total rotation R is consequently corresponding to the following matrix:

$$R = R_2 * R_1$$

$$= \begin{pmatrix} \cos\alpha * (1 + (\cos\theta - 1) * \sin\gamma^2) + sin\alpha * (1 - cos\theta) * \cos\gamma * \sin\gamma & \cos\alpha * (1 - cos\theta) * \cos\gamma * \sin\gamma + sin\alpha * (1 + (\cos\theta - 1) * \cos\gamma^2) & \cos\alpha * sin\theta * \sin\gamma - sin\alpha * sin\theta * \cos\gamma \\ -\sin\alpha * (1 + (\cos\theta - 1) * \sin\gamma^2) + cos\alpha * (1 - cos\theta) * \cos\gamma * \sin\gamma & -\sin\alpha * (1 - cos\theta) * \cos\gamma * \sin\gamma + cos\alpha * (1 + (\cos\theta - 1) * \cos\gamma^2) & -\sin\alpha * sin\theta * \sin\gamma - cos\alpha * sin\theta * \cos\gamma \\ -sin\theta * \sin\gamma & sin\theta * \cos\gamma & cos\theta \end{pmatrix}$$

To calculate the compass angle α, a somewhat unintuitive solution will now be used. First, α is expressed using $sin\alpha$ and $cos\alpha$:

$$\alpha = \begin{cases} \arctan\left(\frac{sin\alpha}{cos\alpha}\right) + \pi \; if \; cos\alpha < 0 \\ \arctan\left(\frac{sin\alpha}{cos\alpha}\right) + 2\pi \; if \; sin\alpha < 0 \; and \; cos\alpha > 0 \\ \arctan\left(\frac{sin\alpha}{cos\alpha}\right) \; if \; sin\alpha > 0 \; and \; cos\alpha > 0 \\ \frac{\pi}{2} \; if \; sin\alpha > 0 \; and \; cos\alpha = 0 \\ \frac{3\pi}{2} \; if \; sin\alpha < 0 \; and \; cos\alpha = 0 \end{cases}$$

The fraction that needs to be calculated is consequently $\frac{sin\alpha}{cos\alpha}$. This is possible using the rotation matrix, if the following calculations is employed:

$$\frac{sin\alpha}{cos\alpha} = \frac{sin\alpha(1+\cos\theta)}{cos\alpha(1+cos\theta)} = \frac{sin\alpha(2+(\cos\theta-1))}{cos\alpha(2+(\cos\theta-1))}$$
$$= \frac{sin\alpha(1+(\cos\theta-1)cosy^2+1+(\cos\theta-1)siny^2)+\cos\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma-\cos\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma}{cos\alpha(1+(\cos\theta-1)cosy^2+1+(\cos\theta-1)siny^2)+sin\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma-sin\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma}$$
$$= \frac{\left(\cos\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma+sin\alpha*(1+(\cos\theta-1)*\cos\gamma^2)\right)-(-\sin\alpha*(1+(\cos\theta-1)*\sin\gamma^2)+cos\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma)}{(\cos\alpha*(1+(\cos\theta-1)*\sin\gamma^2)+sin\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma)+(-\sin\alpha*(1-cos\theta)*\cos\gamma*\sin\gamma+cos\alpha*(1+(\cos\theta-1)*\cos\gamma^2))}$$
$$= \frac{R(1)-R(3)}{R(0)+R(4)}$$

3.2.3 Addition of the camera view to the application

In order to add the camera view an basic understanding of the app layout is needed. Android has good resources available for how to add the necessary parts for the camera in the MainActivity of the program. The tricky part was how to properly add the camera preview to the layout and to change gradle files et cetera so the correct version of the camera preview could be run. This is approximated to have taken 1 day.

3.2.4 Implementation of the retrieval of the user's current position

To implement the location listener was, despite the small amount of code needed, a bit challenging. For the permission requests, the infrastructure already created for the permission requests of the camera could be used. The tricky part was how to create the location listener. The solution ended up being to create an object, which in Kotlin is an anonymous class, for the location listener.

After working perfectly for a week, there suddenly was a problem with the location retrieval. After the app stopped working correctly, some different tests made it clear that the onLocationChanged()-function never was called anymore. It took several hours of debugging to find the root of the problem and then some additional time to fix it.

The initial thought was that some Android update was the cause and that the code now had to be changed. Another thought was that some new addition of code had affected the location listener object. After verifying that these two cases were not the cause, it was hard to know how to continue. After several different efforts and research on the internet, GPS_PROVIDER was changed to NETWORK_PROVIDER for the location listener. The application started working, but not with enough precision. However, the root of the problem could now be concluded - it was the GPS_PROVIDER that was malfunctioning. To find out if it was a hardware problem, an app called "GPS Test" was downloaded to the smartphone device. Using that app, it could be verified that there was a hardware problem. The GPS signal was too weak, even though the smartphone device had worked previously at the same location. The solution was then to use the application's "Clear and update"-function. After using that function, the GPS_PROVIDER and consequently the AR application started to work again.

In total the work with the location detection corresponds to 2 days of work.

3.2.5 Implementations of a text display that writes out the objects displayed on the screen

This sprint can be subdivided into several sprints. As all locations initially are given in coordinates, it is important to note that a latitude degree always corresponds to the same distance, meanwhile the distance of a longitude degree varies depending on the latitude. When not using existing distance-functions from Kotlin, the calculations are based on the haversine formula. Both solutions iterate over the arraylist of objects, checking each object.

The work needed for these two implementations corresponds to 2 days of work. These days also includes creating a sample database by finding objects and their coordinates, as well as extensive testing and adjustments to different parameters.

3.2.5.1 Implementation 1

This implementation calculates the distance to each object and then uses that distance as a factor that multiplies the normalized compass angle vector. When adding the resulting vector to the current position, a new position is received. When viewing towards the object, this position is the same as the object's position. If the distance between the calculated position and the object's position is under a certain threshold, the object information is displayed on the screen.

This solution was concluded to be a little unintuitive - what distance threshold should be set and why?

3.2.5.2 Implementation 2

This implementation is more intuitive than the previous one. This solution compares the compass angle to the angle towards the object. If the difference of the two angles is under a certain threshold, the object information is displayed on the screen. In this implementation, the threshold to be set is clearer – it is the angle corresponding to the wideness with which the camera lens captures the surroundings. The angle towards the object is calculated in the following way:

$$dLatitude = (object_{latitude} - currentposition_{latitude}) * \frac{\pi}{180}$$

$$dLongitude = (object_{longitude} - currentposition_{longitude}) * \frac{\pi}{180}$$

$$oppositeLeg = arcsin\left(\left|\sin\frac{dLatitude}{2}\right|\right)$$

$$adjacentLeg = arcsin\left(\sqrt{cos\left(object_{latitude} * \frac{\pi}{180}\right) * cos\left(currentposition_{latitude} * \frac{\pi}{180}\right) * sin\left(\frac{dLongitude}{2}\right)^2}\right)$$

$$angleToObject = \begin{cases} \pi \; if \; if \; adjacentLeg = 0 \; and \; dLatitude > 0 \\ 3\frac{\pi}{2} \; if \; adjacentLeg = 0 \; and \; dLatitude < 0 \\ null \; if \; dLatitude = 0 \; and \; dLongitude = 0 \\ else \\ \arctan\left(\frac{oppositeLeg}{adjacentLeg}\right) + \pi \; if \; dLongitude < 0 \; and \; dLatitude < 0 \\ -\arctan\left(\frac{oppositeLeg}{adjacentLeg}\right) + 2\pi \; if \; dLongitude > 0 \; and \; dLatitude < 0 \\ -\arctan\left(\frac{oppositeLeg}{adjacentLeg}\right) + \pi \; if \; dLongitude < 0 \; and \; dLatitude \geq 0 \\ \arctan\left(\frac{oppositeLeg}{adjacentLeg}\right) \; if \; dLongitude > 0 \; and \; dLatitude \geq 0 \end{cases}$$

The calculation of opposite and adjacent leg is based on the haversine formula:

$$distance = 2 * earthRadius * arcsin\left(\sqrt{sin\left(\frac{dLatitude}{2}\right)^2 + cos\left(object_{latitude} * \frac{\pi}{180}\right) * cos\left(currentposition_{latitude} * \frac{\pi}{180}\right) * sin\left(\frac{dLongitude}{2}\right)^2}\right)$$

The angle threshold is set through approximations made using the camera view. The smartphone was held parallel to a wall and then the distance to the wall as well as the horizontal length of the viewed part of the wall was calculated. Using trigonometry, the angle of the field of view could be calculated. This approximation was then also used in construction the frustum matrix.

5.2.6 Implementation of a graphical element that distinguishes one object from another if several objects are present on the screen

The final sprint of the project was also a very time-consuming one. This sprint has been subdivided into several sprints. It is approximated to have corresponded to 7 days of work.

5.2.6.1 Integration of transparent OpenGL-layer

To integrate a transparent layer of OpenGL to the application took a substantial amount of time. Efforts were made to create a new layout, however the final solution ended up simply adding an OpenGL layer to the already existing layout. After trying several different approaches inspired by existing implementations online, a solution compatible with the current project and its environment was found. The remaining task was then to make the layer transparent, which only required the addition of one single row of code.

5.2.6.2 Integration of the graphical elements

The next challenge was to set up matrices so that the graphics could be displayed according to the locations of objects. The first step was to decide what type of coordinate system that would be used for the objects. After that different type of matrices were created and implemented to get the intended final result.

5.2.6.2.1 Creation of coordinate system

Using the previously mentioned haversine formula, each position was transferred into a coordinate system using the metric system. The coordinate pair where x=z=0 correspond to the null Island, where latitude=longitude=0. In this coordinate system, the x axis points east, the z axis points south, and the y axis corresponds to a normal vector to the earth's surface. This was done through the following calculations:

$$\begin{cases} z = -latitude * \frac{\pi}{180} * earthRadius \\ distanceToNullIsland = 2 * earthRadius * arcsinus\left(\sqrt{sin\left(\frac{latitude * \frac{\pi}{180}}{2}\right)^2 + cos\left(latitude * \frac{\pi}{180}\right) * sin\left(\frac{longitude * \frac{\pi}{180}}{2}\right)^2}\right) \\ x = sign(longitude) * \sqrt{distanceToNullIsland^2 - z^2} \end{cases}$$

5.2.6.2.2 Implementation of matrices

In the vertex shader, the local position of each vertex that constitutes an object is multiplied by the MVP matrix so that it is rendered to the right final position. The MVP matrix is built using several matrices:

$$MVP\ matrix = frustum\ matrix * view\ matrix * model\ matrix$$

This section will go through the creation of each of the matrices.

5.2.6.2.2.1 Model matrix

The model matrix transforms the object and its vertices' positions so that it has the right size, orientation and position in the global coordinate system. The model matrix is thus also built using several matrices:

$$model\ matrix = translation\ matrix * rotation\ matrix * scale\ matrix$$

The scale matrix is an identity matrix that is multiplied by a factor corresponding to the distance between the user and the object. If the object is further away from the user it needs to be bigger to be seen, whereas it needs to be smaller when the object is closer in order to not cover the whole screen. The distance is calculated using the Pythagorean theorem.

The rotation matrix rotates the object so that it is always perpendicular to the user's viewing direction. The rotation is about the y axis and is corresponding to the angle to the user from the object. This is due to the object being perpendicular to the user when they are at the same z coordinate by default. Since each position now is given in the newly created coordinate system, the haversine formula is not needed. The angle is calculated using the calculations on the following page.

$$dX = userX - objectX$$

$$dZ = userZ - objectZ$$

$$angleToObject = \begin{cases} \frac{\pi}{2}\ if\ dX = 0\ and - dZ > 0 \\ 3\frac{\pi}{2}\ if\ dX = 0\ and - dZ < 0 \\ null\ if\ dX = 0\ and\ dZ = 0 \\ \arctan\left(\frac{-dZ}{dX}\right) + \pi\ if\ dX < 0 \\ \arctan\left(\frac{-dZ}{dX}\right) + 2\pi\ if\ dX > 0\ and - dZ < 0 \\ \arctan\left(\frac{-dZ}{dX}\right) if\ dX > 0\ and - dZ > 0. \end{cases}$$

The translation matrix is created using a built in OpenGL-function and translates the object from the origin to the correct x and z coordinate.

Even though all matrix calculations for the model matrix were theoretically correct from scratch, there was a misinterpretation in the implementation of the initial placement of each object and a potential faulty implementation of a matrix multiplication that caused a major issue. This made the addition of the scaling matrix and the rotation matrix making the object invisible. A big effort was put into trying to find errors and trying different implementations of these two matrices, changing their order of multiplication et cetera. It took a substantial amount of time until the root cause was found.

5.2.6.2.2.2 View matrix

The view matrix is what makes the program able to display the right graphics in terms of the position of the user and the direction in which it looks. The view matrix is created using a built in OpenGL-function. The input parameters for this function are the current position, the position that the user looks towards and an up vector that corresponds to the normal vector to the earth's surface.

Consequently, the implementation of this matrix seems straight forward. However, a small misinterpretation in the view input caused an error that took a long time to find. Instead of the position at which the user is looking towards, the initial view input was thought to be the vector corresponding to the viewing direction. When verifying the view matrix function, the user position was set to the origin, which made the viewing direction vector correspond to the viewed position and in consequence the right object was visible. This made the view matrix seem to be correct, which caused big efforts being put into finding a problem that was thought to be elsewhere. It therefore took a long time to understand that the initial implementation of the view matrix only was correct at the origin.

5.2.6.2.2.3 Frustum matrix

The frustum matrix is responsible for precising the square frustum, in the “world of objects” that the camera points towards, that contains the objects that should be rendered into the smartphone display. This matrix is set through a built in OpenGL-function. The input parameters for this function are left, right, top and bottom which set the sides of the square in the near plane, as well as the parameters near and far which set the distance to the near and the far plane respectively. The ratio between left/right and top/bottom is set according to the screen size, whereas the ratio between left/right and near is set according to the angle of the field of view. The angle of the field of view is the same angle as the one calculated in section 3.2.5.2 Implementation 2.

The near parameter is set to 1 and the far parameter is set to 100. The latter must be increased if the application is to be used for settings in which the objects can be visible even though they are located further away than 100 meters.

4 Conclusion

4.1 Final proof of concept

The final proof of concept is an application that displays the name and distance to visible objects, as well as a vertical line indicating its position on the screen. The application works when the phone is held in a vertical position, being perpendicular to the ground. A major issue of the application is the inaccuracy of the GPS coordinates. This becomes apparent when the application is used in environments in which the objects are small and the distances short. Nevertheless, the graphical elements still help distinguishing objects from each other since they despite not always marking the exact right place are offset in the same direction. If there are two monuments present on the screen, the graphical element for the leftmost monument will always be to the left of the graphical element for the other monument. Another factor that augments the functionality of the application is that the distance to each visible object is displayed, which also becomes helpful when distinguishing visible objects from each other.

4.2 Possible extensions

If more time was available, there are some useful extensions that could have been made. The first one would be to add the altitude aspect to the coordinate system. With the current implementation, the objects and user are always considered to be at the same altitude. In cases where the altitude differs among the objects and the user, the objects that are considered visible by the current application might not be visible due to them being located at a vastly different altitude. In some other cases, object might also be placed on top of each other, which would make them hard to distinguish them from each other.

Another possible extension, which partly is connected to the previous proposition, would be to make the detection and displaying of graphical elements working seamlessly in settings in which the smartphone is not held perpendicular to the ground.

Another simple extension would be to add the possibility of receiving additional information about the visible objects. In contrast, a more complex extension could be to use computer vision to detect objects more accurately. This could be an effective way to avoid being affected by the inaccuracy of the GPS coordinates.

4.3 Personal perspectives

The writing of this report, together with simultaneous updates to the applications, is approximated to have taken 4 days. In total, this project is approximated to correspond to 28 days of work. In addition to the 18 days mentioned in the development phase section, 2 days has been added for general changes as well as implementations that was not used and not considered useful to mention in the report. A substantial amount of these 28 days has consisted of debugging and troubleshooting, which could have been considerably decreased if it was not for the independent nature of this project.

Developing an android application for the first time was challenging. Many of the most time-consuming challenges was not associated to the calculations and the writing of code, but instead to for instance Android Studio, build scripts, gradle files and hardware. Also, much time was required since it was the first time many tasks were made, for example to get data from sensors, add the camera view, add a transparent graphical layer and so on. Even though Google has a great community with a lot of resources available, it was often hard to take advantage of this community due to many of the existing examples being outdated, written in the wrong language, using depreciated APIs, not being appropriate for my specific application and so on.

The project has given a lot in terms of learning about android development – for instance good knowledge about Android Studio and its useful tools, proficiency in Kotlin and expertise in how to implement sensors and the camera. These experiences will be very useful in future development of Android applications, which will be possible to do much more efficiently. It was also fun and educative to implement previous knowledge from a course in Computer Graphics in another context, in which the development had to be made independently.

Personally, I find the future of augmented reality exciting. To be able to recreate a small fraction of what is possible with this phenomenon really sparked my interest. Even though I am unsure about the amount of Android applications I will develop in the future, the project and the interesting discussions with my supervising professor has given me a lot of inspiration and ideas that surely will be useful in the future. The project has diversified my skill set and it has made me a better computer scientist.

5 References

[1] “Augmented Reality”, Google
(https://arvr.google.com/ar/)

[2] “What is HADO”, HADO
(https://hado-official.com/en/about/)

[3] “IKEA Place” - Apps, IKEA
(https://www.ikea.com/se/sv/customer-service/mobile-apps/)

[4] “Smartphone users worldwide 2016-2021”, statista (2021)
(https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/)

[5] “Market share of mobile operating systems worldwide 2012-2021”, statista (2021)
(https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/)

[6] “Kotlin for Android”, Kotlin (2021)
(https://kotlinlang.org/docs/android-overview.html)

[7] “Choose your development environment”, Google Developers (2021)
(https://developers.google.com/ar/develop)

[8] “Values” - SensorEvent, Google Developers (2021)
(https://developer.android.com/reference/android/hardware/SensorEvent#values)

[9] “getRotationMatrix” - SensorManager, Google Developers (2021)
(https://developer.android.com/reference/android/hardware/SensorManager#getRotationMatrix(float[],%20float[],%20float[],%20float[]))

[10] “getOrientation” - SensorManager, Google Developers (2021)
(https://developer.android.com/reference/android/hardware/SensorManager#getOrientation(float[],%20float[]))

[11] “What’s the best 3D angular co-ordinate system for working with smartphone apps”, by User Stochastically on StackExchange (2017)
(https://math.stackexchange.com/questions/381649/whats-the-best-3d-angular-co-ordinate-system-for-working-with-smartphone-apps/382048#382048)

The links to all sources were verified to work on October 3rd, 2021.