Masterproef
Exploring different visualization techniques for Augmented Reality navigation helpers inspired by Open World Games

Promotor:
Prof. dr. Johannes SCHOENING

Cedric Lodts
Scriptie ingediend tot het behalen van de graad van master in de informatica
Masterproef
Exploring different visualization techniques for Augmented Reality navigation helpers inspired by Open World Games

Promotor:
Prof. dr. Johannes SCHOENING

Cedric Lodts
Scriptie ingediend tot het behalen van de graad van master in de informatica
ABSTRACT

Optical head mounted displays (OHMD) are a special kind of head mounted display (HMD) that are a big topic within wearable computing. They have been heavily researched within the past three decades and are finally starting to become available for regular consumers. An OHMD consists of a see-through display that is worn near the eyes that allows us to augment the view of users with extra information. Besides its output possibilities, an OHMD can have added features like a camera, microphone and touch control for user input. A prominent example is the Google Glass.

An OHMD can help users with orientation and navigation. While a smartphone can be used for displaying a map or other helper for navigation, it constricts users to keeping their phone in their hand. Furthermore optical head mounted displays enable users to keep their hands free. There has already been done some research for using Augmented Reality (AR) to help with navigation but the effect of different visualization techniques has not been explored yet. Augmented Reality adds virtual components to the real world by rendering computer generated objects directly on the user’s vision.

A genre of computer games called “Open World Games” has similar navigation problems as the real world. Open World Games (OWG) are games where the player can freely roam a world and complete objectives at the player’s leisure in stead of being forced to follow a linear story or environment that the game presents to the player. We looked at a corpus of 100 OWG from the past 25 years and analysed AR-helpers and Widget-helpers in these games. This thesis explores whether these visualizations from OWG can be used in real-world situations.

In our user study we compared these different kinds of AR-helpers and Widget-helpers for navigation, orientation, annotation and exploration. We found out that in general these helpers are well perceived. For navigation-tasks, the AR-helper ”Static Arrows” (which are some 3D-arrows floating above the ground that point in the direction users need to go) was chosen as the most helpful by users and the most helpful Widget-helper was a mini-map with the path marked on it.

This thesis aims to provide new and useful ways for displaying visualizations for navigation helpers through Augmented Reality with the help of an optical head mounted display.
DUTCH SUMMARY

Open World Games

Open World Games zijn games waarbij de spelers de vrijheid hebben om zelf te kiezen wanneer ze aan bepaalde missies beginnen en waar en wanneer ze ergens naar toe reizen. Dit genre van games is de afgelopen jaren zeer populair geworden, met namen zoals GTA 5 en The Witcher 3: Wild Hunt.

Navigatie in een Open Wereld


Studie van Games en Reeds Bestaand Onderzoek

Deze visualisaties zijn ook vergeleken met degene die gebruikt worden in onderzoeken voor echte navigatie. Hierdoor zagen we dat deze games Widgets (zaken die in beeld blijven zoals bv. een mini-map in de hoek van het scherm) en Augmented Reality (3D-objecten die in de wereld naast het personage zweven) gebruiken om bv. de weg te tonen. Deze konden dan elk ook nog eens opnieuw opgedeeld worden in Navigatie (de weg vinden van A naar B), Oriëntatie (zien wat zich om je heen bevindt), Annotatie (naamplaatjes kleven) en Ontdekking (besef krijgen wat er in de buurt allemaal te beleven valt).

1 http://www.igta5.com/map
2 http://www.giantbomb.com/the-witcher-3-wild-hunt/3030-41484/
Navigatie: Een speler in The Elder Scrolls: Skyrim die “Clairvoyance” gebruikt om de weg te vinden.

Oriëntatie: Met behulp van Eagle Vision kan een speler in Assassin’s Creed zien waar de guards zich rond hem/haar bevinden.

Annotatie: Het spel Watch Dogs labelt alle apparaten en personen waar de speler mee kan interaggeren.

Ontdekking: Tom Clancy’s: The Division helpt spelers met zien wat er zich allemaal in de buurt bevindt door dit op de “Mega Map” te tonen.

Navigatie in de Echte Wereld

Nu we weten hoe deze games spelers helpen met navigatie, hebben we iets nodig om deze visualisaties in de echte wereld te krijgen. Hierbij kunnen we gebruik maken van een “Optical Head Mounted Display” zoals Google Glass. Dit is een soort van bril waarbij de brilglazen ook als beeldscherm kunnen functioneren.

Google Glass

Terwijl je door de brilglazen kijkt, kan de Optical Head Mounted Display extra informatie (objecten of tekst) op de glazen tekenen zodat het lijkt alsof die objecten zich in de echte wereld bevinden. Het idee van deze thesis is dan ook dit soort van bril te gebruiken om mensen te helpen met navigatie. De bedoeling is dat ze hun handen vrij kunnen houden en rechtstreeks in hun gezichtsveld met navigatie geholpen kunnen worden.
Prototype met Oculus Rift

Met behulp van de Unity game engine, is na de studie een prototype gemaakt om met een Oculus Rift wat van deze visualisaties door testgebruikers te laten uitproberen. De testpersonen kregen 18 verschillende navigatie-helpers voorgeschoteld waarmee ze voor elke helper vier taken moesten oplossen.

Oculus Rift

Uit deze testen hebben we dan kunnen concluderen dat over het algemeen Augmented Reality (AR) beter schijnt te zijn dan Widgets omdat gebruikers veel sneller én met minder fouten de taken tijdens de gebruikerstest konden oplossen met de AR helpers.

Met behulp van AR konden testgebruikers o.a. de weg gewezen worden door een pad van mist. (AR-Navigatie)

De testgebruikers moesten o.a. een minimap gebruiken om de weg te kunnen vinden. (Widget-Navigatie)

Daarnaast moesten de testgebruikers een enquête invullen waar ze op moesten aanduiden welke helper ze het best en het mooist vonden. De resultaten hiervan toonden ook aan dat AR in alle opzichten superieur is aan Widgets.

Opmerkelijk is wel dat wanneer we de gebruikers als laatste taak zelf een interface laten samenstellen van verschillende visualisaties om hen te helpen navigeren, ze toch eerst Widgets zullen kiezen boven AR. Zo werd bijvoorbeeld mini-map veel vaker gekozen dan eender welke AR-helper.

De volgende stap is een applicatie uitwerken die met een Optical Head Mounted Display kan werken zoals de Google Glass. Dit in een echte situatie (echte wereld) waarin gebruikers zelf kunnen kiezen hoe ze geholpen worden met welke helpers.
ACKNOWLEDGEMENTS

I would first like to thank my promotor Prof. dr. Johannes Schöning for his great guidance, all his suggestions and for giving me the opportunity to explore such an interesting subject.

Next I would like to thank dr. Dirk Wenig for proofreading my thesis and providing suggestions, as well as everyone else at EDM who gave me advice during the development of the prototypes.

And lastly I want to thank my friends and family for their support and volunteering for my user tests.
# CONTENTS

1. Introduction and Motivation ................................................. 1
   1.1 Contribution .............................................................. 3

2. Related Work .............................................................. 5
   2.1 Brief Overview on Video See-Through Displays ................. 5
   2.2 Current Head-Mounted Displays and Cameras .................. 8
      2.2.1 Augmented Reality and Widgets ................................. 8
      2.2.2 Properties of Displays ........................................... 10
      2.2.3 Properties of Cameras ........................................... 10
      2.2.4 Hardware Overview .............................................. 11
   2.3 Research Domains ...................................................... 13
      2.3.1 Medical ............................................................. 13
      2.3.2 Cultural and Learning ........................................... 14
      2.3.3 Community ....................................................... 14
      2.3.4 Smart Homes and Ubi-comp Environments .................. 15
      2.3.5 Sports ............................................................ 16

3. AR Navigation-Helpers in Open World Games + Research ........... 17
   3.1 Related Research in the Area of AR-Navigation ............... 17
   3.2 Visual Navigation-Helpers in Open World Games ............. 27
      3.2.1 Navigation ....................................................... 27
      3.2.2 Exploration .................................................... 28
      3.2.3 Orientation ..................................................... 28
      3.2.4 Annotation + Labels ............................................ 29

4. Study Comparing Visual Helpers in OWG and in Research .......... 31
   4.1 Visual Helpers in Research .......................................... 31
   4.2 Study of 100 Open World Games ................................... 34
   4.3 Interpretation of the Data ......................................... 40
   4.4 Real World Navigation .............................................. 44

5. Concept ................................................................. 49
   5.1 Augmented Reality ................................................... 49
      5.1.1 AR Navigation .................................................. 50
      5.1.2 AR Orientation ................................................ 52
      5.1.3 AR Annotation .................................................. 52
      5.1.4 AR Exploration ................................................ 53
   5.2 Widgets ............................................................... 53
5.2.1 Widget Navigation ........................................ 53
5.2.2 Widget Orientation .................................... 54
5.2.3 Widget Annotation ....................................... 55
5.2.4 Widget Exploration ..................................... 55

6. Implementation ............................................. 57
6.1 Components ............................................... 57
6.1.1 Unity .................................................. 58
6.1.2 Oculus Rift SDK ....................................... 58
6.1.3 Google Streetview API ................................ 58
6.1.4 Microsoft Image Composite Editor ................. 58
6.1.5 Google Sketchup ....................................... 58
6.1.6 Cardboard SDK for Android ......................... 58
6.2 Implementation of The Google Cardboard Prototype . 59
6.2.1 MainActivity .......................................... 59
6.2.2 Summary ................................................ 63
6.3 Downloading Photos from Google Streetview ............ 64
6.4 Unity Scene for the Oculus Rift Prototype .............. 65
6.5 Implementation of AR-Helpers ......................... 65
6.5.1 AR Navigation ......................................... 66
6.5.2 AR Orientation ......................................... 69
6.5.3 AR Annotation .......................................... 70
6.5.4 AR Exploration ........................................ 71
6.6 Implementation of Widget-Helpers ...................... 72
6.6.1 Widget Navigation ..................................... 72
6.6.2 Widget Orientation .................................... 74
6.6.3 Widget Annotation ..................................... 76
6.6.4 Widget Exploration .................................... 77
6.7 Hierarchy of the Different Components ................. 77
6.7.1 Streetview ............................................. 78
6.7.2 VisualizationController .............................. 78
6.7.3 OVRPlayerController ................................. 81
6.7.4 The Visualization Categories ....................... 82

7. Evaluation .................................................. 83
7.1 Tasks ....................................................... 83
7.1.1 Structure of the Tasks ................................ 83
7.1.2 Navigation Tasks ..................................... 84
7.1.3 Orientation Tasks ..................................... 86
7.1.4 Annotation Tasks ..................................... 87
7.1.5 Exploration Tasks .................................... 89
7.2 Results ..................................................... 90
7.2.1 Task Performance Results ......................... 90
7.2.2 Survey Results ........................................ 93
7.3 Discussion .................................................. 98

8. Conclusion ................................................... 99
## Contents

9. Future Work ................................................................. 101  
  9.1 Creating Seamless Transitions for the Oculus Rift Prototype ... 101  
  9.2 Real-Time Application for an Optical Head Mounted Display ... 102  

Appendices ........................................................................... 103  

A. Study of 100 Open World Games ..................................... 105  
  A.1 Discussed games .......................................................... 105  
  A.2 List Of Different Helpers And Games That Use Them ....... 206  

B. Code Snippets ...................................................................... 231  
  B.1 First Prototype ............................................................ 231  
  B.2 Second Prototype ........................................................ 262  

C. User Test Survey ............................................................ 275  

D. Meetings ........................................................................... 291
1. INTRODUCTION AND MOTIVATION

There has been massive research for the past couple of decades into optical see-through displays and now we have the first mass market products. An optical head mounted display (OHMD) is a special kind of head mounted display containing a display that can be seen through, is worn in front of the eyes and can have added features like a camera, microphone and touch control for user input.

The Google Glass (see Figure 1.1) is a prominent example of an OHMD. Next to its see-through display it contains a touch-pad that allows users to control the device by swiping motions. In addition, it has a camera to take photos and record videos. Lastly, Google Glass has smartphone-like applications that can also be controlled with voice commands.

This thesis contains a feasibility study in which I explore the use of these head mounted displays for different kinds of Augmented Reality (AR) and Widget visualization techniques for navigation-helpers inspired by Open World Games (OWG). AR-helpers add 3D-objects layered on top of the real world which may or may not be in a user’s vision depending on whether or not users are looking in the right direction (see Figure 1.3 for an example) while Widget-helpers will be in a user’s vision at all times (e.g. a mini-map that stays in vision like in Figure 1.2).

Fig. 1.1: Google Glass

Fig. 1.2: Example of a Widget-helper (mini-map in the corner of a user’s vision).

Fig. 1.3: Example of an AR-helper (arrows floating on the road that lead users to their destination).
Using an optical head mounted display for navigation also enables users to keep both their hands free most of the time. In contrast, using a smartphone as a navigation device for example, requires users to keep the phone in their hand while navigating.

This master thesis was motivated by a different kind of map visualization seen in the video game “Tom Clancy’s The Division” (see Figure 1.4), where a player can see his/her location by standing on top of a big “mini”-map of the direct surrounding (the game calls this a “mega map”). The user’s position on the mega map and the physical world is the same. This thesis will explore if and how visualizations for navigation helpers like the ones used in OWG like this one could work in the real world with the help of an OHMD.

![Fig. 1.4: Screenshot from the game: Tom Clancy’s The Division](image)

An OWG is a game that grants players the freedom to move around in a whole virtual world. In addition, it allows players to choose how they will approach the objectives or missions the game offers. Other games will require players to complete it in a certain pattern (e.g., missions that need to be completed in a certain sequence). Two famous examples of Open World Games are GTA V and The Witcher 3 (see Figure 1.5 and Figure 1.6 respectively).

Open World Games face similar problems for navigation as the real world because players need a way to travel around this virtual world without getting lost as well. Many different techniques for helping the player navigate have been used in games for assisting with: navigation, orientation, annotation (+ labels) and exploration. As these kind of techniques are used in famous games, we assume they have been proven useful. So the question is if they can be useful for real-world situations too.
In my master thesis I investigated the use of head worn devices for visualising different Augmented Reality navigation helpers inspired by OWG.

A comparable study was performed by Moura et al. [26]. Instead of trying to find navigation visualizations in (Open World) games, they decided to create a game of their own. In this game they implemented different “navigation cues” (see Figure 1.7 for an example) and had them tested by users to see which cues will lead their users the most effectively to their destination.

Based on the response of the users, they adapted their game. One has to keep in mind however, that they wanted to use their navigation cues for games where there still has to be some form of challenge for finding one’s objective. Whereas the visualizations used in this thesis aim to help users with finding their objective as quickly as possible.

1.1 Contribution

This thesis contributes the first full study of AR-helpers and Widget-helpers used in OWG compared to similar helpers used in research, in a systematic way. Chapter 4 contains a study of 100 different OWG. In this study, the helpers used in these OWG were divided into four different categories (Navigation, Orientation, Annotation and Exploration) and their usage over time was studied as well. After this, a comparable study of 25 research-papers related in the area of AR-navigation that used similar helpers was completed and compared to the OWG.

Before the studies, a first prototype was created with Google Cardboard SDK for OpenGL ES that allowed users to look around in a virtual Streetview scene and annotate buildings with a simple virtual floor-plan based on Google Maps coordinates (see Figure 1.8) and manually assigned georeferences (see Figure 1.9) on the 360° im-

1 http://www.igta5.com/map
2 http://www.giantbomb.com/the-witcher-3-wild-hunt/3030-41484/
1. Introduction and Motivation

Fig. 1.8: A screenshot of the first prototype.

Fig. 1.9: Georeferences are assigned to images so the virtual floor-plan can be mapped with the real building.

After the second study, a prototype was created for the Oculus Rift using the Oculus Rift SDK for Unity. At the time of developing the first prototype, there was no Cardboard SDK for Unity yet. The new prototype used the same kind of Google Streetview images as scenes to look around in. In addition, 18 different helpers found by the earlier two studies were simulated in different Streetview Scenes. Figure 1.10 and Figure 1.11 are two examples of helpers created with the Oculus Rift SDK for Unity.

Fig. 1.10: The Fog Path helper used in the second prototype.

Fig. 1.11: The Heat Vision helper used in the second prototype.

The second prototype was then tested by 16 different users to find out which helpers are the most helpful, useful and visually appealing. Users were given a set of tasks to complete using 18 different kinds of helpers and they were handed a survey to fill in after each set of tasks. The results of the user tests can be found in Chapter 7.
2. RELATED WORK

Optical head mounted displays have been an important research subject within wearable computing for a few decades now. They have evolved from devices requiring a lot of hardware to small mobile computers that can be worn as glasses.

2.1 Brief Overview on Video See-Through Displays

Head mounted displays consist of a display in front of the eyes and they can have added features like a camera, microphone or touch control for user input. Head mounted displays with a video see-through display have a camera that records the real world and projects it on the video see-through display in front of users’ eyes. This way, additional information can be layered on top of the real world by rendering objects on the video see-through display on top of what the camera sees.

Fig. 2.1: Sutherland’s head-mounted display [40].

Fig. 2.2: Mann wearing a video see-through display to control photographic equipment [20].

Already in the 60’s the idea of a head-mounted display emerged. The first of these head-mounted displays was created by Sutherland in 1968 and named “the sword of Damocles” [40]. The system received that name because of the large and heavy construction that was floating above his head. Sutherland’s system allowed users to look around in a “virtual environment”, which at that time was nothing more than some simple wireframe rooms. A head mounted display always has a screen that can be worn in front of the eyes and can have many forms like a pair of glasses or a helmet for example.
In 1981, Mann managed to build the first “wearable” multimedia computer inside a big bag he had to carry around [20]. The screen of his computer was built into a so-called video see-through display (see Figure 2.2). It could display text and graphics on a CRT-screen. In addition, the parts of the system that could be controlled with seven micro-switches built into the handle of a flash-lamp were cameras, flash-bulbs and other photographic equipment.

![Fig. 2.3: A pilot using the IHADSS](image)

![Fig. 2.4: KARMA explaining the maintenance for a laser-printer](image)

The U.S. Army saw a military use in these video see-through mounted displays. In 1985 they equipped helicopter pilots with the so-called Integrated Helmet and Display Sighting System (IHADSS) which was a helmet-mounted display that was designed to provide a visually coupled interface between the helicopter and the pilot (see Figure 2.3). A thermographic camera sensor that was mounted on the nose of the helicopter could be controlled by the pilot’s head movements and its information was shown on the pilot’s helmet-mounted display which had a 40° by 30° FOV (field of view) and aided pilots with night navigation.

MacIntyre and Seligmann developed a system in 1993 called KARMA (Knowledge-based Augmented Reality for Maintenance Assistance). The system consisted of an optical head mounted display (see Figure 2.4) with an optical see-through display in stead of a video see-through display. This meant that only computer-generated data needed to be shown on the screen. KARMA showed 3D graphics floating next to and on top of a printer. For instance, the Augmented Reality-application explained how the printer had to be operated.

Finally, head mounted displays have reached a stage where they can be worn as a mobile device without a giant bag of hardware that users need to carry around. Google Glass (see Figure 2.5) is one of the big names of optical head mounted displays called SmartGlasses because it has smartphone-like functionalities built into a pair of glasses.
SmartGlasses aim to be part of the everyday life and will attempt to make the experience of gathering and sharing information easier and smoother by for example allowing users to enter voice-queries which will make the device google the query and show the answer on the device’s screen right in front of your eyes.

In the other corner we can find a different kind of head mounted display like the Oculus Rift (see Figure 2.6) or Sony’s project morpheus which provides a virtual reality to fully immerse the user inside something like a game, movie or other similar media. The user can only see what is shown on the screen in stead of having an overlay on the real world.
2.2 Current Head-Mounted Displays and Cameras

This section will detail about the hardware properties of a head mounted display (HMD). An overview on different properties of current head mounted displays can be seen in Table 2.1. The first three columns are the brand, type (optical see-through, video see-through or just a regular head mounted display) and model (and operating system if applicable).

The fourth column speaks about the input modalities of the head mounted displays. Most head mounted displays are equipped with a range of input modalities like cameras, gyroscopes, accelerometers and magnetometers for head tracking and position tracking. Some of them include light and proximity sensors as well.

Input of the head mounted displays ranges from voicecommands (like Google Glass) to gestures (like the META Pro in Figure ), to external input-devices. The Epson Moverio BT-200 (see Section 2.2.4) is an example of SmartGlasses with an external input device. To illustrate, its control unit has access to Bluetooth and Wi-Fi. Furthermore, it has a touchpad and runs on Android 4.0.

A head mounted display typically consists of at least of some form of screen that can be worn on the head so the screen is in front of one or both eyes. The fifth column of Table 2.1 lists the different kinds of screens of some leading head mounted displays. Two types of displays exist: displays that cover the eyes so you can only see a screen and displays that are see-through. Head mounted displays with a see-through display are called “Optical head mounted displays” (OHMD) Google Glass (see Figure 2.5) is an example of an OHMD and the Oculus Rift (see Section 2.2.4) is an example of a HMD that can not be seen through and only shows computer generated images.

The other columns talk about the ways these head mounted displays produce sound, what their average battery life is, their cost and if they are capable of virtual and/or Augmented Reality.

2.2.1 Augmented Reality and Widgets

This thesis explores visualizations for helping users with an optical head mounted display reach their destination, find objects they are looking for or just give them a general idea of their surroundings using Widgets and Augmented Reality (AR).

Because users will have a screen in front of their eyes, they can be shown extra information directly in their vision. A simple way to do this is with Widgets. These are objects that will stay directly in their vision at all times. An example of this can be a small map of the surroundings in the corner of their vision.

AR is a way to show new information about the real world by adding a new layer on top of it. For navigation this can be something like showing a line on the road to show the path users need to follow to get to their destination.
Despite the fact that the hardware used in this thesis will be head mounted displays (and an OHMD in particular), one should be aware that there are more devices capable for displaying AR and they do not necessarily need to be attached to one’s head (see Figure 2.7 for a list of displays that can display AR). For example, a smartphone with a camera (hand-held display) can be able to show AR too by drawing computer generated objects on top of what the camera displays on the screen.

Fig. 2.7: Different kinds of displays for Augmented Reality [5].
2.2.2 Properties of Displays

The displays can be separated into two classes. Regular displays and see-through displays. While there are many different kinds of displays for both classes, the regular displays can be CRT, LCD, LCoS, OLED-screens (like the Oculus Rift has) or virtual retinal displays (like the AVEGANT Glyph in Section 2.2.4). A virtual retinal display will draw a raster directly onto the retina of the eye in about the same way as a television. An OHMD can be seen through. Their displays use techniques called “curved mirror” (a curved combiner to refocus the image from the projector) and “waveguide” where you let the light enter a thin plastic window’s edge and spread around the display. These techniques can be uses for very small displays (Google Glass uses waveguide for example).

2.2.3 Properties of Cameras

An OHMD has cameras for augmented and mixed reality. These cameras are around the 5 - 8 megapixel range. Some even have two cameras for a 3D-vision like the Atheer One (see Section 2.2.4) for example.

A HMD that is not see-through can only be used for virtual reality. Even with cameras, the user still does not see the real world but just a projection of what the cameras see alongside with possible added computer-generated images. This is called video see-through.
2.2.4 Hardware Overview

Google Glass

Oculus Rift

Epson Moverio BT-200

Atheer One

Pivothead SMART

Vuzix M100

META Pro

AVEGANT Glyph
<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Type</th>
<th>Model</th>
<th>Input</th>
<th>Screen</th>
<th>Sound</th>
<th>Battery life</th>
<th>Weight</th>
<th>Cost</th>
<th>AR vs VR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foxconn</td>
<td>Optical HMD</td>
<td>Google Glass (Android 4.4.2)</td>
<td>5 MP camera capable of 720p video recording, Wi-Fi, Bluetooth, microphone, touchpad, GPS, gyroscope, accelerometer, magnetometer, Ambient light sensing and proximity sensor</td>
<td>see-through</td>
<td>640 × 360 Himax HX7309 LCoS display</td>
<td>Bone conduction, Transducer, microphone, connect with Android device</td>
<td>5 hrs</td>
<td>$1500</td>
<td>AR</td>
</tr>
<tr>
<td>Oculus VR</td>
<td>Virtual Reality HMD</td>
<td>Oculus rift</td>
<td>gyroscope, accelerometer, magnetometer, Near Infrared CMOS Sensor</td>
<td>960 x 1080 per eye virtual reality display</td>
<td>N/A</td>
<td>DC Power Adapter, Head-set: 2.0 mW / 16 ohm</td>
<td>379g</td>
<td>$350</td>
<td>VR</td>
</tr>
<tr>
<td>Epson</td>
<td>Optical HMD</td>
<td>Moverio BT-200 (Android 4.0)</td>
<td>5 MP Camera, Wi-Fi, Bluetooth, compass, gyroscope, accelerometer, microphone</td>
<td>See-Through</td>
<td>0.42 inch wide LCD</td>
<td>Head-set: 2.0 mW / 16 ohm</td>
<td>6 hrs</td>
<td>$699.99</td>
<td>AR</td>
</tr>
<tr>
<td>Atheer</td>
<td>Optical HMD</td>
<td>Atheer One (Android)</td>
<td>2 x 8 MP Camera, Capacitive touch, compass, gyroscope, accelerator, ambient light sensor, smartphone connection</td>
<td>3D stereo see-through</td>
<td>N/A</td>
<td>N/A</td>
<td>70 g</td>
<td>$350 (850 for dev kit)</td>
<td>AR</td>
</tr>
<tr>
<td>Pivothead</td>
<td>Optical HMD</td>
<td>SMART (Android)</td>
<td>8 MP camera, Bluetooth, wi-fi, NFC, microphone, GPS, gyroscope, accelerometer, magnetometer, proximity, ambient light sensor, smartphone connection</td>
<td>LED Lightguide see-through</td>
<td>Speaker + audio in/out jack</td>
<td>3 hours with continuous filming</td>
<td>N/A</td>
<td>N/A</td>
<td>AR</td>
</tr>
<tr>
<td>Vuzix</td>
<td>Optical HMD</td>
<td>M100 (Android 4.04)</td>
<td>5 MP camera, Bluetooth, wifi, microphone, GPS, gyroscope, accelerometer, magnetometer, proximity, ambient light sensor, gestures, connection with Android device</td>
<td>WQVGA Colour see-through display</td>
<td>Ear speaker</td>
<td>6 hrs</td>
<td>N/A</td>
<td>$999.99</td>
<td>AR</td>
</tr>
<tr>
<td>META</td>
<td>Optical HMD</td>
<td>Pro (Meta)</td>
<td>Twin RGB Cameras, Bluetooth, wifi, GPS, gyroscope, accelerometer, magnetometer, gestures, connection with Meta Pro Pocket Computer</td>
<td>two 1280 x 720-pixel LCD displays see-through displays for a 3D Stereoscopic Display</td>
<td>3D surround sound</td>
<td>4 hrs</td>
<td>180 g</td>
<td>$3000</td>
<td>AR</td>
</tr>
<tr>
<td>AVEGANT</td>
<td>Virtual Reality HMD</td>
<td>Glyph (Glyph)</td>
<td>headtracking, microphone, connection to media-device</td>
<td>WXGA (1280 x 720) per eye resolution</td>
<td>premium stereo</td>
<td>3 hrs video, 48hrs audio</td>
<td>450g</td>
<td>$499</td>
<td>VR</td>
</tr>
</tbody>
</table>

Tab. 2.1: Overview on different properties of current head mounted displays

http://www.itworld.com/answers/topic/personal-tech/question/how-long-battery-life-google-glass/

https://www.layar.com/glass/

https://www.indiegogo.com/projects/pivothead-smart
2.3 Research Domains

Researchers have explored the uses of head mounted devices in different scenarios ranging from medical domains, cultural and learning, community settings, smart homes and ubi-comp environments to sports.

2.3.1 Medical

Albrecht et al. [1] used the Google Glass in a forensic setting for using it as a hands-free camera. The camera of google glass was compared with a DSLR camera. They concluded that it took less effort than a DSLR camera but that the Google Glass still performs inferior in this setting.

The Google Glass was also used in a pharmacy by Fox. et al. [12] where they used it to document medication administration or as a tool to support the dispensing process. Google Glass would be used to replace documents and as a way to easily access information. They also noted that privacy is a concern.

Another medical application of the Google Glass by McNaney et al. [23] was with people with Parkinson’s disease. Google Glass is used mostly for reminders and prompts. They conclude the technology is promising but that future work will also need to address several of the basic functions of Glass, such as the voice recognition and navigation gestures, to ensure it can accommodate for usability issues caused by PD symptoms.

Fig. 2.8: Google Glass in a forensic setting. [12]  
Fig. 2.9: Google Glass used for keeping track of pill intake. [23]
2.3.2 Cultural and Learning

Chen et al. [7] developed TelePort, which gives users the ability to explore a 3D space through a first-person navigation to obtain a reality experience of moving in a certain 3D-environment. They used it to let people experience walking through a cave system.

Sapargaliyev [1] speaks about ways Google Glass could be used in educational settings. An example was a youtube-channel of which the owner explains situations while wearing a Google Glass so he can use both his hands.

![Fig. 2.10: A user looking around in a virtual cave using Teleport]  ![Fig. 2.11: Google Glass used for an educational youtube-channel](image)

2.3.3 Community

Baldauf et al. [3] wanted to create a system to enable people to raise their voice for citizen e-participation. They want people to be able to take part in decision-making processes any time and anywhere through contemporary and upcoming ubiquitous technology resulting in a continuous dialogue between a city and its residents. They used Google Glass so people could literally share their point of view on the city-planning.

Another community-system called Mercator was developed by Simoens et al. [37] which applications can be built because the reconstruction of depth information from 2D images is compute intensive and results in loss of accuracy. The model is continuously updated, refined and expanded by crowd-sourcing depth data from 3D cameras on head-mounted devices such as Google Glass.

Chen et al. made QuiltView [8]. QuiltView’s aim is to create a new kind of near real-time social network. People capture first-person videos with devices such as Google Glass to respond with a small video to certain queries that were asked by other QuiltView users in the area.

1 [http://cloudworks.ac.uk/cloud/view/8612](http://cloudworks.ac.uk/cloud/view/8612)
2 [http://www.youtube.com/user/STEMbite](http://www.youtube.com/user/STEMbite)
2.3.4 Smart Homes and Ubi-comp Environments

Simoens et al. used the Google Glass’ camera in combination with a neuro-headset to create a system called Vision to get the subject’s first-person view and control smart homes only by looking at certain objects and thinking about them.
2.3.5 Sports

The Google Glass was also used in a sports-setting where it could offer a hands-free user interface that’s more convenient than traditional speedometers, and it provides instant performance feedback and context-aware notifications overlaid on the biker’s view. This was made by Sörös et al. and named Cyclo [38].

Nguyen et al. used Google Glass to create an application called Fitnamo [29] to encourage exercise by having Augmented Reality games that require you to exercise to play them.
3. AR NAVIGATION-HELPERS IN OPEN WORLD GAMES + RESEARCH

3.1 Related Research in the Area of AR-Navigation

Research in the area of Augmented Reality (AR) already investigated the use of Head Mounted Displays (HMD) for navigation in the real world. The real world vision of the users is overlayed with augmented information (e.g. arrows or labels). This allows the users to focus on the environment while keeping them from being distracted by looking at maps or additional screens. This section discusses what kind of visualization techniques from Open World Games (OWG) can be used in real world situations and how current research is developing similar techniques for real world navigation.

![Fig. 3.1: View shot through the see-through headworn display, showing campus buildings with overlaid names. Labels increase in brightness as they near the center of the display [11].](image1)

One of the first AR-systems for exploration is the system developed by Feiner et al. [11]. Users could walk around the campus with a big backpack, a head mounted display and a hand-held display (see Figure 3.2) and get information of the buildings around the campus shown on top of them. Buildings can be selected by using the hand-held display. After a building is selected, a green compass-arrow on the bottom of the screen will start pointing in the direction of the building.

![Fig. 3.2: The backpack, HMD and hand-held display needed for the prototype campus information system.](image2)
The MARS-system is an AR navigation-systems developed by Höllerer et al. [14]. They developed MARS, a system for navigating through an unknown area by creating a path and flags layered on top of the real world. The user had to wear a head-mounted display and a backpack as can be seen in Figure 3.4 similar to the one in Figure 3.2 while navigating through the environment. In addition to the real world, users see a 3D-path and flags rendered on top of it (see Figure 3.3).

The CyPhone application by Pyssysalo et al. [30] was designed as an AR based personal navigation service for “future mobile phones”. Figure 3.5b shows four different AR-visualizations used in the application for helping users reach their destination.
Reitmayer et al. [31] developed a tourist guide application for the city of Vienna built with \textit{Studierstube} as a software platform for developing AR applications. It uses AR for showing information about the city and its buildings. They used a Sony Glasstron optical see-through stereoscopic color HMD fixed to a helmet as an output device.

It is an AR application for a head-mounted display that shows points of interest and additional information about the landmarks of the city of Vienna. The \textit{Studierstube} system could render this information to the landmarks and streets as 3D-rendered objects as can be seen in Figure 3.6a and Figure 3.6b. It has functionalities for navigation, information browsing and annotation.

For navigation their system can show a path overlayed on the real world like in Figure 3.6b. It can even be clipped so that objects in the real world can obstruct the view and hide part of the path that would be hidden if it existed in the real world. The path can be drawn to follow another user, another user can “guide” you by selecting your destination or two users can “meet” and let the system generate a path for them to meet halfway. Information browsing is done by having the user look at a building and use a touchpad to select a part of the building that is highlighted (see Figure 3.6a). Once selected, a 2D-information box will appear near the edge of the screen showing information about the selected object.

Tenmoku et al. [41] developed an AR system that uses an inertial sensor for retrieving users’ orientation and a pedometer for finding out users’ positions. Their system can annotate the user’s view with arrows that show to way to certain objects and it can display additional information about these objects in a text field with an image (see Figure 3.7).
Sawano et al. [33] explore AR for car-navigation. Their ideal display (as can be seen in Figure 3.8) draws a line on the road that has to be followed as well as highlight traffic lights. There are also some widgets that show the map, a compass, some textual information and a clock.

Burigat et al. [6] used the LAMP3D system to create a mobile tourist guide by having a virtual environment on a PDA that relates to the real world. Users can navigate manually in this virtual environment or by using GPS-coordinate. They can tap objects on the PDA with a stylus (see Figure 3.9a) that triggers a window to open and display information about the tapped object (see Figure 3.9b).

Sawano et al. [32] developed an AR system for car-navigation where the user has a “photo-realistic” road overlayed on top of the real road to show the path the driver needs to follow (see Figure 3.10). It also shows a clock, some textual information and a compass.
3.1. Related Research in the Area of AR-Navigation

Tonnis et al. [42] conducted a study to find out what kind of AR-objects are best used so they can still be correctly perceived over large distances. Because the resolution of see-through displays is lower than the human eye, objects can get pixelated. They compared 3D-arrows with soft and hard corners, and 2D-arrows. In their study they concluded that 3D-arrows with hard corners were best perceived over large distances. Figure 3.11 shows an example with rounded corners.

Kim et al. [17] try to find a solution for the spatial cognition for elderly drivers. They propose an AR display on the windshield. With this technique they visualise the route drivers need to take by transitioning the real road into a “floating” map. They also show extra information in text-format (see Figure 3.12).

An AR Platform for Maritime Navigation is explored by Hugues et al. [15]. In their paper they describe the following objects that are used in navigation software and shown on charts.

- WayPoints (WP): Object representing a buoy signalling a specific geographical position
- Route: Succession of points that the user needs to plan a route
- Trace: Succession of points where the vessel has already sailed
- Targets: Two major families of targets: ARPA and AIS
One of their suggested “views” uses AR to display information (waypoints etc.) on the real world while having a virtual reality thumbnail in the corner of the view to show additional information (see Figure 3.13).

Fröhlich et al. [13] developed a system for simulating different visualizations for displaying safety-related information to drivers. One of their visualization techniques can be seen in Figure 3.14 and augments the road ahead with a line that displays the route the driver has to follow as well as with possible arrows to clarify locations where something happened.

Mulloni et al. [28] conducted a study to see where AR helpers on smartphones would be used most. They hypothesised that users would use it more at static locations (e.g. to verify that they are about to take the right turn). Their results show, however, that users tend to use the AR helper while walking. Their AR helper was a green arrow layered on top of the road that was pointing in the direction users had to follow (see Figure 3.15).

A handheld indoor navigation application was developed by Mulloni et al. [27] Their activity-based instruction application has two views. A world-in-miniature (WIM) view when users stop at an information point which shows the entire trajectory, and an AR-view which shows the current instruction and an arrow on the floor pointing in the direction users need to walk to (see Figure 3.16). Both views have the current task on the bottom of the screen visualised by using icons and text.
Medenica et al. [24] compared a simulated AR personal navigation device (PND) with a streetview personal navigation device. Their AR device used a head-up display (HUD) and showed the path users had to follow by visualising a yellow floating path above the road. From their user study they noticed that while using the AR PND, participants spent about 5.7 sec and 4.2 sec more each minute looking at the road ahead in comparison to a standard PND (regular GPS-system) and streetview PND (shows the real world with navigation advice layered on top of it on the regular GPS’ screen), respectively.

![Highlighting an interaction point through a soft border visualization.](image1)

![The navigation system mockup.](image2)

**Fig. 3.18:** Indoor navigation with vision-based localization by Möller et al. [25]

Möller et al. [25] explored the use of AR with vision-based localization for indoor navigation with a smartphone. They propose to use interaction points which can be highlighted as can be seen in Figure 3.18a and while navigating an arrow will be shown on the floor, leading users to their destination (see Figure 3.18b). They also propose to use virtual reality for showing the surroundings so that the users do not need to keep pointing their phone to everything but can just “swipe around” a virtual environment on their phone to take a look at everything.

![AR View as proposed by Jang et al.](image3)

![The AR interface that was used for comparison in the study of Düüser et al.](image4)

**Fig. 3.19:** AR View as proposed by Jang et al. [16] **Fig. 3.20:** The AR interface that was used for comparison in the study of Düüser et al. [10]
Jang et al. [16] present a scenario-based approach to developing a mobile AR system which uses spatial information. They propose an AR view with an arrow that points to the users’ destination along with the path they have to follow so they do not need to switch between AR view and map views (see Figure 3.19).

Dünser er al. [10] compared three different views for navigating: a map, AR + map and AR. Their AR interface (see Figure 3.20) has a radar that shows nearby waypoints. Way-points that are within a certain distance from the user are shown as billboards on the AR overlay.

PerPosNav was developed by Schougaard et al. [34]. It is an application for indoor navigation and uses four different kinds of navigation: Augmented Signs, Mobile Map, Mobile Auditory, Mobile AR. Two of them use AR. The mobile AR mode requires users to hold a smartphone in front of them and will display arrows and a path to help the user navigate through the environment. The Augmented Signs are physical signs at certain locations in the building that display navigation instructions that are relevant to a single user who’s standing next to it (see Figure 3.21).

Li et al. [18] developed an AR application and hardware for helping with navigation while riding a Segway. They argue that regular GPS-systems can not be used while riding a Segway because of ambient noise that restricts users from hearing where they need to go. Their system uses AR to show the path (see Figure 3.22) users need to follow and give vibro-tactile feedback at every turn.

GeoTrooper is an application developed by Cummings et al. [9] and is a multimodal interface that uses haptic feedback and AR to deliver navigation information to paratroopers in the field. It uses a phone’s camera to detect a beacon and display it on the phone’s screen with annotated information (see Figure 3.23). The phone is attached to a soldier’s head-gear and vibrates when soldiers are walking towards the beacon.
Turicel is a mobile application developed by Mata et al. [21]. The application consists of an AR system that provides navigation facilities, generation of itineraries and services delivery. When users want to get information about a given landmark, they need to point their smartphone at it. The application will show relevant icons or information on top of the landmarks as can be seen in Figure 3.24. Users can press these icons to open a new window for detailed information.

INSAR is a smartphone application developed by Alnabhan et al. [2] which was implemented and tested for navigating around a college campus. It uses a Wi-Fi fingerprinting technique for positioning. Users select a destination from a list and then the system will display an arrow on the screen that points in the direction the user has to go to (see Figure 3.25).

Li et al. [19] present Shvil, an AR system for collaborative land navigation. The system has an indoor user who can define paths on a physical model with an AR application, and an outdoor user who can see the path overlayed on the real world on the outdoor user’s handheld display (see Figure 3.26).
The Personal Navi was developed by Bark et al. [4]. It is a vehicular AR navigational aid designed for use with a see-through 3D volumetric Head-Up Display (HUD). The interface uses virtual airplanes flying above the street, indicating the path the driver needs to follow (see Figure 3.27). They chose for the airplanes above the street because earlier prototypes where they drew a line on the street or in the sky, made them conclude that these techniques take away the attention too much of the view directly in front of the driver.

Mata et al. [22] developed an AR navigation guide based on data from social media. The system recommends places to visit and events to attend by using implicit information derived from a large tweet (twitter) repository related to Mexico city (where the system was tested). The application shows information layered on top of objects. Figure 3.28 shows an AR view for displaying popular places.

The swiss firm WayRay is developing Navion[1] see Figure 3.29). Navion will project holographic images directly on the windshield of your car from a device on your dashboard. This way you could use this in any car. Navion will display navigation information and safety notifications on your windshield. It will be controlled using hand gestures.

Fig. 3.27: The Personal Navi interface. [4]  
Fig. 3.28: The social navigation guide system by Mata et al. [22]  
Fig. 3.29: Navion showing the way by projecting information on the windshield.

[1] https://wayray.com/navion
3.2 Visual Navigation-Helpers in Open World Games

Open World Games (OWG) are games where the player can freely roam a world and complete objectives at the player’s leisure in stead of being forced to follow a linear story/environment that the game presents to the player. I looked through a list of 100 OWG\(^2\). Next I compared the used AR-helpers and Widget-helpers for the tasks: navigation, exploration, orientation and annotation (and labels).

3.2.1 Navigation

Navigation is the task for the user for getting from point A to point B. Many games use digital maps or mini-maps in the corner of the screen as at least part of the solution for helping the player. Some games add extra information to the 3D-world. One way can be by directly highlighting the road that needs to be followed by adding 3D-objects on it. The game “The Elder Scrolls V: Skyrim” does this by giving its players a spell that summons a line of fog on the road (see Figure 3.30) that needs to be followed. Similarly, a widget can be used for this task. A mini-map for instance. The game “Saints Row 4” places an icon on the mini-map for showing the player’s destination (see Figure 3.31).

![Fig. 3.30: A player casting the spell “Clairvoyance” that will show them the way.](http://gamingbolt.com/100-greatest-open-world-games)

![Fig. 3.31: The player’s destination is marked on the mini-map.](https://www.youtube.com/watch?v=CD5x8eL31rU) at 8:42

\(^2\) http://gamingbolt.com/100-greatest-open-world-games

\(^3\) https://www.youtube.com/watch?v=CD5x8eL31rU at 8:42
3.2.2 Exploration

Exploration is a navigation-task for users who have no certain destination but want to know/see more about the surrounding area. The game “Tom Clancy’s: The Division” employs a system the developers call the “mega map” (see Figure 3.32). This system brings up a virtual 3D-map that surrounds the playable character and displays information about nearby buildings, districts or other objects of interest. The player can use a cursor to select an object on this map to get specific information about this and zoom in on it. The widget-version of this is once again the mini-map. The mini-map can place icons or other information on top of objects to display their functions like e.g. in the game “Red Dead Redemption” (see Figure 3.33).

![Fig. 3.32: A player who is selecting a building on the mega map](https://www.youtube.com/watch?v=1sUXWiqGLqo at 16:52)

![Fig. 3.33: The mini-map displays what functions the nearby objects have by marking them with an icon](https://www.youtube.com/watch?v=cVBsIuTT5iY at 0:34)

3.2.3 Orientation

Orientation is the task to give the user a grip on his/her surrounding. In this scenario the user is not exploring but might want to gain a quick idea of what his/her local surrounding has to offer. A way games solve this is by using some sort of “heat-vision”-system which allows the player to see objects of interest through buildings, walls or other irrelevant objects. The game Batman: Arkham Origins has a so-called “detective vision” and shows non-playable characters (NPC) and objects the player can interact with in a highlighted colour while everything else stays in a dark blue tint (see Figure 3.34). Highlighting a goal by letting a pillar of light come out of it for example, is another AR-visualization for this task as well. The game “Borderlands” uses a compass widget and displays objectives with a diamond-shape on it when the player is looking in the right direction (see Figure 3.35).
3.2. Visual Navigation-Helpers in Open World Games

3.2.4 Annotation + Labels

The annotation-task’s purpose is, after an object is located or is in sight, to display to the user what an object has to offer for him/her. Games essentially use two different systems for this. The first one is by directly showing an annotation on the object and telling the player what it is/does with text/icons. An example of this is the game Splinter Cell: Blacklist which straight up puts big text on objects, indicating what it is or does (see Figure 3.36). One could assume that almost all games use a widget for this. Almost every game displays text for giving the player information. A more original way for displaying a text for annotation is used in the game “Batman: Arkham Origins” where it would show text on the head-up display (HUD) but still have a line going from the text-box to the relevant object (see Figure 3.37).

---

6 https://www.youtube.com/watch?v=SJmR_f0X-zw at 1:58
7 https://www.youtube.com/watch?v=GD0NdCobEJ at 1:34
8 https://www.youtube.com/watch?v=vog0PVvnBKU at 7:13
9 https://www.youtube.com/watch?v=SJmR_f0X-zw at 1:54
4. STUDY COMPARING VISUAL HELPERS IN OWG AND IN RESEARCH

4.1 Visual Helpers in Research

A list of 25 research-works were analysed and categorised within four categories of visual helpers: Navigation, Orientation, Annotation and Exploration. AR was inspected for these 25 works.

Fig. 4.1: Percentage of research that uses a certain AR-helper in function of the time.

Section 3.1 has a list of related research in the area of AR-navigation. When we categorise the applications and research-works from this list into the four tasks as discussed in Section 3.2, we gain the following information about helpers in related research-works by 2015 (as you can see in Figure 4.1):

- 76% Of the works use navigation-helpers.
- 8% Of the works use exploration-helpers.
- 24% Of the works use orientation-helpers.
- 48% Of the works use annotation-helpers.

Research sees Navigation (getting users from A to B) as the most important task for AR navigation-applications with Annotation being the second most important.
4. Study Comparing Visual Helpers in OWG and in Research

Fig. 4.2: Percentage of research that uses an exact certain amount of different categories of helpers.

Fig. 4.3: Percentage of research that uses at least a certain amount of different categories of helpers.

From the previous two graphs (see Figure 4.2 and Figure 4.3) we can conclude that research-works tend to stick to only one or two different categories of helpers (being mostly Navigation and Annotation). Around half of research uses exactly one helper and 40% of research uses exactly two helpers. On the second graph (see Figure 4.3) can be seen that 48% of research uses two or more categories of helpers in their work.
and only 8% uses three different categories. An example that uses three different kinds of helpers is the tourist guide application than can be seen in Figure 3.6. This application helps with navigation, orientation and annotation. One could also argue that it helps a bit with exploration too because it displays information of nearby buildings that users can see. However, it does not show any nearby buildings that are not in a user’s vision.

Looking at this information, it is clear that research will focus mostly on one or two elements in their applications in stead of something combined. And these two will primarily be Navigation and Annotation.
4.2 Study of 100 Open World Games

Appendix A contains a study of 100 different OWG. In this study, all kinds of helpers using AR and/or widgets during regular gameplay were identified and categorised (see Section 3.2 for the differences between these helpers). The relevant helpers are:

- AR-helper for Navigation
- AR-helper for Exploration
- AR-helper for Orientation
- AR-helper for Annotation+labels
- Widget-helper for Navigation
- Widget-helper for Exploration
- Widget-helper for Orientation
- Widget-helper for Annotation+labels

The last item on this list is not included on the following graphs because all games use some way to display some text on the screen. Also keep in mind that accumulated numbers of games will be used in the following graphs. This means that for earlier dates, less games will have been studied than at later dates (e.g. 1998 only contains data for eight games while 2015 has data for 100 games because this also contains the data for the previous years). The red lines in Figure 4.4 and Figure 4.5 make the amount of games at a certain date clear.

From this study, the following results were found:

![Fig. 4.4: The accumulated number of games that use a certain helper.](image-url)
4.2. Study of 100 Open World Games

Fig. 4.5: The number of games that were released in a each year that use a certain helper. The accumulated version of this graph can be seen in Figure 4.4.

The number of OWG (OWG) is growing rapidly as the top red curve on the graph in Figure 4.4 and Figure 4.5 clearly shows. From this same graph, one can conclude that these games prefer widgets over AR for helping players. AR-annotation is the exception to this rule however but is mostly used in addition to others. This graph makes it seem that widgets are the most important helpers in OWG. One must also note however, that most of these games use a mini-map that it its own can help within different categories (e.g. the mini-map in Figure 3.31 helps with both orientation and navigation).

The most important tasks for widget-helpers in OWG are orientation and navigation (see Figure 4.6). They help with these tasks by using compasses and mini-maps for example. For AR, Annotation (at 81% of OWG using this helper) is the most important helper. Less than 50% of OWG use AR for the other navigation tasks.

Figure 4.6 also shows that the usage of AR-navigation and AR-exploration in games has been stable for six years now. The curve for AR-navigation has stayed at about 20% of OWG using this helper the last four years. The curve for AR-exploration has been stable for 6 years at about 6% of OWG using this helper. One can also see that AR-orientation became more important than AR-navigation after 2003 and is slowly rising. At the same time Widget-navigation is rising fast. This means that widgets are favoured over AR for navigation but some games still keep AR for orientation as an extra.
Fig. 4.6: The percentage of games that use a certain helper.

This graph also confirms that Widget-orientation (e.g. a compass) and AR-annotation (e.g. displaying names above buildings) are the most important as 80% of OWG use these.

Fig. 4.7: The percentage of games that use a minimum amount of different helpers.
The rise of the processing power of computers can be seen on Figure 4.7. Older games (early 90’s) only use two or three different helpers while the further we look in time, the more helpers are used in OWG. After the year 2000, games suddenly start using even more than 5 different helpers because even household computers are starting to get powerful enough for more complex calculations.

![Figure 4.8](image)

*Fig. 4.8: Percentage of games using both Widgets and AR.*

Figure 4.8 shows a graph that displays the percentage of games that use both AR and Widgets (Widget-annotation not included because this is merely displaying text somewhere and that is something every game does). Another fast rise can be seen on this graph. This indicates that as time goes on, more and more OWG favour to use a mix of both.

From Figure 4.9, we can tell that helping users with navigation tasks in OWG is an important task by looking at games that use exactly five or six different helpers. The percentage of games using five or six different helpers started rising very fast around 2005. In 2015, OWG tend to favour using five or six different kinds of helpers. Something else that is noteworthy is that after the late 90’s, games using only one kind of helper have been declining except for a small rise around 2010 after which they started declining again.
Another thing we learn is that games try to solve all the navigation-tasks in the first place by using widgets and have AR as “an extra”. Figure 4.10 shows that 60% of games use widgets for all four navigation-tasks by 2015. Games that only use at least two different AR-helpers take up only 43%. And only 4% of OWG use AR for all four tasks.
Figure 4.11 tells us that by 2015, games tend to use one AR-helper (annotation + labels mostly) and four widget-helpers as these games present the two highest curves on the graph. The same graph also shows a decline in games that use one or two widget-helpers. This is an indication that more recent games use better widgets that can help with multiple tasks (e.g. a mini-map that can help with orientation, navigation and exploration).

By 2015:

- 81% of the games use a widget-helper (excluding annotation) and 85% of games use an AR-helper. When we exclude annotation as a task however, only 48% of games use AR-helpers for the other tasks.
- 6% of games use AR for helping with exploration while 62% use widgets for this task
- 20% of games use AR for helping with navigation while 69% use widgets for this task
- 41% of games use AR for helping with orientation while 81% use widgets for this task
- 80% of games use AR for annotation and ±100% of games display text on screen in some way.
4. Study Comparing Visual Helpers in OWG and in Research

4.3 Interpretation of the Data

When comparing games to the research in Section 3.1 by looking at the graphs in Section 4.2 and Section 4.1, we can draw some conclusions.

OWG will use annotation as the most important AR-helper in contrary to navigation being the most important in research.

The AR-helper that is used the most in OWG is annotation, while navigation is the most used helper in research (see Figure 4.1). Orientation is the second most important helper in OWG. This means games favour helping the player with exploring the world in stead of just getting them from point A to point B which is logical because that is one of the biggest parts of OWG is the exploration of its world. Research tends to look for more practical applications.

When comparing the different visualizations for the AR-navigation helpers in the research-works, we see that they always more or less look the same (colours and size being the biggest differences). The research-works tend to use lines or arrows drawn on the ground or floating above the ground for visualising this helper. There are a few exceptions like The Personal Navi which uses green paper planes as a visualization (see Figure 3.27) or INSAR which has a red 3D-arrow that points to users’ goals (see Figure 3.25). OWG have more variation in the visualization of their helpers, usually fitting around the theme of the game.

Because navigation is the most important helper for research, we can take a look at OWG to see how they visualise their navigation-helpers with the aid of AR.

The first game on the list in Appendix A that helped with navigation in an “AR”-helper was Grand Theft Auto (Section A.1.40). There is a small yellow dot that floats next to the playable character and points in the direction of a current goal.

Fig. 4.12: AR-navigation helper in GTA 1.

The first game to use a 3D-object for this task was “The Simpsons: Road Rage” (Section A.1.1). The game had a floating hand that points in the direction of the objective. Much like the arrow used for INSAR (see Figure 3.25).

Fig. 4.13: AR-navigation helper in The Simpsons: Road Rage.
Another way for visualising navigation-helpers is used by Need for speed: Most Wanted (Section A.1.54) which uses 3D-holograms of arrows that also act like barriers to point the player in the right direction.

The game that first used the most used navigation-visualization in research (showing a line in some way or another that users need to follow) is the game Okami (Section A.1.38) which put sea urchins near the coast in a line which lead to an object of interest.
By looking at the AR-visualizations for navigation used in the OWG in Section 4.2, we can conclude that in general there are three visualizations for navigation-helpers with the aid of AR:

1. Displaying a line or arrow on the route users need to follow. This can be a real line or a sequence of objects.

2. 3D-objects on the path that point users in the right way.

3. A 3D-object that stays near the user that points towards the goal.
The second most used AR-helper in research is annotation, which is the most important AR-helper in OWG (see Figure 4.1). The usage of AR-annotation in research has had a significant drop of 35% the past nine years which currently puts it at 48% (see Figure 4.1) of research using AR-annotation helpers.

When comparing the visualizations used in research to the ones used in OWG, one can conclude that both show text or icons above or near objects (or highlight them) but that some OWG also use a more “immersive” variant where they attempt to merge the text-annotation with the 3D-object.

Highlighting objects can be found in both OWG and research but only scarcely in research. The indoor navigation system by Möller et al. [25] (Section 3.1 and Figure 4.16) uses highlighting as an annotation-helper. A game where this is used is Fallout 3 where the part players are aiming at is highlighted (see Figure 4.17).

Below is the Turicel application (see Figure 4.18) which uses an annotation-helper compared to annotation-helper from the game State of Decay (see Figure 4.19). Both display regular text and/or icons next to the object of interest.
4. Study Comparing Visual Helpers in OWG and in Research

OWG also sometimes have a more immersive way of annotating objects with text. They do this by merging the text with the object and adding the text as an extra layer on top of the object. See Figure 4.20 for an example.

![Image](infiltrate_the_mansion.png)

**Fig. 4.20:** Splinter Cell: Conviction shows mission details on the building where this mission will take place.

4.4 Real World Navigation

Section 4.3 drew some conclusions from the gathered data and compared the helpers in OWG to helpers used in research. The motivation for this is that OWG need to help players with navigation just like in the real world. This rises the question whether lessons about the visualization AR-helpers can be learned from OWG. The graph in Figure 4.4 showed a fast increase in OWG, so this means these games are successful and the same counts indirectly for the AR-helpers in these games. So which visualizations would prove the most useful in the real world?

OWG want the player to explore their world. This means that helping getting the player from A to B is less important than showing players what they can do in the world. Helping with navigation (getting from A to B) is (in games with a modern setting) something that is mostly used once the player enters a vehicle. While players are on foot, they will mostly see helpers that show what is in the area around them.

A game that can be an example of this, is the game “Watch Dogs”. While the player is walking around, their character will be aided mostly with annotations (see Figure 4.21).

![Image](watch_dogs.png)

**Fig. 4.21:** The game Watch Dogs annotates surrounding objects.

---

1 [https://www.youtube.com/watch?v=Lrro1GE9JUI](https://www.youtube.com/watch?v=Lrro1GE9JUI) at 2:34
Upon entering a vehicle however, another AR-helper for navigation can be used by letting players select their destination on a map. A line will appear in front of players' vehicles on the path that they need to follow to get to their destination. See Figure 4.22.

Some OWG like this one choose to divide helpers for navigation based on method of transportation and indirectly on the player’s speed. When players are walking, they will have more time to take a good look at what is around them, while travelling by vehicle means they just want to get somewhere fast and they could no longer have the time to read what every individual object does.

Perhaps this means annotations are more effective at a lower speed while navigation-helpers like the ones in Section 3.2.1 can be used at relatively high speeds. Both versions will be tested in this thesis for scenarios where users are travelling on foot.

Finally, one needs to take into consideration that there can be different kinds of visualizations for these helpers. Section 4.3 talked about three different visualizations for Annotation (highlighting, displaying text near objects and merging the text on the object) and three different visualizations for Navigation (“lines” in front of users, 3D-objects on the path and 3D-objects near users that point to the objective).
Below are some AR navigation-helpers used in OWG (see Figure 4.23).

The last two AR-helpers in OWG (Orientation and Exploration) can prove useful for navigation at slower speeds as well because they can provide a lot of information but require some time for processing the information.

OWG help with Orientation by showing a 3D-object near the objectives of players so that once players are near, they can immediately identify where the game wants them to go. The game Driver: San Francisco shows a big hovering blue circle around the way-point players need to reach (see Figure 4.24).

Fig. 4.23: Different kinds of AR navigation-helpers used in OWG.

Fig. 4.24: Driver: San Francisco shows a big hovering blue circle above a way-point.
Another form of using AR for orientation is used by the Assassin’s Creed series. When activating a certain function in the game, everything in the game will take a dark shade of blue except for objects that players can interact with (see Figure 4.25). These are in turn coloured in different colours based on their function to the players (e.g. enemies will be red).

Players need to know what every colour means in order for this visualization to be optimal but they will at least get a sense of where everything that can be interacted with is located in their direct environment.

Fig. 4.25: All moving objects are highlighted in a different colour based on their function.

Fig. 4.26: Different kinds of AR orientation-helpers used in OWG.
The final helper used in OWG is Explo-
ration. This helper barely has any AR
visualizations (most games just use a
mini-map for this) but a game that uses
an AR-visualization for exploration, is
the game “Tom Clancy’s: The Division”
where players can look at a “mega-
map” that surrounds them and shows a
3D-version of their direct environment
around them (see Figure 4.27).

In order to know the function of objects around them, players can individually select
these objects and see what they do. But a disadvantage to this is that players can
not move while doing this.

While looking at the AR helpers used in OWG, one can draw some conclusions
about when certain visualizations are used. Exploration-helpers can be useful for
showing the direct environment to users in an original way but could require them
to stand still while doing this as if they were looking at a real map to figure out
where they need to go. Orientation can be used while moving but would be most
effective at slower speeds so users can take their time to look at the individual ob-
jects. It is possible for using Orientation-helpers at higher speeds but developers
need to make sure their players will be able to read everything. The same is true for
AR-annotation. While travelling at higher speeds, users would not have the time
to read everything that moves by them if there is a lot of text. One solution is to
use annotations for objects that are very far away so it would take some time before
users pass the object. However, this could be distracting while driving a car where
users need to keep their focus on the road ahead. That problem is fixed by using
AR navigation-helpers where users are shown the path they need to follow while
still allowing them to look at the road directly in front of them so they do not get
distracted.

When travelling on foot, users go slow enough to allow for an environment where
every visualization can be useful. We wonder which combination of widgets and
AR-helpers creates the best aid for users to reach their destination.
5. CONCEPT

Open World Games use a great amount of visualizations to help players with navigation, orientation, annotation and exploration. With the help of Widgets and AR, this thesis investigates if visualizations inspired by ones used in OWG can be effective in real-world situations. Using Google Streetview, Unity and the Oculus Rift, a simulation was created for users to experience these visualizations in virtual reality scenarios. These visualizations were based on the different kind of visualizations used in the studied list of OWG.

At first, a simulation was created using the Google Cardboard SDK for Android (OpenGL ES) but was later later discarded for the Unity engine along with the Oculus Rift DK2 because at the time of development of the Google Cardboard app, Unity had no SDK for Google Cardboard yet. The Google Cardboard app that was created had capabilities of looking around in a 360° image and rendering the outline of buildings (floor-plan) based on the building’s coordinates and a set of manually created georeferences to align the map created with the coordinates with the building. In comparison to OpenGL ES, the Unity engine allows for a much easier way to create and edit 3D-objects.

5.1 Augmented Reality

Augmented Reality visualizations are 3D-objects layered on the real world that are not necessarily always in a user’s vision.
5.1.1 AR Navigation

Fog Path

The path users need to follow is shown by having a trail of fog float over it. It was created as an extra to resemble the helper in Figure 3.30 and to see if it has a different impact than the “Highlighted Path”.

Object Path

The same path as the “Fog Path” is used but the Object Path uses a series of objects (orange diamonds in this case) that are put next to each other to show the path to the user’s destination.

Highlighted Path

This kind of visualization is a more “classic” way of showing the way by just having a regular line on the ground leading to a user’s destination.
5.1. Augmented Reality

Moving Arrow

The moving arrow will float from the user’s location towards the current goal for the user. This can be the next intersection on the road or a user’s end destination.

3D Arrow Next To User

An arrow will be shown at all times that floats directly in front of the user’s vision and stays there. This arrow will point to the direction users need to follow.

Static Arrows

Static arrows are, like the “Object Path”, different objects on the path but the difference is that these arrows will only be shown on “key positions”. This can be one near the user to show the direction they need to follow and then another one near an intersection or bend.
5. Concept

5.1.2 AR Orientation

Heat Vision

The “heat vision” metaphor is used to make it easier for users to find certain objects/targets. By turning the overall colours to blue and turning important objects orange, these objects will really stand out and make it easy for users to find them. The important part is that these objects can also be behind other objects or inside buildings and will still be made visible.

5.1.3 AR Annotation

Object Names

A simple yet effective way to annotate objects is by just displaying their names or functionalities right on top of them. In this case the names of buildings and roads are displayed floating above them in plain text.

Object Icons/Logos

This visualization is more or less the same as with “Object Names” except that this displays images. This implementation shows the logos of the different companies that reside in the buildings.
Highlighting

Highlighting objects uses a similar approach as “Heat Vision” where important objects are changed in colour to stand out more. The difference is that these objects should be visible without the need of “heat vision”. Highlighting makes them easier to find by adding some colour to them.

5.1.4 AR Exploration

Mega Map

This visualization is inspired by the visualization used in the game “Tom Clancy’s: The Division”. It shows a small 3D-model of the local area and the names of buildings will be shown on the HUD when looking at them.

5.2 Widgets

Widgets are mostly 2D-objects that will stay in a fixed position within the vision of users at all times.

5.2.1 Widget Navigation

Mini-map With Marked path

A minimap will be shown on the HUD with a blue line drawn from the middle (the user’s location) towards the user’s destination.
5. Concept

2D-Arrow On HUD

This arrow will more or less take the approach of the “3D Arrow Next To User” used in Section 5.1.1 except that this is in 2D. Users will have a top-down view of this arrow on their HUD and the arrow will point towards their current objective/destination.

5.2.2 Widget Orientation

Compass

The compass will show users their heading using wind directions. A small diamond placed on the compass tells user the direction they need to follow. This compass has the form of a rectangle on which the wind directions will move along with the user’s heading.

Radar

A simple version of a mini-map is the radar. It only shows the bare necessities which is the direction of the current objective/destination based on the heading of the user. A green dot that moves along the edge of the radar will show users the direction they need to follow. When the dot is on the top part of the radar, it means users are headed in the right direction.
5.2.3 Widget Annotation

Minimap Annotations

Using the mini-map, different objects like buildings will be given extra information by displaying their name on the shapes of the matching buildings/objects on the mini-map.

Minimap Icons

Like mini-map annotations, icons will be displayed on the shapes of the matching buildings/objects on the mini-map. The icons will help the user find out what functions these objects have.

5.2.4 Widget Exploration

Mini-map

Exploration is a task for users to find out what is in their surrounding. These kind of helpers mostly just show the user where to find objects. In this case, a mini-map is used without further annotations so the user knows where to find objects, buildings and streets in their direct surrounding.
6. IMPLEMENTATION

6.1 Components

Fig. 6.1: Pipeline of used tools. The purple steps are used in both prototypes. Orange is for the first prototype and red for the second.

The application used for user tests in this thesis uses the Unity engine with the Oculus Rift SDK for showing navigation visualizations. The Google Streetview API is used for getting images from google streetview at certain positions. Lastly Microsoft’s Image Composite Editor is used for stitching together the Google Streetview images into a 360° image for displaying around the user along with Google SketchUp for creating 3D-models.

An early prototype was created using the Cardboard SDK for Android but was not further used later in favour for using the Oculus Rift and the Unity game engine. They share the use of the Google Streetview API for downloading pictures (Section 6.1.3) and Image Composite Editor (see Section 6.1.4) for stitching them. Then the Google Cardboard prototype was only able to generate a simple floor-
plan based on Google Maps coordinates and georeferences (see Figure 6.4) around a real building (see Figure 6.2), whereas the Oculus Rift prototype contained the previously discussed navigation-, orientation-, annotation- and exploration-helpers.

6.1.1 Unity

The Unity game engine is used for rendering the scene and the navigation-helpers. The Unity engine was developed by Unity Technologies and is a cross-platform game engine for more than fifteen different platforms.

6.1.2 Oculus Rift SDK

For using Unity with the Oculus Rift, the Oculus Rift SDK is needed. It sets up the double cameras at users’ eyes and has some built-in functions for navigating with an XBOX-controller in a virtual environment.

6.1.3 Google Streetview API

With the Google Streetview API the 360° picture is gathered at certain chosen locations by running a small Java program that downloads a 640x640 picture (maximum size possible) for every rotation at that position.

6.1.4 Microsoft Image Composite Editor

After the images using the Google Streetview API are collected, they are stitched together using Image Composite Editor to create a 2560x1280 360°-image that can be loaded in the Unity scene.

6.1.5 Google Sketchup

Google Sketchup was used to create certain 3D-models (like the 3D-arrows) used for the visualizations.

6.1.6 Cardboard SDK for Android

The first prototype was creating with a Google Cardboard using the Cardboard SDK for Android with OpenGL ES. This required many lines of code for displaying simple objects that could very easily be displayed with the Unity game engine. Therefore later it was decided to switch to Unity with the Oculus Rift.

1 http://unity3d.com/
3 https://developers.google.com/maps/documentation/streetview/
5 http://www.sketchup.com/products/sketchup-make
6 https://developers.google.com/cardboard/android/
6.2 Implementation of The Google Cardboard Prototype

The first prototype was created for the Google Cardboard with OpenGL ES. Its final version, before the decision to switch to the Oculus Rift was made, was capable of looking around in a 360° image and rendering the floor-plan of a building based on its google maps coordinates and some manually defined georeferences (see Figure 6.2). The code was created starting from Google’s treasurehunt example for Google Cardboard.

![Google maps outline of a building](image)

Fig. 6.2: Google maps outline of a building drawn around the real building.

6.2.1 MainActivity

The MainActivity class contains all the used functions which will be explained one by one below. First I will show the variables important for creating the sphere, texture and lines. The sphere consists of three parts. Two small cones on top and below the sphere and the rest of the sphere. FloatBuffers need to be created for storing vertices, colours, normals, texture coordinates and for keeping track of the model (transformation matrix) of the objects so objects can be moved and rotated individually if needed.

```java
// Bottom cone vertices
private FloatBuffer mGroundVertices;
...
private float[] mModelGround;
```
The `onCreate()`-function is expanded by adding the transformation matrices that are used to render the scene. They are initialized in this function. The view will also be set to the CardboardView here. See Appendix B.1.2 for its implementation.

The next important function is `onSurfaceCreated()` (see Appendix B.1.6). It fills the buffers to store information about the 3D-world. First the two cones will be created. The functions that generate their vertices, normals and texture coordinates will be explained later on. This part generates said arrays and converts them into ByteBuffers because OpenGL does not use Java arrays but can only use these ByteBuffers. Next the sphere itself gets generated.

The last part of this function consists of creating filling the buffers for creating the map of the floor-plan based on coordinates from Google Maps. These coordinates also need to be converted first to x/y-coordinates or they will look warped. Lastly, the shaders will be created and the transformation matrices will be initialized (Appendix B.1.7).

Next, the function in Appendix B.1.1 creates the vertices for a circle (or cone if the y-value of the circle is higher or lower than the ones on its edge) for closing the bottom and top of the sphere.

Alongside vertices, the texture coordinates and normals need to be generated to tell the application what part of the image should end up where. It also makes sure the image will be put on the inside of the sphere (normals pointing inwards). See Appendix B.1.3.

For the creation of the sphere, a defined number of “cylinders” will be created with a different sized top and bottom circle and will be put on top of each other to form a sphere. The code for generating the vertices of a sphere can be found in Appendix B.1.4.

Next, the sphere needs texture coordinates and normals as well. Two functions analogue to the one for generating vertices were made for creating coordinates and normals. Its code is shown in Appendix B.1.5.

The code to transform the Google Maps coordinates to x/y-points was based on JavaScript-code from a Google Maps example[^1] and can be found in Appendix B.1.8. See Figure 6.3 for a visual explanation of what this function does. Basically, it prevents the map from looking warped when using coordinates straight from Google Maps.

The next function that is used is `SetNextTexture()`, which loads a texture and loads a set of coordinates for drawing a map (see Appendix B.1.11).

[^1]: https://developers.google.com/maps/documentation/javascript/examples/map-projection-simple
Then, setNewCenterPos() creates new vertices for the floor-plan if a new one should be created after loading a new 360°-image. It also transforms the coordinates of the floor-plan to fit with the current loaded 360°-image based on georeferences by calling the function remapSurroundMapWithGeoreferences(). See Appendix B.1.9.

The georeferences link a pixel-coordinate of the 360°-image to a Google Maps coordinate so the application knows where to move and rotate the map towards (see Figure 6.4). Then, remapSurroundMapWithGeoreferences() will use a number of georeferences to shift, rotate and scale the map using vectors so it fits around a real building. The first georeference will be used to shift the map to the real building. The second one will be used to rotate the map and scale it up or down. Any more georeferences will increase precision. See Figure 6.5 for a more visual explanation.

What happens in this function is the following:

1. Find georeferences stored for the current map.

2. Load coordinates for the current map, convert them to X/Y coordinates, and shift them around the origin (position of the user).

3. Use the first georeference and shift the corresponding map-vertex towards it by creating a vector between the new point and the old point. Shift all the other points of the map by applying the same vector shift to them.
4. Use a second georeference to find where a second point from the map needs to end up. Using the first georeference as a pivot, the angle will be calculated that the map will need to rotate as well as how much the map will need to be scaled up or down to fit.

5. After calculating the angle and scale, all other points of the map will be transformed using the same vector-transformation as the first one. See Appendix [B.1.12] for the full code.

This function calls remapCoordsAroundCenterPos(). This can change the origin of a map from the top-left corner to a new specified location and it will scale the map up or down. This function is used for generating the map from Google Maps coordinates before transforming it using georeferences (e.g. if as OpenGL size “10” is given, it will draw the map around the user in that size). Its code can be found in Appendix [B.1.13].

Another important function is getOpenGLPosFromTextureCoords(), which is needed to find where the map should end up. We know the pixel-coordinate of where a Google Maps-coordinate should end up but this will still need to be found in the 3D-space. This function will locate a pixel-coordinate on the sphere with the 360°-image texture on by virtually generating the sphere again and finding the nearest sphere vertex. See Appendix [B.1.10] for the implementation. Now all that is left, is to draw everything with OpenGL. How to do this, is shown in Appendix [B.1.14].
6.2.2 Summary

1. Declare the buffers used for storing vertices and models.

2. Initialize the buffers by generating vertices, normals and texture coordinates for the sphere and map.
   (a) Generate a cone on the top and the bottom of where the sphere will be.
   (b) Generate cylinders with different sized bottom and top planes stacked on top of each other to form a sphere.
   (c) Generate normals that point inwards.
   (d) Texture the sphere

3. Transform the Google Maps coordinates to coordinates that can be used by the application.

4. Shift, rotate and scale the map so it fits its georeferences.

5. Call the Draw-functions to draw the scene for each eye.
A small Java-program was created for downloading a series of pictures at a certain location using the Streetview API by giving coordinates as input. The pictures that are downloaded can be used as input for Image Composite Editor to stitch them together into a 360°-picture.

```java
public static void Download360ImagesAt(double lat, double lon) {
    String savePath = "D:\streetviewphotos\";
    URL website = null;
    try {
        // Street View images can be returned in any size up to 640 by 640 pixels.
        // Google Maps API for Work customers who are correctly signing their URLs
        // can request images up to 2048 by 2048 pixels
        int photoWidth = 640;
        int photoHeight = 640;

        // For each heading (looking left and right)
        for (int heading = 0; heading < 360; heading += 15) {
            // For each pitch (looking up and down)
            for (int pitch = -90; pitch <= 90; pitch += 15) {
                String streetviewUrl = "https://maps.googleapis.com/maps/api/streetview?size=
                        \"x\" + photoHeight + \"&location=\" + lat
                        + \",\" + lon + \"&heading=\" + heading + \"&pitch=\" +
                        pitch + \"&key=\" + APIKEY;
                website = new URL(streetviewUrl);
                ReadableByteChannel rbc =
                        Channels.newChannel(website.openStream());
                System.out.println(savePath + heading + "\" + pitch + 
                        \".jpg\"); FileoutputStream fos = new FileoutputStream(savePath +
                        \"heading\" + heading + \"pitch\" + pitch + 
                        \".jpg\");
                fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
                rbc.close();
                rbc = null;
                fos.close();
                fos = null;
            }
        }
    } catch (MalformedURLException e1) {
        e1.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("Done");
}
```
6.4 Unity Scene for the Oculus Rift Prototype

Unity-projects use scenes that are filled with gameobjects that can run scripts. This section will discuss how to implement the visualization components first. Their structure within the scene and how they are activated will be shown after. The objects with a script have their own class that is inherited from MonoBehaviour and use the Update()-function that is called every frame to change their appearance if needed. All components have their own class. Only the Update()-function and other relevant functions will be shown. Below is one example of a simple class.

```
using UnityEngine;
using System.Collections;

public class MinimapScript : MonoBehaviour {
    // The middle of the eyes of users and where they are looking at
    public GameObject centerEyeAnchor;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        RectTransform thisRectTransform = GetComponent<RectTransform>();
        Vector3 userRotation = centerEyeAnchor.transform.rotation.eulerAngles;
        Vector3 newRotation = transform.eulerAngles;
        newRotation.z = (userRotation.y);
        thisRectTransform.eulerAngles = newRotation;
    }
}
```

6.5 Implementation of AR-Helpers

AR-Helpers are the visualizations that will be 3D-objects around users and may not always be in their vision.
6.5.1 AR Navigation

Fog Path

The Fog Path is a particle emitter which renders transparent smoke-textures that fade in time and slightly change colour. The visualizations were created by placing these particle emitters at the right position and angle so users see them on the road. The properties for the particle emitter used in this prototype can be seen in Figure 6.6.

Object Path

The Object Path is much like the Fog Path except that the difference is that this a sequence of objects (diamonds in this case). In Unity, 3D-models of little diamonds were placed in the scene at locations so they look like they are floating above the road.

Fig. 6.6: Properties for the particle emitter for the Fog Path.

Fig. 6.7: Placing an object for the Object Path.
6.5. Implementation of AR-Helpers

**Highlighted Path**

The highlighted path consists of cubes that were transformed to have a path-like (rectangle) shape and given a different colour by changing their material to a semi-transparent orange one.

![Highlighted Path](image)

*Fig. 6.8: Placing an object for the Highlighted Path.*

**Moving Arrow**

The moving arrow is a 3D-arrow that moves towards a certain user-defined point. The arrow is a 3D-model created with Google Sketchup. It has a small script that makes it fly towards a user-defined point (endPos) and then hop back to its starting location. The step defines a distance it needs to fly between each frame, then it will be moved forward that distance on the vector it is looking at. If its destination is reached or got stuck, it will be put back at its start.

```csharp
1  void Update () {
2      // Distance travelled between frames
3      float step = speed * Time.deltaTime;
4      // Travelling said distance towards a user defined point
5      transform.position = Vector3.MoveTowards(transform.position, endPos, step);
6      // If endPos reached or managed to get stuck -> reset to start position
7      if (transform.position == endPos || transform.position == prevPos)
8          transform.position = startPos;
9      prevPos = transform.position;
10  }
```
Arrow Next To User

This 3D-arrow stays next to users (in front of their eyes) but points at their target. It uses two functions. One to put it in front of the user and the other to keep it pointed at the user’s target (dest).

```csharp
void InitArrow() {
    // Put the arrow in front of the user’s eyes
    Vector3 startPos = centerEyeAnchor.transform.position +
        3*centerEyeAnchor.transform.forward + new
        Vector3(0f,-2f,0f);
    transform.position = startPos;
    transform.parent = centerEyeAnchor.transform;
}

void Update() {
    // Keeps the object pointed towards dest
    transform.LookAt(dest);
}
```

This visualization also has a small activation-script with data relevant to the current active streetview image. Objects that are activated with the VisualizationController (Section 6.7.2) hold this script. These activation-scripts (Arrow Next To User is one of the visualizations that use these) all hold a reference to the object and update this with their data.

```csharp
public class InitArrowNextToPlayer : MonoBehaviour {
    public Vector3 dest = new Vector3(0f,0f,0f);
    public GameObject arrowNextToPlayer;

    void Start () {
        EnableArrow ();
    }

    void OnEnable () {
        EnableArrow ();
    }

    void OnDisable () {
        DisableArrow ();
    }

    public void EnableArrow () {
        ArrowNextToPlayer arrowScript =
            arrowNextToPlayer.GetComponent<ArrowNextToPlayer> ();
        arrowScript.setLookAtDest (dest);
        arrowNextToPlayer.SetActive(true);
    }

    public void DisableArrow() {
        if (arrowNextToPlayer != null)
            arrowNextToPlayer.SetActive(false);
    }
}
```
Static Arrows

This visualization is created by placing some 3D-models of arrows that were created with Google Sketchup on the road users need to follow.

![Fig. 6.9: Placing an object for the Static Arrows.](image)

6.5.2 AR Orientation

Heat Vision

Heat Vision uses a combination of enabling quads (object users need to find) with an orange texture (to make it stand out) and enabling a custom shader on the streetview-image which gives the colours a blue shade. It is mostly the same as the standard built-in UnlitTexture shader except that this one accepts other colours to blend in with textures by having a Color property. The a Color can be mixed with the colours of the texture in the fragment shader. The code for this shader can be found in Appendix B.2.1.

```
void ShowHeatVision() {
    if (!heatVisionOn)
        StreetView.GetComponent<Renderer>().material.SetColor("_Color", Color.blue);
    else
        StreetView.GetComponent<Renderer>().material.SetColor("_Color", Color.white);
    heatVisionOn = !heatVisionOn;
}
```

Having this shader will enable us to change colours of a texture. When it is white, everything will look like normal but putting in a different colour will add in that colour (blue in this case).
6. Implementation

6.5.3 AR Annotation

Object Names

Object names are floating text-objects. One white text with slightly shifted black text right behind it to make sure the text is always readable. The black text is a child gameobject of the white text so it rotates along with the white text. These text objects have a script to keep them pointed towards the user.

```
void Update () {
    transform.LookAt (centerEyeAnchor.transform.position);
    // Text looked away from user, so we rotate it 180 degrees
    // around its y-axis
    transform.Rotate (0f, 180f, 0f);
}
```

Object Logos

This visualization is more or less the same as Object Names except that textured quads (small planes) are used in stead of text elements.

Highlighting

Highlighting creates a coloured transparent plane that can be moved in front of the desired object to create a highlighting.

```
void Update () {
    transform.LookAt (centerEyeAnchor.transform.position);
    // Text looked away from user, so we rotate it 180 degrees
    // around its y-axis
    transform.Rotate (0f, 180f, 0f);
}
```

Fig. 6.10: Text mesh and script used with Object Names.

Fig. 6.11: Material used for a highlighting object.
6.5.4 AR Exploration

Mega Map

The Mega Map is a 3D-model created with Google Sketchup. Its meshes are given a name that can be displayed on the HUD of users. A line will be drawn from the mesh to the text on the HUD using a LineRenderer. The script contains a camera to raycast from so we know what the user is looking at. The layer mask is there so rays only hit objects from that specific layer. There are some variables for changing the font and the last one is a text object on the HUD that can be activated from the megamap-script.

Fig. 6.12: Components of a megamap mesh.

First a ray will be cast from the center of the camera towards the direction users are looking. It can only hit objects on a certain layer (Mega Map layer), so when we hit something, we can be certain it is a Mega Map object.

Next we find the middle of this object (by taking the average of all the positions of its vertices) because we want to draw a line from the object to the users’ HUD to make sure they know what they are looking at. Once we know the middle of the object, we create a LineRenderer and draw a line from the middle of the object to the text on a user’s HUD. The line and text (its name) get removed when users are no longer looking at an object. Its implementation is shown in Appendix B.2.3.
6. Implementation

6.6 Implementation of Widget-Helpers

Widget-helpers are mostly 2D-visualizations that will be on a user’s vision at all times.

6.6.1 Widget Navigation

Mini-map with marked path

The mini-map is a round texture displaying a map that rotates along with users. In this prototype a round texture of Google Maps around the current location was displayed.

The mini-map is a gameobject on a canvas (the HUD) that stays in front of a user’s vision. One must also make sure that the orientation between streetview-images remains the same (e.g. North should always be in the same direction) or the mini-map will no longer map with the “real world” unless the texture is manually rotated. The mini-map has a child-object with another transparent texture that will be layered on top of the mini-map.

![Fig. 6.13: Components of a minimap.](image)

This transparent texture contains the blue line on the road users need to follow. See Figure 6.13 for the components of the mini-map (the child-object for the blue line has a similar structure). Its script uses the CenterEyeAnchor supplied by the Oculus Rift SDK to know the direction users are looking at and rotating the map around with it. It also needs a small script for activating it.

```csharp
void Update () {
    RectTransform thisRectTransform = GetComponent<RectTransform> ();
    Vector3 userRotation = centerEyeAnchor.transform.rotation.eulerAngles;
    Vector3 newRotation = transform.eulerAngles;
    newRotation.z = (userRotation.y);
    thisRectTransform.eulerAngles = newRotation;
}
```

Objects that are activated with the VisualizationController (see Section 6.7.2) hold the following script. This script contains the texture the mini-map needs, the texture for the blue line (road that needs to be followed) on the mini-map and a reference to the canvases for the mini-map.
public class InitMiniMapNav : MonoBehaviour {

    public void EnableMiniMap() {
        minimap.SetActive(true);
        minimapnav.SetActive(true);
        // As long as there is at least one mini-map visualization active
        // The mini-map should be shown (the mini-map is on a different canvas than its helpers)
        visualizationController vC = GameObject.Find("visualizationController").GetComponent<visualizationController>();
        vC.increaseMinimapHelpers();
    }

    public void DisableMiniMap() {
        GameObject findVC = GameObject.Find("visualizationController");
        visualizationController vC = null;
        if (findVC != null) {
            vC = findVC.GetComponent<visualizationController>();
            vC.decreaseMinimapHelpers();
            if (vC.getNumMinimapHelpersActive() < 1)
                minimap.SetActive(false);
        }
        if (minimapnav != null)
            minimapnav.SetActive(false);
    }
}

2D-arrow on HUD

This is a little white arrow (Raw Image) in the top right corner of a user’s vision. It needs a point in 3D-space and uses the vector between the user and this point to rotate itself towards it.
6. Implementation

```csharp
void Update () {
    RectTransform thisRectTransform =
        twoDPointingArrow.GetComponent<RectTransform> ();
    Vector3 userRotation =
        centerEyeAnchor.transform.rotation.eulerAngles;

    // Vector of what the user is looking at
    Vector3 userLookAt = centerEyeAnchor.transform.forward;
    userLookAt.y = 0f;

    // Find the vector between the user and the destination
    Vector3 pointAtDirectionVector = new Vector3
        (pointAtDirection.x -
        centerEyeAnchor.transform.position.x, 0f,
        pointAtDirection.z - centerEyeAnchor.transform.position.z);
    pointAtDirectionVector.Normalize ();

    // Calculate the angle between what users are looking it
    // and what they should be looking at
    float rotationAngle = 360 - Vector3.Angle (userLookAt,
        pointAtDirectionVector);
    Vector3 cross = Vector3.Cross (userLookAt,
        pointAtDirectionVector);
    if (cross.y < 0)
        rotationAngle = -rotationAngle;
    Vector3 newRotation = new Vector3 (0f, 0f, rotationAngle);

    // localEulerAngles are needed here so it rotates around its
    // own axis on the canvas it is on. Regular eulerAngles
    // create unwanted results.
    thisRectTransform.localEulerAngles = newRotation;
}
```

6.6.2 Widget Orientation

Compass

The compass which remains in front of the user is actually a small cylinder that rotates along with a user’s rotation. Users only see the side of this cylinder so it more or less looks like a rectangle to them. Another transparent cylinder which only has a diamond-shape that is not transparent rotates on top of the first cylinder to create the dot users need to follow. The position of this dot on the first cylinder is given by a single value (starting rotation). Only the first cylinder has a script to rotate along with the user because the second one is a child-object of the first one and will therefore perform the same rotations.

![Fig. 6.14: Texture for the Compass.](image)
Another small initialization script is used for activating this visualization. It defines the textures that need to be used and at which location the target-diamond should be placed. It also sets the objects as enabled so they are no longer hidden.

```csharp
public class InitCompass : MonoBehaviour {
    public GameObject compass;
    public GameObject compassDiamond;
    public Texture compassTexture;
    public float diamondYRot = 0;

    // Use this for initialization
    void Start () {
        EnableCompass();
    }

    void OnEnable() {
        EnableCompass();
    }

    void OnDisable() {
        DisableCompass();
    }

    public void EnableCompass() {
        compass.transform.parent.gameObject.SetActive(true);
        Vector3 diamondEuler =
            compassDiamond.transform.localEulerAngles;
        diamondEuler.y = diamondYRot;
        compass.transform.localEulerAngles = diamondEuler;
        compass.GetComponent<Renderer>().material.SetTexture
            ("_MainTex", compassTexture);
    }

    public void DisableCompass() {
        if (compass != null)
            compass.transform.parent.gameObject.SetActive(false);
    }
}
```
Radar

The radar consists of a non-moving “background-image” and a rotating orange dot on top of it on a canvas-element. The script for rotating it is more or less the same as the one for rotating the 2D-arrow on HUD (see Appendix B.2.4 for the radar-implementation). The radar also needs an activation-script that keeps track of the current objective for a specific streetview-image and activates the radar with this data.

1 public class InitRadar : MonoBehaviour {
2    public GameObject radar;
3    public Vector3 endPoint;
4    // Use this for initialization
5    void Start () {
6        EnableRadar ();
7    }
8
9    void OnEnable () {
10       EnableRadar ();
11    }
12
13    void OnDisable () {
14       DisableRadar ();
15    }
16
17    public void EnableRadar () {
18       radar.transform.parent.gameObject.SetActive(true);
19       radar.GetComponent<RadarScript> ().pointAtDirection = endPoint;
20    }
21
22    public void DisableRadar () {
23       if (radar != null)
24          radar.transform.parent.gameObject.SetActive(false);
25    }
26}

6.6.3 Widget Annotation

Mini-map Annotations

Mini-map annotations are small Raw Images manually placed on top of the right buildings on a mini-map. They have a small script so they stay rotated in the same direction to prevent them from rotating upside-down when users are looking around.

1 void Update () {
2     transform.LookAt (transform.forward*0.5f);
3     transform.Rotate (0f, 180f, 0f);
4 }

An activation-script is needed again in the object that activates the mini-map to load the mini-map with the correct texture and annotations. This is similar to the one in Section 6.6.1.
6.7. Hierarchy of the Different Components

Mini-map Icons

This visualization is the same as the annotations except that the images now contain icons instead of text (see section 6.6.3). It also uses the same activation-script. Image 6.16 shows how icons (and annotations) were placed on the mini-map.

Fig. 6.16: Placing a mini-map icon on the mini-map.

6.6.4 Widget Exploration

Mini-map

This is just a regular mini-map without any further text or other helpers. It is there just so users get a sense of what is around them. It is the same as the mini-map in section 6.6.1 but this one does not have any other textures layered on top of it. The objects that toggle the mini-map have an activation-script that can be found in Appendix B.2.5.

6.7 Hierarchy of the Different Components

The 3D-part of the scene consists of a big sphere with inverted normals so we can texture the inside of it (see Figure 6.17) and the hierarchy lists the structure of the gameobjects (see Figure 6.18). The objects will be discussed in this section one by one.

Fig. 6.17: Appearance of the scene zoomed out.  
Fig. 6.18: Hierarchy of the scene.
6.7.1 Streetview

The streetview gameobject is the big sphere that contains the texture in which users look around. Placing the script in Appendix B.2.2 in [Path To Unity Project]/Assets/Editor allows you to easily put a sphere with inverted normals in your scene by creating a new option in the menu (see Figure 6.19).

![Image of placing a sphere with inverted normals in a scene.](image)

**Fig. 6.19:** Placing a sphere with inverted normals in a scene.

6.7.2 VisualizationController

This gameobject contains the script that orchestrates when which visualizations (helpers) should be shown and swaps out different versions of a visualization in case the scene changes to another location. Visualizations are shown by activating a gameobject that contains the corresponding data for the visualization of a particular streetview-position (e.g. Figure 6.20). In stead of just containing 3D-objects, these gameobjects can also be empty and contain a script with some data that activates the corresponding visualization with data for that position (activation-scripts).

---

6.7. Hierarchy of the Different Components

Fig. 6.20: List of mega-map objects for each created position/streetview image.

The visualizationController keeps track of which streetview image users are currently looking at and will enable/disable the corresponding gameobjects from a list of the active visualization. Activating one of the objects in this list, can just result in showing a 3D-object (like the Highlighted Path) or it can start a script to run (e.g. blending blue in the streetview-image for Heat Vision). This means that the visualization-controller needs to access the lists of helpers and has a reference to each visualization-category (Navigation, Orientation, Annotation and Exploration). It also has access to the HUD of users.

The scene can be changed by pulling the trigger on an XBOX-controller. Once the value of the triggers exceed 0.98 (pushed in all the way), a function will be called and a boolean will be set to true to prevent that this function will be called more than once (because Update() is called for each frame).
void Update () {
    if (canSwitchScene) {
        if ((OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.RightTrigger) >= 0.98f) && !rightTriggerPressed) {
            rightTriggerPressed = true;
            IncreasePos();
        }
        if (OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.RightTrigger) < 0.98f) {
            rightTriggerPressed = false;
        }
        if ((OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.LeftTrigger) >= 0.98f) && !leftTriggerPressed) {
            leftTriggerPressed = true;
            DecreasePos();
        }
    }
}

Which calls the following functions:

void IncreasePos() {
    if (currentPosition + 1 < maxPos) {
        ToggleCurrentHelpers();
        currentPosition++;
        SetupPos();
    }
}

void DecreasePos() {
    if (currentPosition - 1 >= 0) {
        ToggleCurrentHelpers();
        currentPosition--;
        SetupPos();
    }
}

ToggleCurrentHelpers() will activate or deactivate the current active visualizations and add it to a list to load up the new version of that visualization so it will fit the new streetview-image.
6.7. Hierarchy of the Different Components

```csharp
void ToggleCurrentHelpers() {
    List<NavigationVisualization> tempActiveHelpers = new List<NavigationVisualization>(activeHelpers);
    foreach (NavigationVisualization navV in tempActiveHelpers) {
        ShowNavigationHelper(navV.helperName, navV.navigationName);
    }
    activeHelpers = new List<NavigationVisualization>(tempActiveHelpers);
}
```

SetupPos() switches the streetview image and enables the visualizations again.

```csharp
void SetupPos() {
    Texture streetviewTexture = Resources.Load("Streetviews/streetviewPos" + (currentPosition + 1)) as Texture;
    StreetView.GetComponent<Renderer>().material.SetTexture("_MainTex", streetviewTexture);
    ToggleCurrentHelpers();
}
```

The last important function for VisualizationController is ShowNavigationHelper(). Based on a name and category, this function will look in the right category-list and enable the gameobject at the right position (like the ones in Figure 6.20) for the streetview that is currently active.

First there will be checked if the visualization is already active and will add or remove it from the list of active visualizations accordingly. Based on the name of the visualization, the correct list of visualizations will be used (Navigation, Orientation, Annotation or Exploration) from its references.

Then it will look for the right visualization within the current category and enable that visualization at the current position (the current streetview image). Enabling this object may just make the object appear from being hidden or it may run an activation-script. See Appendix B.2.6 for the implementation of ShowNavigationHelper().

6.7.3 OVRPlayerController

This gameobject and its children (see Figure 6.21) are supplied by the Oculus Rift for Unity SDK. These are objects that have some built-in functionalities (e.g. cameras for each eye).
It is in this gameobject that the HUD was placed because placing it there as a child will make it keep its local position and rotation in front of users. In addition, it contains three canvases. One for the minimap, one for the radar and one for the menu and 2D-arrow on HUD. The “3D-arrow next to user” and compass are also children to make them stay in front of the user.

A small menu was also created to easily activate and deactivate visualizations manually with an XBOX-controller while the application was running. Its implementation can be found in Appendix B.2.7.

### 6.7.4 The Visualization Categories

Every category of the visualizations is a gameobject that contains a list of corresponding visualizations that in turn have another list that contains the data of that visualization for each streetview-location (see Figure 6.22).

![Fig. 6.21: Objects in OVRPlayer-Controller.](image)

![Fig. 6.22: Data for a visualization for each position for AR-annotation.](image)
7. EVALUATION

The helpers discussed in Chapter 5 were tested by sixteen different users. They were given a scenario and a set of tasks to solve. This test was a way to figure out which of these helpers proves the most useful, the most visually appealing and which one users would use most. One most also however take in mind the fact that this test was done with an Oculus Rift DK2. It has a 1920x1080 screen that is divided in half for each eye. This means that some parts can appear blurry. It is also still possible to “see the pixels”.

7.1 Tasks

Users will be tasked with testing which navigation-helpers they like best for finding targets or their destination. Each helper will be tested. Before the test, users will be given a scenario and a task they have to perform. The helpers are divided into the categories: Navigation, Orientation, Annotation and Exploration. After performing the tasks of a category, users will be asked to fill in a survey (Appendix C) where they can evaluate the helpers they just used based on their subjective satisfaction. Users will be asked to stand up before starting. They will be asked to tell what they think they need to do while solving these problems. A picture will be shown below for each task to show which helper will be used (different locations will be used in a scrambled order so users do not know where to go based on their previous position).

“I will be evaluating helpers for Google Glass that are shown using a head-mounted display for helping users with navigation-tasks. There are four categories of tasks and after each category I will ask you to fill in a small survey asking about your subjective satisfaction of the helpers you just used. I also ask that you try to think out loud. So say what you think you should do. And finally, the system will be tested, not you. So do not worry about mistakes.”

7.1.1 Structure of the Tasks

There are 18 different tasks users need to complete. For each category of helpers (Navigation, Orientation, Annotation and Exploration), users will need to perform the corresponding tasks four times, each time with another amount of degrees they need to turn around to find their target. For each of the helpers they will get a task where they need to turn 45, 90, 120 or 180 degrees. (The tasks are spread in such a pattern that they do not realize when e.g. the 180 degree turn is coming up.)
7. Evaluation

7.1.2 Navigation Tasks

Users will be told they are equipped with a system that has a built-in GPS that will help them reach their destination. They are currently standing on the road and will be asked to point which direction they think they need to go based on the active helper at that time. Some scrambling of positions will be used so users do not know where to go next based on their previous position.

“Imagine you are walking to a new workplace you need to reach and that you currently stopped on the road to take a look where you need to go. You are currently wearing a head-mounted display that has built-in GPS capabilities to help reach your destination. You will be dropped on some locations where you will need to look in the direction you think you need to go based on a navigation-helper that will be activated. Look in the direction of the first decision point. This means that you should look at the crossroads if you think you need to take a turn there.”

Widget tasks

Task 1: Mini-map with marked path  Task 2: 2D-arrow on HUD

Augmented Reality tasks

Task 3: Fog Path  Task 4: Object Path
After the Navigation-tasks, users were asked to fill in the first part of the survey to rate the navigation-helpers (questions 1 and 2).
7.1.3 Orientation Tasks

For finding out which orientation-helper users prefer, they will be tasked with finding a person somewhere around them who may be inside a building.

“You know you are somewhere near the place where you will meet a friend and go get some ice-cream with him. You are currently looking for him but he could be inside a building or behind a wall. Based on the active helper, you will be tasked with looking at this person’s location or general direction even if you can not see him.”

Widget Tasks

Task 9: Compass

Task 10: Radar

Augmented Reality Tasks

Task 11: Heat Vision

After the Orientation-tasks, users were asked to fill in the second part of the survey (questions 3 and 4).
7.1.4 Annotation Tasks

While standing on the road, users will be told to look at a certain building. They will be shown a name first, followed by a logo.

**Widget Tasks**

“While walking around with no real destination you decide to take a look around because you just remembered you need to visit the building with the name or logo that will be shown to you. All you have to do is look at the corresponding building.”

Task 12: Mini-map annotations  
Task 13: Mini-map Icons/logos

**Augmented Reality Tasks**

Task 14: Object Names  
Task 15: Object Icons/Logos
After the Annotation-tasks, users were asked to fill in the third part of the survey (questions 5 and 6).
7.1.5 Exploration Tasks

This task will find out which map-helper is best for an exploratory approach. Users will be asked to tell which real highlighted building corresponds to which building on their map helper. For the widget-task, numbers will be put on the buildings on the mini-map for easier clarifying which building the users mean.

“While walking around with no real destination you stop for a moment to get a grip on your surroundings. You look at a building that you want to verify is on your map. The application will highlight this building. You have to tell which building this is on your map helper. For the mini-map tasks, you can just tell me the number of the building. Sometimes a building will be highlighted on the “mega-map” (this is a red 3D-map around you) and you need to look at the real building. This will become clear during the test.”

Widget Tasks

![Mini-map](image1)

Task 17: Mini-map

![Mega map](image2)

Task 18 + 19: Mega map

After the last set of tasks, the users were asked to fill in the rest of the survey.
7.2 Results

Two sets of data were gained from the user test. Task-performance data to see how fast and accurate users could perform the task, and subjective data from the survey.

7.2.1 Task Performance Results

![Error Rate](image1)

Fig. 7.1: The percentage of wrongfully executed Navigation-tasks (lower is better).

![Error Rate](image2)

Fig. 7.2: The percentage of wrongfully executed Orientation-tasks (lower is better).
7.2. Results

Fig. 7.3: The percentage of wrongfully executed Annotation-tasks (lower is better).

Fig. 7.4: The percentage of wrongfully executed Exploration-tasks (lower is better).
With this data we can conclude that the helpers where users made the least mistakes are: Static Arrows, Highlighting, Object Names, Heat Vision and Object Icons/Logos because only 1.6% of tasks were wrongfully executed. Something to note here is that all of these are AR helpers.

![Average Time Spent For Tasks](image)

Fig. 7.5: The average time users needed to complete a certain task. (lower is better)

The top five helpers with fastest completion time in order from fastest to slowest are: 3D-Arrow Next To User, Moving Arrow, Object Icons/Logos, Highlighting and Static/Arrows. These are once again all AR helpers.

For navigation (finding the road to a destination), the helper for the most reliably finding the road users need to follow is Static Arrows and the fastest is 3D-Arrow Next To User.

For orientation (finding a building or other object in a user’s surroundings), the helper for the most reliably finding the object users needed to find is Heat Vision and it is also the fastest.

Annotation helped users with finding a building of which they already knew the name or logo. The helpers that helped users the most reliable for these tasks are the three AR helpers for this category (Object Names, Object Icons/Logos, Highlighting) with the fastest one being Object Icons/Logos.

When walking around without a certain destination, users might want to take a glance at their map to see what is in their direct surroundings (where there are buildings or roads). For exploration we can conclude that it is easiest for a user to find a real building they see on their mega map but that it is hardest to find a real building back on this mega map. Both the amount of errors as well as the time they needed to complete the tasks showed this.
7.2. Results

7.2.2 Survey Results

After each category, users were given a survey to find out their subjective satisfaction of the helpers. The first questions they were asked were to rate all the helpers individually based on helpfulness, usability (how often they would use the helper) and visual appeal. Figure 7.6 shows the mean ratings for all different helpers.

Fig. 7.6: Results from asking users to rate the helpers based on helpfulness, usability (how much they would use it) and visual appeal.

**Individual Results: Navigation Helpers**

For the navigation helpers, the following ratings were given by users concerning the helpfulness of the helpers for completing the tasks. Users were asked to give the helpers a rating of zero to ten with zero being completely unhelpful and ten being very helpful. The following results are the average scores:

- Minimap with marked path: 6.5
- 2D-Arrow On HUD: 5.7
- Fog Path: 6.9
- Object Path: 7.1
- Highlighted Path: 7.8
- Moving Arrow: 7.7
- 3D-Arrow Next To User: 7.4
- **Static Arrows: 8.2**

The “winner” of being the most helpful according to users is the AR-helper: Static Arrows with a score of 8.2/10.
Next, users were asked to rate the helpers according their visual appeal. This yielded the following results:
- Minimap with marked path: 5.7
- 2D-Arrow On HUD: 5.2
- Fog Path: 6.3
- Object Path: 5.6
- Highlighted Path: 6.6
- **Moving Arrow: 7.6**
- 3D-Arrow Next To User: 7.5
- Static Arrows: 7.2

The most visually appealing navigation helper according to users is the “Moving Arrow” with a score of 7.6/10.

The last question users were asked for rating individual helpers was whether how much they would use this helper if they had access to a head-mounted display with the capabilities of using these helpers for a real navigation application.
- Minimap with marked path: 5.1
- 2D-Arrow On HUD: 4.7
- Fog Path: 5.8
- Object Path: 5.9
- Highlighted Path: 7.1
- Moving Arrow: 7.2
- 3D-Arrow Next To User: 7.3
- **Static Arrows: 7.6**

Looking at these results, we can see that the one users would use most is also the one they chose as the most helpful, being Static Arrows with a score of 7.6/10.

**Individual Results: Orientation Helpers**

For the orientation helpers, the following ratings were given by users concerning the helpfulness of the helpers for completing the tasks.
- Compass: 7.4
- Radar: 7.1
- **Heat Vision: 8.6**

The most helpful orientation helper is clearly the Heat Vision metaphor.

Next, users were asked to rate the helpers according their visual appeal. This yielded the following results:
- **Compass: 7.5**
- Radar: 6.4
- Heat Vision: 7.4

The most visually appealing orientation helper according to users is the Compass with a score of 7.5/10 very closely followed by Heat Vision with a score of 7.43.

The last question users were asked for rating individual helpers was whether how much they would use this helper if they had access to a head-mounted display with the capabilities of using these helpers for a real navigation application.
• Compass: 7
• Radar: 6.3
• **Heat Vision: 7.5**

The most helpful helper is also again the one that would be used most. Heat Vision has the highest score (7.5/10). Users noted however that they would not like it if this helper would be active all the time (because of it turns everything except for the target blue).

**Individual Results: Annotation Helpers**

For the annotation helpers, the following ratings were given by users concerning the helpfulness of the helpers for completing the tasks.

• Mini-map annotations: 4.4
• Mini-map Icons/Logos: 5.2
• Object Names: 7.6
• Object Icons/Logos: 8.1
• **Highlighting: 8.3**

For annotating objects around them, users seem to find only highlighting an object they are looking for as the most useful with a score of 8.3/10. If they would not be looking for something, we can conclude that the “Object Icons/Logos” is the best helper with a score of 8.1/10.

Users then rated the helpers again for visual appeal which produced the following results:

• Mini-map annotations: 3.9
• Mini-map Icons/Logos: 5.6
• Object Names: 5.9
• **Object Icons/Logos: 7.9**
• Highlighting: 6.6

“Object Icons/Logos” is the most visually appealing helper according to users.

Lastly users were asked to rank the helpers according to if they would use them if they would have a head-mounted display that could display these helpers in a real situation.

• Mini-map annotations: 3
• Mini-map Icons/Logos: 3.9
• Object Names: 6.3
• **Object Icons/Logos: 7.6**
• Highlighting: 7.4

The helper that would be used most is the “Object Icons/Logos” helper.
**Individual Results: Exploration Helpers**

For the exploration helpers, the following ratings were given by users concerning the helpfulness of the helpers for completing the tasks.

- **Mini-map: 6.1**
- **Mega Map: 6.6**

In an exploratory context, users would prefer the Mega Map over a mini-map. It received an average score of 6.6/10.

Next, users were asked to rate the helpers according to their visual appeal. This yielded the following results:

- **Mini-map: 5.625**
- **Mega Map: 7.6**

The most visually appealing exploration helper according to users is the Mega Map with a score of 7.6/10.

The last question users were asked for rating individual helpers was whether how much they would use this helper if they had access to a head-mounted display with the capabilities of using these helpers for a real navigation application.

- **Mini-map: 5.7**
- **Mega Map: 6.2**

Users would prefer to use the Mega Map over the mini-map. The Mega Map has a score of 6.2/10.

**Individual Results: All Helpers**

When ignoring specific situations and looking at the ratings of all the helpers, we can compare all the helpers to each other and determine which are the most helpful, visually appealing and which one would be used most.

The helper with highest helpfulness-score is “Heat Vision” with a score of 8.6/10.

The helper with highest visual appeal is “Object Icons/Logos” with a score of 7.9/10.

The helper which would be used most is “Static Arrows” with a score of 7.6/10.

All three are AR helpers. One of them is an Orientation-helper (Heat Vision), one is an Annotation-helper (Object Icons/Logos) and Static Arrows is a Navigation-helper.

**Favourite and Least Liked Helpers per Category**

For each category (Navigation, Orientation, Annotation and Exploration), users were asked to tell their favourite helper and their least favourite.

For Widget-navigation “Mini-map with marked path” was preferred over “2D-arrow on HUD”. AR-navigation’s most preferred helper was “3D-arrow next to user” and the least favourite one was “Fog path”.
Widget-orientation had the compass preferred over the radar. Heat Vision (AR-orientation) was generally liked and thirteen of the sixteen users marked that they liked this helper.

Mini-map Icons/Logos was preferred over Mini-map annotations for Widget-annotation. Object Icons/Logos was liked best for AR-annotation and “Object Names” received the lowest score.

Last, for exploration the mini-map was not liked. Ten people liked the mega-map while three did not like it (the other three had no opinion).

**Favourite Category**

Users were asked which category of helpers they think is “best”. Navigation was chosen as the best one with nine votes. Exploration received five and Annotation received the last two. Orientation did not receive any votes.

**Favourite Helpers in General**

![Chart showing ratings for favourite helper]

*Fig. 7.7: Results from asking users to vote for their favourite helpers.*

The second to last question users were asked to fill in, was to tell which three helpers were their favourite. The top three helpers that were marked as “favourite” are the following:

1. Mega Map (8 votes)
2. Object Icons/Logos (6 votes)
3. Heat Vision (5 votes)
Results for: Build Your Own HMD Navigation App

For the last question on the survey, users were asked to choose up to four widget-helpers and put them in one of the corners of the screen of their choice except for the compass which could only be placed in the top or bottom part of the screen (because of its width). Another exception was the mini-map where they were told that they could only have one mini-map but they could “layer” the helpers onto one map. They were also asked to choose up to three AR-helpers that would be active. This is the top five of the most chosen helpers:

1. Mini-map with marked path (10)
2. Mini-map Icons/Logos (7)
3. Compass (5)
4. Object Icons/Logos (5)
5. 2D-arrow on HUD and Mega Map (4)

One should also note that only one user picked a new location for a helper (the mini-map was put in the top-left corner). The reason why two mini-map helpers are at the top could be because users seemed to like the mini-map more when they could layer these two helpers. A mini-map with both a marked path and the Icons/Logos appeared to be a popular choice. The familiarity with a mini-map could also be a factor.

7.3 Discussion

From the user test results we can see that in general AR-helpers are superior in all aspects (error rate, completion time, visual appeal, helpfulness and how much of the time users would have them on) to widgets when users are constricted to only rating helpers for a certain category. This suggests that the ideal optical head mounted display application drops the use of Widgets and focusses only on using the best looking and most helpful AR-helpers. The most reliable application using only AR-helpers for every task uses “Static Arrows” for Navigation, “Heat Vision” for Orientation, “Object Names (or logos)” for Annotation and the “Mega Map” for Exploration.

However when users are given the opportunity to build their own setup, Widgets are preferred again over AR (Section 7.2.2) despite the fact that they were allowed to not use any at all. The application built by users uses a “Mini-map with marked path” for navigation (Widget), a “Compass” for Orientation (Widget), “Mini-map Icons/Logos” for Annotation (Widget) and the “Mega Map” for Exploration (AR).

This means that individually the Widget-helpers “lose” to the AR-helpers on their own but become more valuable when supported by AR-helpers. A certain balance between AR-helpers and Widget-helpers need to be found for the “ideal optical head mounted display navigation app”.
8. CONCLUSION

Because the number of OWG is growing rapidly and is becoming more popular, it is a good idea to take a look at their navigation-helpers as they have the same navigation problems as the real world. If these games have special kinds of visualizations for navigation-tasks for the same problems, why would they not work in real world situations?

From the two studies in this thesis, one can conclude that for real-world applications, Navigation seems to be the most important part. While OWG will favour Orientation. When we looked at research, we also saw that these visualizations all looked more or less the same in contrary to the many visualizations used in OWG. Something else we saw, when we compared the two studies was that Annotation is an important helper for providing extra information next to the main task.

Two prototypes were created of which one looked at these visualizations used in OWG and put them in a simulation that was then tested by 16 different users. These test offered some interesting results as AR-helpers seem to be the best (the least errors were made when using AR as well as the least amount of time needed to complete tasks). However, when users were offered the ability to create their own application, Widgets gained more popularity again. One can assume that Widgets become much more valuable when they can be backed up by AR and that users are more familiar with Widgets like a mini-map.

One must also note that the Oculus Rift DK2 was used for the second prototype. Which means that the Oculus Rift is still in development at the time of writing and still needed to be improved upon. An example of this is that the screen needs a bigger resolution so the raster of pixels is less visible than it is now.

The next steps for finding out what the best OHMD navigation application is, is finding out which helpers are the “best” and creating an application that can work in real-time while moving.
8. Conclusion
9. FUTURE WORK

9.1 Creating Seamless Transitions for the Oculus Rift Prototype

Agost Biro created “Perspective Scaling” \(^1\) (see Figure 9.1). The application uses “immersive panoramas” and applies transformations to them using their depth maps. This way frames between different panoramas can be generated.

![Perspective Scaling](image)

**Fig. 9.1: Perspective Scaling.**

Using the depth map from Google Streetview with the help of the Google Street View depth library GSVPanoDepth\(^2\), this could be applied to the simulation used in this thesis for creating a more visually appealing transition. Street Cloud (see Figure 9.2) is a demo created by Callum Prentice which demonstrates GSVPanoDepth.

---

\(^1\) [http://www.agostbiro.com/demo.html](http://www.agostbiro.com/demo.html)  
\(^2\) [https://github.com/proog128/GSVPanoDepth.js/tree/master](https://github.com/proog128/GSVPanoDepth.js/tree/master)
9. Future Work

9.2 Real-Time Application for an Optical Head Mounted Display

The prototypes used in this thesis were simulations of real situations. A real application needs to be able to use an OHMD like Google Glass, whereas in this thesis the simulation was created using an Oculus Rift (which does not even have a camera for creating a video see-through display). Next, the OHMD needs to show comparable navigation-helpers to the ones in this thesis on top the real world that can be seen through the see-through display and keep them updated and aligned with the real world while a user is moving in contrast to the Oculus Rift prototype where users were constrained to standing in one location to look around in. In addition, the application needs to be able to accept user input of course (e.g. in case a user wants to reach a certain destination, which would enable one or more navigation-helpers).

The best helpers will need to be determined for use in this application. Finally, users need to be given an option to enable and disable certain helpers so they can choose the ones they like best.
Appendices
Appendix A

STUDY OF 100 OPEN WORLD GAMES

In this section, I will discuss the “Augmented Reality”- and widget-functionalities in the “Top 100 Open-World games” by Gamingbolt.

A.1 Discussed games

For each game, the Augmented Reality (AR)-helpers will be listed along with non-augmented helpers (widgets).

A.1.1 Simpsons Road Rage

Fig. A.1: https://www.youtube.com/watch?v=H73zhzv8Jps at 13:23

Release date  4/12/2001
In this game, players need to drive to a certain point. They are helped by a mini-map and a floating hand that is pointing in the direction they need to drive. This game barely uses any AR-helpers. The only added element to the “real world” is a light that shows where the goal is. All other needed information (time left, money, objectives) is displayed on the screen.

1 http://gamingbolt.com/100-greatest-open-world-games
AR-helpers

- a light that shows where the goal is

Widget-Helpers

- Mini-map

- Floating hand that points in the right direction

- Plain text as information in the corner of the screen

A.1.2 Assassin’s Creed: Liberation

Fig. A.2: https://www.youtube.com/watch?v=L3SYS1rVUWy at 11:55
A.1. Discussed games

Fig. A.3: [http://guides.gamepressure.com/assassinscreedliberationhd/gfx/word/429027865.png](http://guides.gamepressure.com/assassinscreedliberationhd/gfx/word/429027865.png)

**Release date**  30/10/2012

Assassin’s Creed: Liberation helps the player navigate in a lot of ways. There is a mini-map in the corner of the screen. Targets can be marked by objects floating above their head and telling how far away they are. The currently selected non-playable character (NPC) will be highlighted with a white glow (if the player would fire a gun for example, it would hit this NPC). Objectives are shown as plain text on the screen next to a health bar. In another corner of the screen, the currently selected weapons are shown. Targets are shown as red icons that stick near the edge of the screen to show their direction.

And as in all games in the Assassin’s Creed-series, the player can also activate “Eagle vision” which turns everything in a shade of blue but highlights certain objects of interest so the player can easily differentiate between useless and useful objects (see Figure A.3).

**AR helpers**

- Annotations and labels floating above NPC’s
- Highlighted NPC’s
- Eagle vision

**Widget-helps**

- Mini-map
- Plain text and icons as information in the corner of the screen
A.1.3 Postal 3

Fig. A.4: [https://www.youtube.com/watch?v=Wyak8mBbm_g](https://www.youtube.com/watch?v=Wyak8mBbm_g) at 11:38

**Release date** 23/11/2011

The only information shown to the player is the currently selected weapon and the weapon wheel (list of available weapons) when the player wants to change his/her weapon. There are no augmented-reality helpers.

**AR-helpers** None

**Widget-Helpers**

- Plain text and icons as information in the corner of the screen
A.1.4 Deadly Premonition

Fig. A.5: [https://www.youtube.com/watch?v=x59NHdX0PfQ](https://www.youtube.com/watch?v=x59NHdX0PfQ) at 12:14

**Release date** 23/02/2010
Deadly Premonition only uses meters, bars and text shown directly on the screen to inform the player of things like health, currently selected weapon, ...

**AR-helpers** None

**Widget-Helpers**

- Plain text and icons as information in the corner of the screen
A.1.5  Dragon Age 2

Fig. A.6: [https://www.youtube.com/watch?v=o9Ez99sSkNg](https://www.youtube.com/watch?v=o9Ez99sSkNg) at 2:10

**Release date**  8/03/2011

The game features a mini-map, a list of members in the player’s party along with some basic information bars (health, mana, ...). In the bottom-right corner some skills that the player can activate are shown. Damaged enemies have a little arrow above their head along with a health-bar.

**AR-helpers**

- Health-bar floating above enemies

**Widget-Helpers**

- Plain text and icons as information in the corner of the screen
- Mini-map
A.1.6 Dead Island

Fig. A.7: [https://www.youtube.com/watch?v=EDkXFCrG_d4](https://www.youtube.com/watch?v=EDkXFCrG_d4) at 6:14

*Release date  6/09/2011*

The game features a mini-map and basic information bars/blocks (health, xp and equipped weapon). It also shows the health of enemies above their head and damage splashes. When picking up items, the name of the items appear.

**AR-helpers**

- Health-bar floating above enemies
- Name of items on top of items

**Widget-Helpers**

- Plain text, icons and bars as information in the corner of the screen
- Mini-map
A.1.7 DayZ

Release date 16/12/2013
DayZ barely uses any helpers at all. It only shows text about your health-state (thirsty) and about interactable objects (“pick up X”). There is also a weapon wheel.

AR-helpers

- Name of items on top of items when looking at them from close-by

Widget-Helpers

- Plain texts as information in the corner of the screen
A.1.8 Mafia 2

Fig. A.9: https://www.youtube.com/watch?v=tt1TTh0wp8 at 6:17

**Release date**  24/08/2010

Mafia 2 shows information about the currently selected weapon and when to reload it. There is a mini-map that shows the enemies and the objective. The objective is also in the corner of the screen as plain text. The crosshair also turns Red or Green when it is pointed at enemies or friendlies respectively.

**AR-helpers**  None

**Widget-Helpers**

- Plain text, icons and bars as information in the corner of the screen
A.1.9  Don’t Starve

Fig. A.10: https://www.youtube.com/watch?v=Bz3aM0Pd8xE at 2:40

Release date  23/04/2013
Don’t starve uses some information boxes that tell the player about the character’s health, what time it is and how hungry their character is. Also shown are the inventory and a list of objects the player can build. When the player hovers the mouse over objects, they get highlighted and a small text shows what the player can do with it.

AR-helpers

- Name of items on top of items when hovering over them
- Objects get highlighted when hovering over them

Widget-Helpers

- The objective and other information (weapons) as plain text
- Colour-changing cross-hair
- minimap
A.1. Discusssed games

A.1.10 The Elder Scrolls: Daggerfall

Fig. A.11: [https://www.youtube.com/watch?v=9VH3iuuiyuo](https://www.youtube.com/watch?v=9VH3iuuiyuo) at 9:39

**Release date** 31/08/1996

The Elder Scrolls: Daggerfall is an old (1996) game that does not have any AR-helpers. The only extra information the player gets on the screen is his/her health, stamina and mana. The other are buttons for opening menus along with a compass.

**AR-helpers** None

**Widget-Helpers**

- Menu buttons
- Compass
A.1.11 Final Fantasy XII

Release date 13/06/2006
There’s a diamond like object that points towards enemies. An arc that points from the playable character to the enemy shows that that character is going to attack and is within reach. The game also has a mini-map in the top right corner and shows statistics like hitpoints (name, HP and MP) in text. The player and NPC’s have their health bars floating above their head during combat.

AR-helpers

- Diamond pointing towards enemy
- Light-arc that signals an attack
- Health bar floating above character’s heads

Widget-Helpers

- Mini-map
- Information box with HP, MP and attack options
A.1.12 Mercenaries: Playground of Destruction

Fig. A.13: [https://www.youtube.com/watch?v=K_WrKSdnIsM](https://www.youtube.com/watch?v=K_WrKSdnIsM) at 15:01

**Release date** 11/01/2005

This game can display small information icons above enemies/objectives. The cross-hair changes to red when pointed at an enemy. There’s a head’s up display (HUD) that shows information (health, bullets, selected weapon, ...) and has a mini-map. It also sometimes shows positions where the player has to move to by making a circle light up at that spot. (e.g. [https://www.youtube.com/watch?v=K_WrKSdnIsM](https://www.youtube.com/watch?v=K_WrKSdnIsM) at 4:31) When shot, a 2D red icon on the screen will point towards the direction of the enemy that fired the shot (even if that enemy is not in sight).

**AR-helpers**

- Information icons/text floating above enemies’ heads
- light-circle that shows positions to move towards

**Widget-Helpers**

- Mini-map
- Information box with health, selected weapon, ...
- Colour-changing cross-hair
- 2D-icon that points towards enemies
Appendix A. Study of 100 Open World Games

A.1.13 Freelancer

Fig. A.14: [https://www.youtube.com/watch?v=NODc1lXn0ak](https://www.youtube.com/watch?v=NODc1lXn0ak) at 5:52

Release date 4/03/2003
Freelancer shows a lot of information through text (objectives, ...) and bars (health, ...) on the HUD. A red box gets drawn around “locked” enemies. The cross-hair changes to red when hitting an enemy target. Little arrows move around the screen to point towards enemies/way-points. Some objects also get their name as text on top of them along with the distance from the player.

AR-helpers

- Information icons/text floating above objects
- Boxes around objects (locked target, way-point, ...)

Widget-Helpers

- Compass
- Information boxes with health, power, ...
- Colour-changing cross-hair
- 2D-icon that points towards objects
A.1.14 Harvest moon

Release date  5/02/1999
Harvest Moon does not use any helpers. All actions are mapped onto buttons of the controller or need to be triggered by opening menus. The only on-screen information are message-text or little icon above an animal/person indicating that they noticed you tried to talk to them but do not have anything to say.

AR-helpers  None

Widget-Helpers

• Message text
A.1.15 True Crime Streets of New York

Release date 15/11/2005
Enemies have a health-bar floating above them. The HUD only contains of a box showing the currently equipped weapons and how many bullets are left, and the player health.

ARhelpers

- Health-bars floating above NPC’s

Widget-Helpers

- Plain text (messages, tutorials)

- Information boxes (Weapon and health)
A.1.6 The Simpsons Hit and Run

Fig. A.17: [https://www.youtube.com/watch?v=1eVIVqXqfLg](https://www.youtube.com/watch?v=1eVIVqXqfLg) at 14:26

**Release date** 16/09/2003

Simpsons Hit and Run can show immobile and moving objectives by drawing a light-circle around them and/or having a big floating icon above it. The game uses a mini-map and can show text-information about objectives on the screen. The game also draws arrows on the road so the player knows where to go. At some places there will be coins in a line on the ground with an object of interest at the end of it, to lead the player towards it.

**AR-helpers**

- Icons floating above objects
- Glowing circle on top of objectives
- Arrows that show the way on the road
- Line of coins

**Widget-Helpers**

- Text-information about objectives
- Mini-map
A.1.17 Midnight club Los Angeles

Fig. A.18: https://www.youtube.com/watch?v=Xz398rxyiX at 3:13

Release date 20/01/2008
This game has a mini-map with a compass and an arrow around it that points in the direction of the current way-point. Way-points in the real world are coloured smoke. There is also extra information shown on the HUD like speed, lap, time, ...

AR-helpers

- Way-points with coloured smoke

Widget-Helpers

- Text-information with speed, position, time, ...
- Mini-map
- Compass
A.1. Discussed games

A.1.18 Midtown Madness 3

Fig. A.19: https://www.youtube.com/watch?v=wmIkwnJ7ycE at 5:04

Release date 17/06/2003
Midtown madness has a mini-map with built in compass. There’s also a speedometer on the screen along with text about the other players and the elapsed time. Floating above the car is a 3D-arrow that points towards the current goal. Goals have a green glowing “dome” around them to show that this is the position the player needs to get to. Other players’ names also float above the corresponding cars.

AR-helpers

- Way-points with a glowing “dome” around

- 3D-arrow floating above car

Widget-Helpers

- Text-information with speed, position, time, ...

- Mini-map

- Compass
A.1.19 Gun

Release date 5/11/2005
This game is a bit of generic western shoot 'em up. It does not use any Augmented Reality. It does use 2D-elements on the HUD to help out the player. There's little mini-map, it shows information about the equipped weapon, bullets and health, the cross-hair changes colour depending on what the player is aiming at (white: nothing, red: enemy, blue: other NPC), and are little white bars that show the direction of incoming bullets.

AR-helpers None.

Widget-helpers
- Icons and bars that tell information about health, bullets, ...
- Mini-map
- White bars that point in the direction of incoming bullets
- Colour-changing cross-hair
A.1. Discussed games

A.1.20 Prototype 2

Fig. A.21: https://www.youtube.com/watch?v=z32IMR5XckE at 8:19

Release date  5/11/2005
The HUD shows the player’s health along with some icons about equipped skills and a mini-map with built in compass. The objectives or other information are shown as text. Enemies and objects of interest get an icon floating above them. When they move out the screen, their icon still stays on the edge of the screen in the direction the player needs to look to get these objects back in vision. Some bigger enemies also get a health bar floating above them. When the player picks up something to throw a “lock-on” box appears on enemies. Way-points have a glow around them.

AR-helpers

- Icons/health bars floating above objects
- Glowing spots to indicate there is a way-point there
- “lock-on” boxes around objects

Widget-Helpers

- Icons, bars and text that tell information about health, skills, objectives, ...
- Mini-map with compass
- Icons near the edge of the screen to indicate there is something there if the player looks in that direction
A.1.21 Assassin’s Creed: revelations

Release date 15/11/2011
For this game I will refer back to section A.1.2 as for AR-mechanics these games are the same apart from the icons looking different. One thing that could be noted is a small tutorial in the beginning of the game where the player needs to follow a glowing “augmented” assassin that shows the player the way.

**AR-helpers**

- Augmented reality character that shows the way
- See section A.1.2

**Widget-Helpers**

- See section A.1.2
A.1.22 Terraria

Release date 16/05/2011
Terraria is a 2D-game and needs less pathfinding methods because of the lack of a third dimension. It has a mini-map and a lot of icons for entering menus and using items.

AR-helpers None

Widget-Helpers

- Icons, bars and text that tell information about health, skills, items, ...
- Mini-map with compass
Appendix A. Study of 100 Open World Games

A.1.23 Fable 3

Fig. A.24: [Link to YouTube video](https://www.youtube.com/watch?v=uO9LEGvdzvI) at 6:58

**Release date** 26/10/2010

Fable 3 does not have a mini-map or a lot of information on the screen but does use icons and information that float on top of objects. This information can change to things that object is about to do (e.g., “Dig spot” indicating that the dog has found a spot that can be dug). The player is helped with finding the way by having a line of glowing little stars on the ground.

**AR-helpers**

- Line of little stars showing the way
- Icons and text floating above characters to indicate what they are/do

**Widget-Helpers**

- Icons indicating the currently equipped weapon
A.1.24 Bully: scholarship edition

Release date 17/10/2006
This game only shows a few essential information boxes on the screen like a health bar and a mini-map with a compass. The health of “enemies” is shown as a circle around their feet. Objective locations have a glow coming out of them.

AR-helps

- Health around enemies’ feet
- Light emitting objectives

Widget-Helpers

- Health bar
- Compass with a mini-map
A.1.25  Mercenary

Release date  1985
This is another very old game that’s too old to have any advanced techniques. It has something that more or less resembles a compass along with an altitude meter and some other icons.

AR helpers  None

Widget-Helpers

- Health bar
- Altitude meter
- Speedometer
- Sort of compass
A.1.26 Crackdown

Fig. A.27: https://www.youtube.com/watch?v=-o9_u1hXRlQ at 5:40

Release date 20/02/2007
Crackdown has a general HUD with information about health, objectives (text), a mini-map and icons for showing the current equipped weapon and how much ammo it has left. In this game some NPC’s get health bars or icons floating above their head indicating what sort of NPC they are. Way-points have a big beam of light coming out of them. When standing near a car, to signify the player he/she can enter it, the car will be highlighted with a green hue. The cross-hair will turn red when aiming at enemies and enemies can get a lock-on box around them.

AR-helpers

- Icons and health bars floating above NPC’s
- Way-points are made visible by having a beam of light come out of them
- Cars are highlighted when standing near them to signify they can be interacted with
- Lock-on boxes around enemies

Widget-Helpers

- Mini-map
- Icons for showing current equipped weapon, ...
- Plain text to give information about objectives
- Colour-changing cross-hair
A.1.27 Mercenaries 2

Fig. A.28: https://www.youtube.com/watch?v=A1-T1EqTdDg at 1:51

Release date 31/08/2008
Mercenaries 2 has a mini-map with built in compass on the HUD. The HUD also shows the player’s health and ammo stock. Messages, objectives and earned money are shown in plain text. Important objects and enemies get an icon floating on or above them indicating what they are. These icons stay on the screen for as long as the object could be in sight (e.g. if the object is behind another one, the icon is still shown). There are also red bars to show from which direction the player is being hit. The cross-hair also changes colour when hitting an enemy.

AR-helpers

- Icons floating above important objects or objectives to indicate what they are (ammo, enemies, ...)

Widget-Helpers

- Mini-map with built in compass
- Icons for showing health
- Plain text to give information about objectives
- Colour-changing cross-hair
- Red arrow to show direction of incoming bullets
A.1.28 Prototype

Fig. A.29: https://www.youtube.com/watch?v=Hx2TcT_FX8g at 10:22

Release date 9/06/2009
For this game I will refer to its successor in Section A.1.20 as this game uses the same helpers or less.

AR-helpers  See Section A.1.20

Widget-Helpers  See Section A.1.20
A.1.29  Fallout: New Vegas

*Fig. A.30: [https://www.youtube.com/watch?v=78Zq5CtQj8s](https://www.youtube.com/watch?v=78Zq5CtQj8s) at 6:06*

**Release date**  19/10/2010
Fallout: new vegas does not use a lot of extra helpers except from a HP- and AP-bar and compass. When aiming at an enemy however, different body-parts get an annotation about hit-percentages and when selecting one, it gets highlighted. When selecting an object, the cross-hair changes form.

**AR-helpers**

- Annotating body-parts of enemies with hit-percentages
- Highlighting body-parts of enemies

**Widget-Helpers**

- HP- and AP-bar
- Compass
- Cross-hair changes form when you can select something
A.1.30  Kingdoms of Amalur: Reckoning

![Image](https://www.youtube.com/watch?v=EEdc3NWq2io) at 1:30

**Release date**  7/02/2012

In Kingdoms of Amalur: Reckoning enemies have a floating health bar along with their name. On the HUD, we can see a few bars (health, mana, energy) and a mini-map with a compass. Loot containers are highlighted and loot itself has a smoke/glow coming out of it. When using a bow, the targeted enemy gets a small icon on it.

**AR helpers**

- Floating health bars and names
- Highlighted loot-containers
- Glowing loot
- Targeted enemies get a mark floating on them

**Widget-Helpers**

- health-bar, mana, ...
- Compass
- Mini-map
- Plain text for showing area-names or mission related text
A.1.31 Saints Row

Release date  29/08/2006

For AR-helpers, this game mainly uses floating icons above enemies, objectives, ... Way-points or locations with a functionality get a glowing circle on top of them. The game has a mini-map and a health bar along with the currently selected weapon. When aiming at enemies, the cross-hair changes colour.

AR-helpers

- Floating icons above objects to indicate their meaning
- Glowing circle around way-points

Widget-Helpers

- health-bar
- Mini-map
- Plain text
A.1.32 Arma 2

Fig. A.33: https://www.youtube.com/watch?v=Q35x7vR5RPs at 4:22

Release date  17/06/2009
Arma 2 is a game that aims to be a realistic fps-shooter game. Because of this it barely has any helpers. The only time the player gets a bit of extra information on the regular screen (not when looking at a map or something) is when he/she is looking at another player. The game shows a bit of text next to that player (e.g. “squad leader”).

AR-helpers

- Floating text next to very nearby other characters

Widget-Helpers

- Plain text
A.1.33 The Elder Scrolls: Arena

Fig. A.34: https://www.youtube.com/watch?v=ydOdkc6ETY8 at 4:12

Release date 1994
The oldest game of The Elder Scrolls series. Like in section A.1.10 there is not much to discuss. There are only a few information bars, a compass and some buttons on the screen.

AR-helpers None

Widget-Helpers

- Health-, mana- and stamina bar
- Compass
A.1.34 Burnout: Paradise

Fig. A.35: [https://www.youtube.com/watch?v=Cs0Gq1f8jsY](https://www.youtube.com/watch?v=Cs0Gq1f8jsY) at 3:08

**Release date**  21/01/2008
Burnout Paradise only uses its HUD for giving the player extra information. The only exception is that during multiplayer, the player’s names hover above their vehicles. The HUD has a mini-map, a compass and shows information about the current race in text.

**AR-helpers**

- Floating player-names

**Widget-Helpers**

- Mini-map
- Compass
- Mission-text (position, information, ...
A.1.35  Assassin’s Creed 3

Fig. A.36: https://www.youtube.com/watch?v=_s9aVCfPdng at 0:19

Release date  30/10/2012
For this game I will again refer back to section A.1.2 as for AR-mechanics these games are the same apart from the icons looking different.

AR-helpers

• See section A.1.2

Widget-Helpers

• See section A.1.2
A.1.36 STALKER: Shadow of Chernobyl

Release date 20/03/2007

In this game there’s a mini-map on the HUD and information about the player’s stance, armour and ammo. While pointing the cross-hair at a nearby item or shooting something, its name appears on top of it. The cross-hair will change colour when shooting an enemy and a 2D-arrow will be displayed on the screen when the player gets hit and points in the direction of the enemy.

AR-helpers

- Names on objects

Widget- Helpers

- Mini-map
- Health/armeour-bars
- Text information (ammo) and icons (stance)
- Colour-changing crosshair
- Red arrow to show direction of incoming bullets
A.1.37 State of Decay

Fig. A.38: https://www.youtube.com/watch?v=kT96eI0udYI at 6:53

Release date 5/06/2013

Players have a mini-map and health bar. At an “Examining” site, players can discover places of interest and annotations are placed on the screen where those places are. Marking one of these locations on the map makes a 2D-icon float in the 3D-world above that location.

AR helpers

- Icons and names on objects of interest

Widget Helpers

- Mini-map
- Health/energy-bars
- Text information (ammo) and icons (stance)
A.1. Discussed games

A.1.38 Okami

Fig. A.39: https://www.youtube.com/watch?v=zoMLeETx400 at 12:56

Release date 20/04/2006
Okami has a few AR-techniques to help the player find his/her way. It can be a 3D-arrow in front of the wolf that points in the right direction or it can be a line of items dropped by an NPC that the player needs to follow/race. NPC’s get an icon above their head indicating their function. In combat, the game also features health bars on the HUD.

AR-helpers

- Icons above NPC’s
- A line of items to show the way
- 3D-arrow that points in the right direction

Widget-Helpers

- Health-bars
- Text information
A.1.39  Fallout

Release date  30/09/1997
Fallout is another relatively old game. The HUD shows a bit of information in text-form like remaining HP. The game uses a mouse-cursor for selecting enemies. The cursor changes form and colour depending on what is underneath it. A selected enemy is highlighted.

AR-helpers

- Highlighted enemy

Widget-Helpers

- Colour- and form-changing cursor
- Text information
A.1.40  Grand theft auto

Fig. A.41:  https://www.youtube.com/watch?v=09K_G4TRvLQ at 2:09

Release date  October 1997  
The first of the GTA-series. The game has a simple HUD only showing information about equipment and the score. It has one helper in the world and that's a 2D-arrow around the playable character that points in the direction of an objective. When hitting objects, the gathered score floats out of that object.

AR-helpers

- 2D-arrow pointing in the direction of an objective

- Score coming out of related objects

Widget-Helpers

- Text on HUD
A.1.41 True Crime: Streets of LA

Fig. A.42: https://www.youtube.com/watch?v=nmzZJllRHXk at 1:01

Release date  3/11/2003
On the HUD, a health bar and mini-map is displayed. In the world, objectives/enemies can have a cylinder of light coming out of them or get a floating icon above them.

AR-helpers

- Icons above NPC’s of interest

- Cylinder of light around way-point

Widget-Helpers

- Plain text

- Mini-map

- Health bar
A.1.42 Fable

Release date 14/09/2004
Fable does not use a lot of helpers. On the HUD we can see a mini-map along with a health- and mana bar, and a skill-bar. In the 3D-world enemies and other NPC’s of interest get highlighted to indicate that they can be interacted with.

AR-helpers

- Highlighted NPC’s

Widget-Helpers

- Plain text
- Mini-map
- Health-, mana- and skill-bar
Appendix A. Study of 100 Open World Games

A.1.43 Mafia

Mafia does not use any AR-helpers. There is a HUD with a compass, mini-map and when driving a speed indicator and also a health-number and bullets left.

Release date 28/08/2002

AR-helpers None

Widget-Helpers

• Plain text
• Mini-map
• Compass
• Speed indicator
A.1.44  Shenmue

*Fig. A.45: [https://www.youtube.com/watch?v=I5gIg5HSi08](https://www.youtube.com/watch?v=I5gIg5HSi08) at 45:49*

**Release date**  29/12/1999

This game only has a compass that gets swapped by a clock on occasions. Other interactions are shown on the HUD as text or icons. (when being able to talk to someone the corresponding button to talk appears in the bottom right corner)

**AR-helpers**  None

**Widget-Helpers**

- Plain text and icons
- Compass
A.1.45 Ride to hell retribution

Fig. A.46: https://www.youtube.com/watch?v=6EC8rvcH7w0 at 17:05

Release date  28/06/2013
This game shows the currently selected weapon on the HUD and can floating icons above enemies’ heads on incoming attacks.

AR-helpers

- Floating icons above enemies’ heads

Widget-Helpers

- Plain text and icons
This game uses many different mechanics to integrate menu-functionalities in the game-world. There is no HUD but the information that would be there is now floating in 3D-space next to the playable character. It keeps track of health, ammo, co-op partners, ... When the player opens the “menu”, in stead of some other screen popping up, the playable character will look at his/her watch and the menu will be a holographic representation floating around it.

The map is integrated in the 3D-world as well, as a holographic representation of the mini-map forms around the playable character. On this map different elements can be selected to get more information. For cinematic moments the game once again shows holograms of what happened at that location. These holograms can have text on them to give extra information. Way-points can get annotations by having text float above them to show what they are.

Enemies and allies get floating health bars above their head. Collected points float out of the playable character. There is an option to get a bit of an “x-ray” view for a moment. The cross-hair can change form to show different actions like reloading the gun.

AR-helpers

- Floating menu and HUD
- Health-bar, ammo, co-op partners on floating menu
- “mega-map” that surrounds the player
- Holograms to show what happened at a certain location
- Floating annotations and labels
- Floating health bars and scores
- “Echo-view” to see through objects

**Widget-Helpers**

- Form-changing cross-hair to show actions (e.g. reloading)

**A.1.47 Driver: San Francisco**

![Image of Driver: San Francisco](https://www.youtube.com/watch?v=dZqRXsOujis)

*Fig. A.48: [https://www.youtube.com/watch?v=dZqRXsOujis](https://www.youtube.com/watch?v=dZqRXsOujis) at 9:16*

**Release date** 28/06/2013

Driver: San Francisco has a HUD with basic things like a mini-map with built-in compass, speedometer, objectives in text, ...

In the 3D-world way-points can have a big circle of light floating above them or a giant 2D-arrow. Way-points are also shown on the HUD with a 2D-icon that stays on the screen and sticks on the edge if the way-point is no longer in sight. These icons also show how much further the way-point is. There is also the option to “tag” other cars so the player can keep track of them and they get an icon floating above them.

This game also has a way to fly through the city to select a car the player wants to drive with, with a form-changing cross-hair to show what can be selected and what cannot.

**AR-helpers**

- Floating icons above objects
• Floating 2D-arrows above way-points
• Light-circle around way-point

Widget-Helpers
• Plain text and icons
• mini-map
• Speedometer, ...
• 2D-icons pointing in the direction of a 3D-object
• form-changing cross-hair

A.1.48 Urban Chaos

Fig. A.49: https://www.youtube.com/watch?v=bbwx_k67bJc at 3:06

Release date  30/11/1999
Another relatively old game. The game has some kind of mini-map. It works as a radar. There’s a health bar and objectives and information are shown as text and icons. When fighting enemies, they get a health bar that stays near them.

AR helpers
• Health-bar near enemies
Widget-Helpers

- Plain text and icons
- Radar
- Health-bar

A.1.49 The Elder Scrolls III: Morrowind

![Morrowind game screenshot](https://www.youtube.com/watch?v=nhP49P1N6HY)

**Fig. A.50:** [https://www.youtube.com/watch?v=nhP49P1N6HY](https://www.youtube.com/watch?v=nhP49P1N6HY) at 9:22

**Release date** 1/05/2002

Morrowind has a mini-map and some information bars on the HUD. There are also some other icons to display information. The only AR-technique morrowind uses is floating text above objects to show what it is.

**AR-helpers**

- Floating text above objects

**Widget-Helpers**

- Plain text and icons
- Mini-map
- Health-bar
A.1. Discusses games

A.1.50 Ultima VII

Release date 16/04/1992
Ultima VII has a few health-bars on the HUD that show the party-members health along with icons to show which characters are currently in your party. The cursor changes form depending on the action (moving, selecting, ...). Items can have their name on top of them.

AR-helpers

- Text on objects

Widget-Helpers

- Plain text and icons
- Health-bars
A.1.51  Borderlands

Fig. A.52: [https://www.youtube.com/watch?v=GDONdCo_bEU](https://www.youtube.com/watch?v=GDONdCo_bEU) at 1:34

**Release date**  20/10/2009

The HUD has a health- and shield-bar along with a compass that shows the direction of the current mission objective. There’s also an xp-bar, an ammo-bar, the skill icon and the current mission details. When shooting enemies, the cross-hair changes colour on successful hits and the enemies’ health bars float above them. When hitting them, your damage is shown as a number floating out of them and the cross-hair. When you’re getting hit, a red bar will appear in the edge of the screen indicating the direction of incoming fire. When looking at items and standing near them, their name of action that can be done with them appears on them.

**AR-helpers**

- Text on objects
- Floating health bars above enemies
- Hit-splashes come out of enemies

**Widget-Helpers**

- Plain text and icons
- Health-, shield-, xp- and ammo-bars
- Colour-changing cross-hair
- Red bars in edge of screen indicating direction of incoming fire
A.1. Discussed games

A.1.52 The Witcher

Fig. A.53: [https://www.youtube.com/watch?v=5nLcXjNAmxk](https://www.youtube.com/watch?v=5nLcXjNAmxk) at 9:58

**Release date** 26/10/2007

The health- and “mana”-bar are shown on the HUD along with a mini-map and icons to open the menus. When selecting an enemy as a target to attack, it gets a red circle around its feet. Objects that can be interacted with have their name floating above them. When attacking an enemy, damage is shown by having the amount of damage float out of the enemy as text.

**AR-helpers**

- Text above objects

- Hit-splashes come out of enemies

**Widget-Helpers**

- Plain text and icons

- Health- and mana-bar

- Coloured circle around selected NPC

- Mini-map
A.1.53 Star Wars: Knights of the Old Republic

![Image](https://www.youtube.com/watch?v=JC-UCTr-XlQ) at 5:10

Release date 15/07/2003

There are health bars for units on the HUD along with a mini-map. Objects that can be interacted with get a circle floating around them. Targeted enemies get a few red arrows floating around their head and when damaging them, the damage flies out of them.

**AR-helpers**

- Circle or arrows around objects
- Hit-splashes come out of enemies

**Widget-Helpers**

- Plain text and icons
- Health-bar
- Mini-map
A.1.54 Need for speed: most wanted

Release date 11/11/2005

Need for speed: most wanted has a HUD with a mini-map, position indicator and a tachometer with speedometer. The game uses 3D light-arrows at intersections that are also used as walls to show the player where to drive to. Buildings that can be destroyed by driving through them get a label floating in front of them and cooldown-locations have a glowing circle above them.

AR-helpers

- 3D-arrows as walls that point the way
- Labels in front of destroyable buildings
- Glowing circle around cool-down location

Widget-Helpers

- Plain text and icons
- Mini-map
- Speedometer and tachometer
A.1.55  L.A.: Noire

Release date  17/05/2011
The screen only has a mini-map with the option to display text to show the current objectives. There are no additional helpers except that if the game is played with a controller, it will make the controller vibrate when the player is investigating interesting objects.

AR-helpers  None

Widget-Helpers

- Plain text
- Mini-map
- Vibrating controller to tell the player he/she is looking at the right angle of an object
A.1.56  Assassin’s Creed

Release date  13/11/2007
For this game I would once again like to refer to section A.1.2 because this is the earliest game in that series and contains less helpers than the game in section A.1.2. There are the same ways to highlight NPC’s and show icons near or above them. There’s also a mini-map, health bar, ...

AR-helpers  See section A.1.2

Widget-Helpers  See section A.1.2
A.1.57 Fable 2

Release date 21/10/2008
This is the previous version of the game in section A.1.23 and has about the same helpers except that there is a feature in this game that can highlight characters of interest.

AR-helpers

- Highlighting characters of interest

- See section A.1.23

Widget-Helpers See section A.1.23
A.1.58 Spiderman: web of shadows

Fig. A.59: https://www.youtube.com/watch?v=BFWPHv5_w7c at 2:15

Release date 21/10/2008
This game has a mini-map and health-bars for the playable character as well as the characters the player is fighting with on the screen. There’s a 2D arrow-icon that stays near the edge of the screen and points in the direction of objects. Some important objects get a ray of light coming out of them or an icon floating above them.

AR-helpers

- Ray of light coming out of objects
- Icons floating above objects

Widget-Helpers

- Mini-map
- Health-bar
- 2D-icon that points in the direction of objects
Appendix A. Study of 100 Open World Games

A.1.59 Infamous 2

Fig. A.60: [https://www.youtube.com/watch?v=KRM2JSUACHQ](https://www.youtube.com/watch?v=KRM2JSUACHQ) at 2:29:48

Release date 7/06/2011

Infamous 2 has a mini-map and some kind of mana-bar on the screen. Targeted characters get highlighted and certain objects of interest get a glow around them. There is no health bar but when the player takes damage, some blood-textures are shown near the edges of the screen, indicating to the player that his/her health is low. The cross-hair changes form/colour depending on the actions of the player.

AR-helpers

- Highlighted characters
- Glow around certain objects

Widget-Helpers

- Mini-map
- Mana-bar
- Blood-textures near the edges of the screen to indicate low health
A.1.60 The Incredible Hulk Ultimate destruction

Release date 23/08/2005
This game has a mini-map with built-in compass, a health-bar and can show objects of interest with a ray of light coming out of them. The game can also display 2D arrow-icons near the edge of the screen pointing in the direction of enemies or objectives.

AR-helpers

- ray of light coming out of certain objects

Widget-Helpers

- Mini-map
- health-bar
- 2D arrow-icons near the edge of the screen pointing in the direction of enemies or objectives
Appendix A. Study of 100 Open World Games

A.1.61 Red Faction: Guerrilla

Release date 2/06/2009

The HUD has a mini-map with built-in compass and a health bar that appears when under fire. Small red arrows will also appear in the direction of incoming bullets. When driving to an objective, a yellow line will be shown on the floor to tell the player what direction to drive to. Way-points or objectives can have a glowing cylinder come out of them along with a floating icon. There’s also a form- and colour-changing cross-hair to indicate what the player can do with a certain object.

AR-helpers

- light-cylinder coming out of way-points
- Yellow line on the ground that shows the way to go
- Floating icons above objects

Widget-Helpers

- Mini-map
- health-bar
- 2D arrow-icons near the edge of the screen pointing in the direction of enemies
- Form- and colour-changing cross-hair
A.1.62 Infamous

Fig. A.63: https://www.youtube.com/watch?v=SPoDk0xAu_8 at 6:07

Release date 26/05/2009
Infamous is an earlier game of the same series as the game in section A.1.59 and therefore uses about the same helpers except that this game also uses icons floating above objects to indicate e.g. that a mission can be started there.

AR-helpers

- Floating icons

- See section A.1.59

Widget-Helpers

- See section A.1.59
A.1.63 Dragon’s dogma

Fig. A.64: https://www.youtube.com/watch?v=OyRhXjdqa1Y at 3:18

Release date 22/05/2012

Dragon’s dogma has a mini-map and shows the player’s health on the screen along with all the possible moves and their corresponding buttons. Enemies and friendlies get a floating health bar above their head. The game also has a form-changing cross-hair when using a bow to indicate how far the arrow will fly (how long the player charges it).

AR helpers

- Floating health bars

Widget helpers

- Mini-map
- Move-set and text
- Form-changing cross-hair
A.1. Discussed games

A.1.64 Tomb Raider (2013)

Fig. A.65: https://www.youtube.com/watch?v=AjRLKp6sKJMc at 0:36

Release date 5/03/2013

The game does not use a health-bar but shows blood-textures when the character is getting damage. The cross-hair changes form and colour depending on the action and target. The button you need to press to pick up an item appears on top of items when standing near them. The player can “scan” the environment to show the current goal. A ray of light will come out of it.

AR-helpers

- Ray of light coming out of objective

Widget-Helpers

- Plain text
- Form- and colour-changing cross-hair
- Blood textures in stead of a health-bar
A.1.65 Shenmue 2

Release date 6/09/2001

Shenmue 2 looks a lot like its predecessor in section A.1.44. This game only has a compass with a clock inside of it. Other interactions are shown on the HUD as text or icons. (when being able to talk to someone the corresponding button to talk appears in the bottom right corner)

AR-helpers None

Widget-Helpers

- Plain text and icons
- Compass
A.1.66  Saints Row 2

Release date  14/10/2008

Another game that looks a lot like its predecessor in the series (see section A.1.66). For AR-helpers, this game mainly uses floating icons above enemies, objectives, ... Way-points or locations with a functionality get a glowing circle on top of them. The game has a mini-map and a health bar along with the currently selected weapon. When aiming at enemies, the cross-hair changes colour. Doors that can be opened have a hand-print on them. When under fire, the direction of enemies will be shown by a red 2D-arrow on the HUD.

AR-helpers

- Floating icons above objects to indicate their meaning
- Glowing cylinder around way-points
- Hand-prints on doors that can be opened

Widget-Helpers

- Health-bar
- Mini-map
- Plain text
- Colour-changing cross-hair
A.1.67  Grand Theft Auto: Vice City

Release date  27/10/2002
GTA: Vice City has a mini-map with built-in compass that also shows the direction of objectives. Objectives are shown in the 3D-world by having a cylinder of light coming out of them. Objects can also have floating icons above them to e.g. show to follow someone.

AR_helpers

- Floating icons above objects
- Glowing cylinder around way-points

Widget-Helpers

- Mini-map
- Plain text
- Colour-changing cross-hair
A.1.68 Yakuza

Release date  8/12/2005
Yakuza has a mini-map and shows health bars of your character and the enemies during combat on the HUD. Other than that, there are only floating icons above characters that can be interacted with.

AR-helpers

- Floating icons above objects

Widget-Helpers

- Mini-map
- Health bars
- Plain text
A.1.69  Just Cause 2

Release date  23/03/2010
There’s a mini-map with a health-bar and an icon for showing the currently equipped weapon. When objects can be interacted with, the button needed for interaction appears on them. Enemies can be targeted which shows a small target box on them. Way-points and objectives are shown on the screen by putting a 2D-icon on them when they’re in sight and a 2D-arrow icon near the edge of the screen to point the player to the direction he/she needs to look. The cross-hair also changes form and colour depending on the currently used action.

AR-helpers
- Floating icons above objects
- Target-box around objects
- 2D-icons on the screen pointing to way-points

Widget-Helpers
- Mini-map
- Health bar and weapon-icons
- Form- and colour-changing cross-hair
- Plain text
A.1.70 Saints Row 3

Release date  15/11/2011
Saints Row 3 keeps the helpers from its predecessors in section A.1.31 and section A.1.66 with the addition of 3D-arrows that are added on the road to show the way to drive without having to look at the mini-map.

AR-helpers

- See section A.1.31 and section A.1.66

- 3D-arrows on roads to show where to drive to

Widget-Helpers

- See section A.1.31 and section A.1.66
A.1.71  Sleeping dogs

![Game Screenshot](https://www.youtube.com/watch?v=OEFK_aI_3w) at 7:25

**Release date**  13/08/2012

This game has a mini-map with a built-in compass and health bar. It uses 2D arrow-icons that stick near the edge of the screen to point to the direction the player has to drive to. When the corresponding objective to these icons is in sight, they will float on this objective. Floating 3D-arrows are shown on roads to show the player which turns he/she has to take. In combat-situations enemies are highlighted to show which enemy is about to hit you. When standing near NPC’s that can be interacted with, icons will appear above their heads.

**ARHelpers**

- Icons floating on top of objects
- 3D-arrows on the road that tell you which turn to take
- Highlighted enemies

**Widget-Helpers**

- Mini-map
- Compass
- Health-bar
- Plain text
A.1.72 Far Cry 2

Release date 21/10/2008
When driving, the playable character has the option to take his map out. This map will get additional information on it about the player’s position and nearby events. The player’s position is shown on this map by having a green arrow on the map that moves along with the player. Because of this map, the game does not use a mini-map. A health-bar appears in combat-situations. Items that can be picked up are highlighted. When the player gets damage, blood textures appear near the edge of the screen in the direction of the incoming damage. The cross-hair can also change form depending on the aiming of the player.

AR-helpers

- Highlighted objects
- 3D-map with augmented information

Widget-Helpers

- Health-bar
- Plain text
- Blood textures near the edge of the screen in the direction of incoming damage
A.1.73 Minecraft

Fig. A.74: http://cache.gamercreated.com/M/1024x768-tagged-scale/25260-MinecraftMaps_20110526200551.jpg

Release date  17/05/2009

Minecraft does not use many techniques to help out the player. It has a very basic HUD with a health-, hunger- and item-bar. Players do have the option to craft a map however that maps the surrounding area and puts a moving icon on it that resembles the player.

AR-helpers

- 3D-map with augmented information

Widget-Helpers

- Health-bar
- Plain text
- Hunger-bar
A.1.74  Assassin’s Creed 2

Fig. A.75: https://www.youtube.com/watch?v=QbprGxi5MLU at 2:22

Release date  17/11/2009
Another early game in the Assassin’s Creed-series where I would like to refer back to section A.1.2

AR-helpers

- See A.1.2

Widget-Helpers

- See A.1.2
A.1.75 Far Cry 3

Fig. A.76: https://www.youtube.com/watch?v=2Kbn8Q3PABI at 5:01

Release date  29/11/2012
Far Cry 3 uses a mini-map with built-in compass and health bar. When the player takes out his/her camera, he/she can “tag” enemies which makes icons appear float above enemies. This also makes the enemies visible through walls and other objects. 2D-icons appear when the player is detected or under fire. These icons point towards the corresponding enemy.

AR-helpers
- Icons floating on top of objects
- Enemies visible through walls and objects
- Information on loot-able objects

Widget-Helpers
- Mini-map
- Compass
- Health-bar
- Plain text
- 2D-arrows on the HUD that point in the direction of enemies
A.1. Discussed games

A.1.76 Saints Row 4

Release date 20/08/2013
This game is a newer version of the games in section A.1.31, section A.1.66 and section A.1.70 and uses the same helpers. This game has an additional feature during racing mini-games that puts floating orb-like objects on the road in a line that the player needs to follow.

AR-helpers

- See section A.1.31, section A.1.66 and section A.1.70

- Floating orb-like objects in a line on the road that the player needs to follow

Widget-Helpers

- See section A.1.31, section A.1.66 and section A.1.70
Appendix A. Study of 100 Open World Games

A.1.77 Fallout 3

Release date 28/10/2008
Fallout 3 is an earlier version of the Fallout-series than the game discussed in section A.1.29 and uses the same helpers.

AR-helpers

- See section A.1.29

Widget-Helpers

- See section A.1.29
A.1.78  The Legend of Zelda: A Link to the past

Release date  21/11/1991
Another relatively old game without a lot of helpers. There’s only a basic HUD with a health-, and power-bar that also shows the currently equipped item. Upon a successful hit on an enemy, it gets highlighted.

AR-helpers

- Enemies get highlighted when hit

Widget-Helpers

- Power-, and health-bar

- Plain text
Appendix A. Study of 100 Open World Games

A.1.79 Final Fantasy VII

Fig. A.80: [https://www.youtube.com/watch?v=gNqTw5KV81o](https://www.youtube.com/watch?v=gNqTw5KV81o) at 12:30

Release date 31/01/1997

Final Fantasy VII only uses some helpers during combat situations. A 3D-icon floats above the attacking character and a 2D-hand icon is used to select a target. There are also health-, MP-, Limit- and Time-bars along with the menu to select attacks on the HUD.

**AR的帮助**

- Floating 3D-icon above attacking character

- 2D-icon next to target

**Widget-Helpers**

- Health-, MP-, Limit- and Time-bars

- Plain text
A.1.80  Far Cry

Release date  23/03/2004
This is the first game in the Far Cry-series discussed in section A.1.72 and section A.1.75 but does not use as many helpers. Far Cry only has a HUD with some information-bars that display health, energy, ... and some icons to show the currently equipped weapons. It does not use any AR-helpers. The only extra helpers it uses is a radar in stead of a mini-map with a compass and a form-changing cross-hair.

AR-helpers  None

Widget-Helpers

- Health-, energy- and ammo-bar
- Plain text
- Form-changing cross-hair
- Compass
- Radar
A.1.81 Grand Theft Auto IV

Release date  29/04/2008

GTA 4 has a mini-map with a built-in compass and also shows the health-bar around this. 2D-icons are drawn above way-points and enemies to signify their function. A red glow appears around important portable objects.

**AR-helpers**

- Floating icons above way-points and NPC’s
- Red glow around important objects

**Widget-Helpers**

- Health-bar
- Plain text
- Form-changing cross-hair
A.1.82 Assassin’s Creed: Brotherhood

*Fig. A.83: [https://www.youtube.com/watch?v=ZtAjR5uNUGU](https://www.youtube.com/watch?v=ZtAjR5uNUGU) at 4:24*

**Release date** 16/11/2010

Another earlier game in the Assassin’s Creed-series where I’d like to refer back to section A.1.2 for because this game uses the same (or less) helpers. This game also uses a mechanic for showing the player the aim on character’s wrist-gun. A holographic line will be drawn from the character’s hand to the target and when the line is fully drawn, it means the target is “locked”.

**AR-helpers**

- Holographic line to show current target and aim
- See section A.1.2

**Widget-Helpers**

- See section A.1.2
Appendix A. Study of 100 Open World Games

A.1.83 Destiny

![Destiny Image]

Fig. A.84: [https://www.youtube.com/watch?v=mfzIDQQLCqk](https://www.youtube.com/watch?v=mfzIDQQLCqk) at 3:10

**Release date** 9/09/2014

Destiny does not have a mini-map but uses a NAV-MODE to show a 2D icon on the place where the objective would be if it would be in sight. The icon floats above the objective when it is in sight. The position of this kept on a radar in the top-left edge of the screen to show where you need to go. There is a health-bar that appears during combat. Damage is also shown by drawing blood textures near the edges of the screen upon being hit. Enemies’ health is shown by a health-bar floating above their head. Other player’s names and level float above their head. There’s also an xp-bar and icons to show the currently equipped weapon on the HUD. A red 3D-arrow appears around the player when being hit that points in the direction of the shooting enemy. The cross-hair can also change form to indicate current actions.

**AR-helpers**
- 2D-icon floating on objective
- Floating health-bar and level above enemies and players
- 3D-arrow around the player that points to enemies

**Widget-Helpers**
- Radar
- Health- and xp-bar
- Blood textures near edges of screen when hit
- Plain text and icons
A.1.84 The Witcher 3: Wild Hunt

Release date  19/05/2015
The Witcher 3 has a mini-map on the HUD with the current objective below it and some information about the time and weather next to it. A health- and mana-bar appears during combat situations. Characters that can be interacted with get their name or “class” floating above them along with their health if they’re an enemy. Certain hunting clues (e.g. monster tracks) are highlighted to help the player see them. One can also assume a way for seeing enemies through walls will return by having glowing silhouettes that represent enemies behind these walls because both previous games in the series had this mechanic.

AR-helpers

- 2D-icon floating on objective
- Floating health-bar and level above enemies and players
- Glowing silhouettes to represent enemies behind objects
- Highlighted objects

Widget-Helpers

- Mini-map
- Health- and mana-bar
- Plain text and icons
A.1.85  Assassin’s Creed IV: Black Flag

Fig. A.86: https://www.youtube.com/watch?v=hWgptUuJxBo at 7:26

Release date  29/10/2013
Another game in the Assassin’s Creed series that retains the same mechanics as in previous versions. Only the visualization is slightly different (icons looking slightly different) but they work the same.

AR-helpers

• See secion A.1.2

Widget-Helpers

• See secion A.1.2
A.1.86 The Elder Scrolls IV: Oblivion

Release date 20/03/2006
The Elder Scrolls IV: Oblivion has a compass and health-, stamina-, and mana bar on the HUD. The cross-hair also changes form depending on the action (e.g. it turns into a different icon when standing close enough to NPC’s to talk to them).

AR-helpers None

Widget-Helpers

- Compass

- Plain text and icons

- Health-, stamina- and mana-bars

- Form-changing cross-hair
A.1.87  Shadow of the Colossus

Release date  18/10/2005
Shadow of the Colossus has a simple HUD that appears during combat and shows the player’s and enemies’ health bars and currently equipped weapon.

AR-helpers

- Highlighted “weak spots”

Widget-Helpers

- Plain text and icons

- Health-bar
A.1.88  The Legend of Zelda: The Wind Waker

Release date  13/12/2002
This game shows a health-bar in the top-left part of the screen and the possible moves the player can execute along with the button the player has to press. There’s also a compass and clock on the screen. A 3D-arrow that’s next to the player points to the direction he/she needs to go. Rupees (in-game currency) can also be tactically placed in a line so the player goes the right direction. Enemies get an arrow floating above their head, indicating which one you’re currently attacking.

AR-helpers

• 3D-arrow next to player that points to objective
• Arrows floating above enemies
• Rupees in a line that the player needs to follow

Widget-Helpers

• Plain text and icons
• Health-bar
• Move-set on the HUD
• Compass and clock
A.1.89 Dragon Age: Origins

Fig. A.90: [https://www.youtube.com/watch?v=OKy1pXHWbz4](https://www.youtube.com/watch?v=OKy1pXHWbz4) at 13:46

Release date 13/12/2002

Dragon Age: Origins has a mini-map, a skill-bar and a health-, and mana-bar around the player icon along with the menu-buttons. During combat, the selected enemy will have its name and health above its head and have a red circle around it. When hovering with the cursor over objects that can be interacted with, they will be highlighted.

AR-helpers

- Health and name above enemies
- Highlighted objects

Widget-Helpers

- Plain text and icons
- Health- and mana-bar
- Skill-bar
- Mini-map
A.1.90  The Legend of Zelda: Skyward Sword

Fig. A.91: [https://www.youtube.com/watch?v=55KyWpr1FQI](https://www.youtube.com/watch?v=55KyWpr1FQI) at 7:31

**Release date**  18/11/2011

The Legend of Zelda: Skyward Sword has a HUD with a health-bar and the currently equipped move-set/items along with an energy bar and money-counter. Enemies get an arrow floating above their head.

**AR helpers**

- Arrows floating above enemies

**Widget-helpers**

- Plain text and icons
- Health- and energy-bar
- Move-set
A.1.91  Grand Theft Auto: San Andreas

Fig. A.92: https://www.youtube.com/watch?v=o-mJykfoRbE at 16:36

Release date  26/10/2004
For this game I would like to refer back to GTA: Vice City in section A.1.67 as this game is the direct successor to that game and does not introduce any new helpers except that the health-number has been replaced by a bar.

AR-helpers

• See section A.1.67

Widget-Helpers

• Health-bar

• See section A.1.67
A.1.92 Borderlands 2

Release date 18/09/2012
Borderlands 2 has a HUD with a health-bar, an xp-bar and a bar for showing how much ammo you have left. The current objective is also always displayed in the top-right corner of the screen. When looking at NPC’s their name and health will appear above their head. When standing near items other than ammo and looking at them, their a small information box will float above them. When you need to collect items for a certain mission, a holographic representation will be shown where you have to put the real item so the player easily knows where to put the item. When the objective is a door or object, an icon will stick to this object. When hit a blood- or shield-texture will appear near the edge of the screen indicating where the incoming fire is coming from. The game also has a mini-map and colour- and form-changing cross-hair.

AR-helpers
- Health and names above NPC’s
- Information pop-ups above items
- 2D-icon that points to objective
- Holographic representation of items to show where to put them

Widget-Helpers
- Health-, ammo- and xp-bar
• Blood- or shield-texture near the edge of the screen pointing to incoming fire
• Plain text and icons

A.1.93 Watch Dogs

Fig. A.94: [https://www.youtube.com/watch?v=POqAMFdcsjg](https://www.youtube.com/watch?v=POqAMFdcsjg) at 1:26

Release date 27/05/2014
Watch Dogs has a very basic HUD with a mini-map with built-in compass and an energy meter. It helps the player by drawing a line on the road to the current objective, highlighting interact-able objects, showing their names on these objects and having a 2D-icon on the screen that points to the objective. There’s also an ability to hack security cameras which allows the player not only to get an overview of the current situation but to also see through walls. Icons will appear on top of enemies that spot the player. When being hit a blood-texture will be drawn near the edge of the screen to indicate low health along with a red arrow that points to the direction of the incoming bullets. NPC’s can be inspected and an information-box will pop-up. The cross-hair also changes form and colour depending on the current weapon or action.

AR-helpers
• Line on road that the player needs to follow to get to the current objective
• Highlighting interact-able objects
• 2D-icon that points to current objective
• See through walls
• Information pop-ups and icons above NPC’s
• 2D-icon that points to objective
• Red 2D-arrow pointing to attacking enemies

Widget-Helpers
• Mini-map with built-in compass
• Energy meter
• Plain text and icons
• Blood-textures near the edge of the screen indicating low health in stead of a health-bar

A.1.94 Metal Gear Solid V: The Phantom Pain

![Image of Metal Gear Solid V]

Fig. A.95: [https://www.youtube.com/watch?v=seChZifeI7k](https://www.youtube.com/watch?v=seChZifeI7k) at 5:53

*Release date* 2015

Metal Gear Solid V only shows a HUD that displays the currently equipped weapon and how much ammo there’s left when the player starts aiming. Objectives get a box around them and when they’re out of sight a 2D-arrow icon stays near the edge of the screen pointing towards objectives. Enemies can be tagged which makes an icon float above their head and makes them highlighted and visible through other objects. When an enemy is about to spot you, a white 2D-arc will appear on the screen pointing to the enemy. When hit, blood-textures will appear on the edge of the screen indicating low health. When aiming, the cross-hair changes form and colour indicating hits and aiming.
AR-helpers

- 2D-box around objectives
- See through walls and objects
- Icons above NPC’s
- Highlighted enemies

Widget-Helpers

- Icons and text for equipped weapon
- Plain text and icons
- Blood-textures near the edge of the screen indicating low health in stead of a health-bar
- 2D-icon near the edge of the screen that points to current objective
- 2D-icon that points to enemies

A.1.95 Red Dead Redemption

Fig. A.96: https://www.youtube.com/watch?v=cVBsIuTT5iY at 0:34

Release date 18/05/2010

Red Dead Redemption has a mini-map with a built-in compass. The only other helpers are a health-bar next to the mini-map and the mini-map itself which like most mini-maps also displays the direction objectives, and floating yellow X-icons in front of buildings that can be entered.
AR-helpers

- 2D-icons floating in front of buildings that can be entered

Widget-Helpers

- Health-bar
- Plain text and icons
- Mini-map

A.1.96 The Legend of Zelda: Ocarina of time

Release date 21/11/1998

The Legend of Zelda: Ocarina of time has a health-bar on the HUD along with currently equipped items and which button to press to use them. A mini-map can be unlocked in dungeons by finding a map and compass. Targeted enemies or objects will get a floating icon above them.

AR-helpers

- Icons floating above targeted objects

Widget-Helpers

- Health-bar
- Plain text and icons
- Mini-map
A.1.97 The Witcher 2

Fig. A.98: [https://www.youtube.com/watch?v=nmNIJ3rS9C4](https://www.youtube.com/watch?v=nmNIJ3rS9C4) at 23:00

**Release date** 17/05/2011

The Witcher 2 has a mini-map on the HUD with the current objective below it and some information about the time next to it. A health- and mana-bar appears during combat situations. Characters that can be interacted with get their name or “class” floating above them along with their health if they’re an enemy. Certain hunting clues (e.g. monster tracks) are highlighted to help the player see them. There’s an ability for seeing enemies through walls by having glowing silhouettes that represent enemies behind these walls.

**AR-helpers**

- 2D-icon floating on objective
- Floating health-bar and level above enemies and players
- Glowing silhouettes to represent enemies behind objects
- Highlighted objects

**Widget-Headers**

- Mini-map
- Health- and mana-bar
- Plain text and icons
A.1.98  Batman: Arkham City

Release date  18/10/2011

Batman: Arkham City has a compass at the top of the screen and a health-bar. “Detective view” can be started for seeing NPC’s and objects through walls. Targeted enemies will get a big icon floating above them. Enemies’ attacks can be blocked and the moment when to press a button will be shown to the player by making an icon appear above the enemy’s head. Ledges that can be grappled will get a white circle on them.

AR-helpers

- Icons floating above objectives, enemies and other objects
- Detective vision which allows the player to see through walls

Widget-Helpers

- Compass
- Health-bar
- Plain text and icons
A.1.99  Grand Theft Auto V

Fig. A.100: https://www.youtube.com/watch?v=s_3ltWTDH_0 at 12:20

Release date  17/09/2013
There’s a mini-map with built-in compass on the HUD. Beneath the mini-map is a health- and energy-bar. Way-points have a small yellow cylinder coming out of them. When being shot, a red bar will appear near the edge of the screen, pointing towards the shooting enemy.

AR-helpers

- Small yellow cylinders on way-points

Widget-Helpers

- Mini-map with compass
- Health-, and energy-bar
- Plain text and icons
- Red bar near the edge of the screen pointing to shooting enemies
A.1. Discussed games

A.1.100 The Elder Scrolls V: Skyrim

*Fig. A.101: [https://www.youtube.com/watch?v=B6uVksblx-w](https://www.youtube.com/watch?v=B6uVksblx-w) at 9:23*

**Release date**  11/11/2011
Skyrim has a compass that shows the direction of objectives. When standing near objectives, an icon will be shown on top of it. The HUD has a health-, mana- and energy-bar with at the top the possibility to show the health-bar of the enemy the player is currently attacking. When casting the spell “Clairvoyance” a trail of smoke will appear on the ground, leading the player towards his/her current objective. The game will also show blood-textures near the edge of the screen when the player gets damage.

**AR-helpers**
- 2D-icons floating above or on objectives
- A trail of smoke that leads to the current objective

**Widget-Helpers**
- Compass
- Health-, mana- and energy-bar
- Plain text and icons
- Blood-textures when the player gets damage
- The cross-hair can change form during certain actions (e.g. when sneaking the cross-hair will change into a closed eye which opens when spotted)
Appendix A. Study of 100 Open World Games

A.2 List Of Different Helpers And Games That Use Them

A.2.1 Plain text/buttons/icons on the screen

- Simpsons Road Rage  
- Assassin’s Creed: Liberation  
- Postal 3  
- Deadly Premonition  
- Dragon Age 2  
- Dead Island  
- DayZ  
- Mafia 2  
- Don’t Starve  
- The Elder Scrolls: Daggerfall  
- Final Fantasy XII  
- Mercenaries: Playground of Destruction  
- Freelancer  
- Harvest Moon  
- True Crime Streets of New York  
- The Simpsons Hit and Run  
- Midnight club Los Angeles  
- Midtown Madness 3  
- Gun  
- Prototype 2  
- Assassin’s Creed: Revelations  

[Image]
• Terraria (A.1.22)
• Fable 3 (A.1.23)
• Bully: scholarship edition (A.1.24)
• Mercenary (A.1.25)
• Crackdown (A.1.26)
• Mercenaries 2 (A.1.27)
• Fallout: New Vegas (A.1.29)
• Kingdoms of Amalur: Reckoning (A.1.30)
• Saints Row (A.1.31)
• Arma 2 (A.1.32)
• The Elder Scrolls: Arena (A.1.33)
• Burnout Paradise (A.1.34)
• Assassin’s Creed 3 (A.1.35)
• STALKER: Shadow of Chernobyl (A.1.36)
• State of decay (A.1.37)
• Okami (A.1.38)
• Fallout (A.1.39)
• Grand Theft Auto (A.1.40)
• True Crime Streets of LA (A.1.41)
• Fable (A.1.42)
• Mafia (A.1.43)
• Shenmue (A.1.44)
• Ride to hell retribution (A.1.45)
• Tom Clancy’s: The Division (A.1.46)
• Driver: San Francisco (A.1.47)
• Urban Chaos (A.1.48)
• Morrowind (A.1.49)
• Ultima VII (A.1.50)
• Borderlands (A.1.51)
• The Witcher (A.1.52)
• Star Wars: Knights of the Old Republic (A.1.53)
• Need for speed: most wanted (A.1.54)
• L.A.: Noire (A.1.55)
• Assassin’s Creed (A.1.56)
• Fable 2 (A.1.57)
• Spiderman: Web of Shadows (A.1.58)
• Infamous 2 (A.1.59)
• The Incredible Hulk Ultimate destruction (A.1.60)
• Red faction guerilla (A.1.61)
• Infamous (A.1.62)
• Dragon’s dogma (A.1.63)
• Tomb Raider (2013) (A.1.64)
• Shenmue 2 (A.1.65)
• Saints Row 2 (A.1.66)
• GTA: Vice City (A.1.67)
• Yakuza (A.1.68)
• Just cause 2 (A.1.69)
• Saints Row 3 (A.1.70)
• Sleeping Dogs (A.1.71)
• Far Cry 2 (A.1.72)
• Minecraft (A.1.73)
• Assassin’s Creed 2 (A.1.74)
• Far Cry 3 (A.1.75)
• Saints Row 4 (A.1.76)
• Fallout 3 (A.1.77)
• The Legend of Zelda: A Link to the past (A.1.78)
- Final Fantasy VII (A.1.79)
- Far Cry (A.1.80)
- GTA 4 (A.1.81)
- Assassin’s Creed: Brotherhood (A.1.82)
- Destiny (A.1.83)
- The Witcher 3: Wild Hunt (A.1.84)
- Assassin’s Creed IV: Black Flag (A.1.85)
- The elder scrolls IV: Oblivion (A.1.86)
- Shadow of the Colossus (A.1.87)
- The Legend of Zelda: Wind Waker (A.1.88)
- Dragon Age: Origins (A.1.89)
- The Legend of Zelda: Skyward Sword (A.1.90)
- GTA: San Andreas (A.1.91)
- Borderlands 2 (A.1.92)
- Watch Dogs (A.1.93)
- Metal Gear Solid V: The Phantom Pain (A.1.94)
- Red Dead Redemption (A.1.95)
- The Legend of Zelda: Ocarina of time (A.1.96)
- The Witcher 2 (A.1.97)
- Batman: Arkham City (A.1.98)
- GTA V (A.1.99)
- The elder scrolls V: Skyrim (A.1.100)
A.2.2 Information bars (health, mana, speed, ...) on HUD

- Assassin’s Creed: Liberation (A.1.2)
- Postal 3 (A.1.3)
- Deadly Premonition (A.1.4)
- Dragon Age 2 (A.1.5)
- Don’t Starve (A.1.9)
- The Elder Scrolls: Daggerfall (A.1.10)
- Final Fantasy XII (A.1.11)
- Mercenaries: Playground of Destruction (A.1.12)
- Freelancer (A.1.13)
- True Crime Streets of New York (A.1.15)
- Midnight club Los Angeles (A.1.17)
- Midtown Madness 3 (A.1.18)
- Gun (A.1.19)
- Prototype 2 (A.1.20)
- Assassin’s Creed: Revelations (A.1.21)
- Terraria (A.1.22)
- Bully: scholarship edition (A.1.24)
- Crackdown (A.1.26)
- Fallout: New Vegas (A.1.29)
- Kingdoms of Amalur: Reckoning (A.1.30)
- Saints Row (A.1.31)
- The Elder Scrolls: Arena (A.1.33)
A.2. List Of Different Helpers And Games That Use Them

- Assassin’s Creed 3 (A.1.35)
- STALKER: Shadow of Chernobyl (A.1.36)
- State of decay (A.1.37)
- Okami (A.1.38)
- True Crime Streets of LA (A.1.41)
- Fable (A.1.42)
- Driver: San Francisco (A.1.47)
- Urban Chaos (A.1.48)
- Morrowind (A.1.49)
- Ultima VII (A.1.50)
- Borderlands (A.1.51)
- The Witcher (A.1.52)
- Star Wars: Knights of the Old Republic (A.1.53)
- Need for speed: most wanted (A.1.54)
- Assassin’s Creed (A.1.56)
- Spiderman: Web of Shadows (A.1.58)
- Infamous 2 (A.1.59)
- The Incredible Hulk Ultimate destruction (A.1.60)
- Red faction guerilla (A.1.61)
- Infamous (A.1.62)
- Dragon’s dogma (A.1.63)
- Shenmue 2 (A.1.65)
- Saints Row 2 (A.1.66)
- GTA: Vice City (A.1.67)
- Yakuza (A.1.68)
- Just cause 2 (A.1.69)
- Saints Row 3 (A.1.70)
- Sleeping Dogs (A.1.71)
- Far Cry 2 (A.1.72)
- Minecraft (A.1.73)
- Assassin’s Creed 2 (A.1.74)
- Far Cry 3 (A.1.75)
- Saints Row 4 (A.1.76)
- Fallout 3 (A.1.77)
- The Legend of Zelda: A Link to the past (A.1.78)
- Final Fantasy VII (A.1.79)
- Far Cry (A.1.80)
- GTA 4 (A.1.81)
- Assassin’s Creed: Brotherhood (A.1.82)
- Destiny (A.1.83)
- The Witcher 3: Wild Hunt (A.1.84)
- Assassin’s Creed IV: Black Flag (A.1.85)
- The elder scrolls IV: Oblivion (A.1.86)
- Shadow of the Colossus (A.1.87)
- The Legend of Zelda: Wind Waker (A.1.88)
- Dragon Age: Origins (A.1.89)
- The Legend of Zelda: Skyward Sword (A.1.90)
- GTA: San Andreas (A.1.91)
- Borderlands 2 (A.1.92)
- Watch Dogs (A.1.93)
- Metal Gear Solid V: The Phantom Pain (A.1.94)
- Red Dead Redemption (A.1.95)
- The Legend of Zelda: Ocarina of time (A.1.96)
- The Witcher 2 (A.1.97)
- Batman: Arkham City (A.1.98)
- GTA V (A.1.99)
- The elder scrolls V: Skyrim (A.1.100)
A.2. List Of Different Helpers And Games That Use Them

A.2.3 Mini-map

- Simpsons Road Rage (A.1.1)
- Assassin’s Creed: Liberation (A.1.2)
- Dragon Age 2 (A.1.5)
- Dead Island (A.1.6)
- Mafia 2 (A.1.8)
- Final Fantasy XII (A.1.11)
- Mercenaries: Playground of Destruction (A.1.12)
- The Simpsons Hit and Run (A.1.16)
- Midnight club Los Angeles (A.1.17)
- Midtown Madness 3 (A.1.18)
- Gun (A.1.19)
- Prototype 2 (A.1.20)
- Assassin’s Creed: Revelations (A.1.21)
- Terraria (A.1.22)
- Bully: scholarship edition (A.1.24)
- Crackdown (A.1.26)
- Mercenaries 2 (A.1.27)
- Kingdoms of Amalur: Reckoning (A.1.30)
- Saints Row (A.1.31)
- Burnout Paradise (A.1.34)
- Assassin’s Creed 3 (A.1.35)
- STALKER: Shadow of Chernobyl (A.1.36)
• State of decay (A.1.37)
• True Crime Streets of LA (A.1.41)
• Fable (A.1.42)
• Mafia (A.1.43)
• Driver: San Fransisco (A.1.47)
• Morrowind (A.1.49)
• The Witcher (A.1.52)
• Star Wars: Knights of the Old Republic (A.1.53)
• Need for speed: most wanted (A.1.54)
• L.A.: Noire (A.1.55)
• Assassin’s Creed (A.1.56)
• Spiderman: Web of Shadows (A.1.58)
• Infamous 2 (A.1.59)
• The Incredible Hulk Ultimate destruction (A.1.60)
• Red faction guerilla (A.1.61)
• Infamous (A.1.62)
• Dragon’s dogma (A.1.63)
• Saints Row 2 (A.1.66)
• GTA: Vice City (A.1.67)
• Yakuza (A.1.68)
• Just cause 2 (A.1.69)
• Saints Row 3 (A.1.70)
• Sleeping Dogs (A.1.71)
• Assassin’s Creed 2 (A.1.74)
• Far Cry 3 (A.1.75)
• Saints Row 4 (A.1.76)
• GTA 4 (A.1.81)
• Assassin’s Creed: Brotherhood (A.1.82)
• The Witcher 3: Wild Hunt (A.1.84)
• Assassin’s Creed IV: Black Flag (A.1.85)
• Dragon Age: Origins (A.1.89)
• GTA: San Andreas (A.1.91)
• Borderlands 2 (A.1.92)
• Watch Dogs (A.1.93)
• Red Dead Redemption (A.1.95)
• The Legend of Zelda: Ocarina of time (A.1.96)
• The Witcher 2 (A.1.97)
• GTA V (A.1.99)
A.2.4 Compass

- The Elder Scrolls: Daggerfall \(A.1.10\)
- Mafia 2 \(A.1.8\)
- Freelancer \(A.1.13\)
- Midnight club Los Angeles \(A.1.17\)
- Midtown Madness 3 \(A.1.18\)
- Prototype 2 \(A.1.20\)
- Bully: scholarship edition \(A.1.24\)
- Mercenaries 2 \(A.1.27\)
- Fallout: New Vegas \(A.1.29\)
- Kingdoms of Amalur: Reckoning \(A.1.30\)
- The Elder Scrolls: Arena \(A.1.33\)
- Burnout Paradise \(A.1.34\)
- Mafia \(A.1.43\)
- Shenmue \(A.1.44\)
- Driver: San Francisco \(A.1.47\)
- Borderlands \(A.1.51\)
- The Incredible Hulk Ultimate destruction \(A.1.60\)
- Red faction guerilla \(A.1.61\)
- Shenmue 2 \(A.1.65\)
- Saints Row 2 \(A.1.66\)
- GTA: Vice City \(A.1.67\)
- Saints Row 3 \(A.1.70\)
A.2. List Of Different Helpers And Games That Use Them

- Sleeping Dogs (A.1.71)
- Assassin’s Creed 2 (A.1.74)
- Far Cry 3 (A.1.75)
- Saints Row 4 (A.1.76)
- Fallout 3 (A.1.77)
- Far Cry (A.1.80)
- GTA 4 (A.1.81)
- Assassin’s Creed: Brotherhood (A.1.82)
- The elder scrolls IV: Oblivion (A.1.86)
- The Legend of Zelda: Wind Waker (A.1.88)
- GTA: San Andreas (A.1.91)
- Borderlands 2 (A.1.92)
- Watch Dogs (A.1.93)
- Batman: Arkham City (A.1.98)
- GTA V (A.1.99)
- The elder scrolls V: Skyrim (A.1.100)

A.2.5 Radar

- Urban Chaos (A.1.48)
- Far Cry (A.1.80)
- Destiny (A.1.83)
A.2.6 3D-object around playable character that points in the direction of an object/goal

- Simpsons Road Rage \([A.1.1]\)
- Final Fantasy XII \([A.1.11]\)
- Midtown Madness 3 \([A.1.18]\)
- Okami \([A.1.38]\)
- Destiny \([A.1.83]\)
- The Legend of Zelda: Wind Waker \([A.1.88]\)
A.2.7 2D-object on the HUD that points in the direction of an object/goal/enemy

- Assassin’s Creed: Liberation [A.1.2]
- Mercenaries: Playground of Destruction [A.1.12]
- Freelancer [A.1.13]
- Assassin’s Creed: Revelations [A.1.21]
- Gun [A.1.19]
- Prototype 2 [A.1.20]
- Mercenaries 2 [A.1.27]
- Assassin’s Creed 3 [A.1.35]
- STALKER: Shadow of Chernobyl [A.1.36]
- Grand Theft Auto [A.1.40]
- Driver: San Francsico [A.1.47]
- Assassin’s Creed [A.1.56]
- Spiderman: Web of Shadows [A.1.58]
- Red faction guerilla [A.1.61]
- Saints Row 2 [A.1.66]
- Just cause 2 [A.1.69]
- Saints Row 3 [A.1.70]
- Sleeping Dogs [A.1.71]
- Far Cry 2 [A.1.72]
- Assassin’s Creed 2 [A.1.74]
- Far Cry 3 [A.1.75]
- Saints Row 4 [A.1.76]
• Assassin’s Creed: Brotherhood (A.1.82)
• Destiny (A.1.83)
• Assassin’s Creed IV: Black Flag (A.1.85)
• Borderlands 2 (A.1.92)
• Watch Dogs (A.1.93)
• Metal Gear Solid V: The Phantom Pain (A.1.94)

A.2.8 Line or objects in a line in the world that show the way

• The Simpsons Hit and Run (A.1.16)
• Fable 3 (A.1.23)
• Okami (A.1.38)
• Fable 2 (A.1.57)
• Red faction guerilla (A.1.61)
• Saints Row 4 (A.1.76)
• The Legend of Zelda: Wind Waker (A.1.88)
• Watch Dogs (A.1.93)
• The elder scrolls V: Skyrim (A.1.100)
A.2.9 X-ray/Heat-vision

- Assassin’s Creed: Liberation (A.1.2)
- Assassin’s Creed: Revelations (A.1.21)
- Assassin’s Creed 3 (A.1.35)
- Tom Clancy’s: The Division (A.1.46)
- Tomb Raider (2013) (A.1.64)
- Assassin’s Creed 2 (A.1.74)
- Far Cry 3 (A.1.75)
- Assassin’s Creed: Brotherhood (A.1.82)
- The Witcher 3: Wild Hunt (A.1.84)
- Assassin’s Creed IV: Black Flag (A.1.85)
- Watch Dogs (A.1.93)
- Metal Gear Solid V: The Phantom Pain (A.1.94)
- The Witcher 2 (A.1.97)
- Batman: Arkham City (A.1.98)
A.2.10 Annotations and labels/icons on or near objects

This can be a health bar floating above a character, the name of a gun floating above it when the player is looking at it, ...

- Assassin’s Creed: Liberation (A.1.2)
- Dragon Age 2 (A.1.5)
- Dead Island (A.1.6)
- DayZ (A.1.7)
- Don’t Starve (A.1.9)
- Final Fantasy XII (A.1.11)
- Mercenaries: Playground of Destruction (A.1.12)
- Freelancer (A.1.13)
- True Crime Streets of New York (A.1.15)
- The Simpsons Hit and Run (A.1.16)
- Midnight club Los Angeles (A.1.17)
- Midtown Madness 3 (A.1.18)
- Prototype 2 (A.1.20)
- Assassin’s Creed: Revelations (A.1.21)
- Fable 3 (A.1.23)
- Bully: scholarship edition (A.1.24)
- Crackdown (A.1.26)
- Mercenaries 2 (A.1.27)
- Fallout: New Vegas (A.1.29)
- Kingdoms of Amalur: Reckoning (A.1.30)
- Saints Row (A.1.31)
• Arma 2 (A.1.32)
• Assassin’s Creed 3 (A.1.35)
• STALKER: Shadow of Chernobyl (A.1.36)
• State of decay (A.1.37)
• Okami (A.1.38)
• Grand Theft Auto (A.1.40)
• True Crime Streets of LA (A.1.41)
• Ride to hell retribution (A.1.45)
• Tom Clancy’s: The Division (A.1.46)
• Driver: San Francisco (A.1.47)
• Urban Chaos (A.1.48)
• Morrowind (A.1.49)
• Ultima VII (A.1.50)
• Borderlands (A.1.51)
• The Witcher (A.1.52)
• Star Wars: Knights of the Old Republic (A.1.53)
• Need for speed: most wanted (A.1.54)
• Assassin’s Creed (A.1.56)
• Fable 2 (A.1.57)
• Spiderman: Web of Shadows (A.1.58)
• Red faction guerilla (A.1.61)
• Infamous (A.1.62)
• Dragon’s dogma (A.1.63)
• Tomb Raider (2013) (A.1.64)
• Saints Row 2 (A.1.66)
• GTA: Vice City (A.1.67)
• Yakuza (A.1.68)
• Just cause 2 (A.1.69)
Appendix A. Study of 100 Open World Games

- Saints Row 3 (A.1.70)
- Sleeping Dogs (A.1.71)
- Assassin’s Creed 2 (A.1.74)
- Far Cry 3 (A.1.75)
- Saints Row 4 (A.1.76)
- Final Fantasy VII (A.1.79)
- GTA 4 (A.1.81)
- Assassin’s Creed: Brotherhood (A.1.82)
- Destiny (A.1.83)
- The Witcher 3: Wild Hunt (A.1.84)
- Assassin’s Creed IV: Black Flag (A.1.85)
- The Legend of Zelda: Wind Waker (A.1.88)
- Dragon Age: Origins (A.1.89)
- The Legend of Zelda: Skyward Sword (A.1.90)
- GTA: San Andreas (A.1.91)
- Borderlands 2 (A.1.92)
- Watch Dogs (A.1.93)
- Red Dead Redemption (A.1.95)
- The Witcher 2 (A.1.97)
- Batman: Arkham City (A.1.98)
- The elder scrolls V: Skyrim (A.1.100)
A.2. List Of Different Helpers And Games That Use Them

A.2.11 3D-objects to show to the player that this spot is a way-point/objective/...

- The Simpsons Hit and Run (A.1.16)
- Midnight club Los Angeles (A.1.17)
- Midtown Madness 3 (A.1.18)
- Prototype 2 (A.1.20)
- Bully: scholarship edition (A.1.24)
- Crackdown (A.1.26)
- Saints Row (A.1.31)
- True Crime Streets of LA (A.1.41)
- Tom Clancy’s: The Division (A.1.46)
- Driver: San Francisco (A.1.47)
- Need for speed: most wanted (A.1.54)
- Spiderman: Web of Shadows (A.1.58)
- Infamous 2 (A.1.59)
- The Incredible Hulk Ultimate destruction (A.1.60)
- Red faction guerilla (A.1.61)
- Infamous (A.1.62)
- Tomb Raider (2013) (A.1.64)
- Saints Row 2 (A.1.66)
- GTA: Vice City (A.1.67)
- Saints Row 3 (A.1.70)
- Assassin’s Creed 2 (A.1.74)
- Saints Row 4 (A.1.76)
• Assassin’s Creed: Brotherhood [A.1.82]
• Assassin’s Creed IV: Black Flag [A.1.85]
• GTA: San Andreas [A.1.91]
• Batman: Arkham City [A.1.98]
• GTA V [A.1.99]
A.2.12 Highlighting objects

- Assassin’s Creed: Liberation (A.1.2)
- Don’t Starve (A.1.9)
- Assassin’s Creed: Revelations (A.1.21)
- Crackdown (A.1.26)
- Fallout: New Vegas (A.1.29)
- Kingdoms of Amalur: Reckoning (A.1.30)
- Assassin’s Creed 3 (A.1.35)
- Fallout (A.1.39)
- Assassin’s Creed (A.1.56)
- Fable 2 (A.1.57)
- Infamous 2 (A.1.59)
- Far Cry 2 (A.1.72)
- Assassin’s Creed 2 (A.1.74)
- Fallout 3 (A.1.77)
- The Legend of Zelda: A Link to the past (A.1.78)
- Assassin’s Creed: Brotherhood (A.1.82)
- The Witcher 3: Wild Hunt (A.1.84)
- Assassin’s Creed IV: Black Flag (A.1.85)
- Dragon Age: Origins (A.1.89)
- Borderlands 2 (A.1.92)
- Watch Dogs (A.1.93)
- Metal Gear Solid V: The Phantom Pain (A.1.94)
- The Witcher 2 (A.1.97)
- Batman: Arkham City (A.1.98)
A.2.13 Form/colour-changing cross-hair

- Mafia 2 (A.1.8)
- Mercenaries: Playground of Destruction (A.1.12)
- Freelancer (A.1.13)
- Gun (A.1.19)
- Crackdown (A.1.26)
- Mercenaries 2 (A.1.27)
- Fallout: New Vegas (A.1.29)
- Saints Row (A.1.31)
- STALKER: Shadow of Chernobyl (A.1.36)
- Fallout (A.1.39)
- Tom Clancy’s: The Division (A.1.46)
- Driver: San Francisco (A.1.47)
- Infamous 2 (A.1.62)
- Red faction guerilla (A.1.61)
- Infamous (A.1.62)
- Dragon’s dogma (A.1.63)
- Saints Row 2 (A.1.66)
- Just cause 2 (A.1.69)
- GTA: Vice City (A.1.67)
A.2. List Of Different Helpers And Games That Use Them

- Saints Row 3
- Far Cry 2
- Far Cry 3
- Saints Row 4
- Far Cry
- GTA 4
- Destiny
- The elder scrolls IV: Oblivion
- GTA: San Andreas
- Borderlands 2
- Watch Dogs
- Metal Gear Solid V: The Phantom Pain
- The elder scrolls V: Skyrim

A.2.14 “Holographic” 3D-elements for showing the player certain information (not for indicating a point of interest)

- Assassin’s Creed: Revelations
- Tom Clancy’s: The Division
- Need for speed: most wanted
- Saints Row 3
- Sleeping Dogs
- Saints Row 4
- Assassin’s Creed: Brotherhood
- Borderlands 2
A.2.15 3D-map with augmented information

- The Division (A.1.46)
- Far Cry 2 (A.1.72)
- Minecraft (A.1.73)

A.2.16 Vibration or audio-feedback to tell you you’re “near”

- L.A.: Noire (A.1.55)

A.2.17 On-screen textures to show information like low health

- Infamous 2 (A.1.59)
- Infamous (A.1.62)
- Tomb Raider (2013) (A.1.64)
- Borderlands 2 (A.1.92)
- Metal Gear Solid V (A.1.94)
Appendix B

CODE SNIPPETS

B.1 First Prototype

B.1.1 Creating the Vertices For a Circle or Cone

```java
public float[] generateCircleCoords(float middleX, float middleY, float middleZ, float radius, float yValue, int numPoints) {
    float[] returnCircle = null;
    returnCircle = new float[numPoints*9];
    int circleArrayPos = 0;
    float currentAngle = 0.0f;
    float currentX = (float) Math.cos(degreesToRad(currentAngle))*radius;
    float currentZ = (float) Math.sin(degreesToRad(currentAngle))*radius;
    float prevX = (float) Math.sin(degreesToRad(0.0f))*radius;
    float prevZ = (float) Math.sin(degreesToRad(0.0f))*radius;

    prevX = currentX;
    prevZ = currentZ;
    currentAngle += 360.0f/(float) numPoints;
    currentX = (float) Math.cos(degreesToRad(currentAngle))*radius;
    currentZ = (float) Math.sin(degreesToRad(currentAngle))*radius;

    // Skip the first 2 points for now to make the circle connect better in the end
    for (int i = 0; i < 2; i++) {
        prevX = currentX;
        prevZ = currentZ;
        currentAngle += 360.0f/(float) numPoints;
        if (currentAngle >= 360.0f)
            currentAngle -= 360.0f;
    }

    // When we are full circle, make sure the first and last coords are the same
    if (i == (numPoints - 2)) {
        currentX =
            (float) Math.cos(degreesToRad(360.0f/(float) numPoints))*radius;
        currentZ =
            (float) Math.sin(degreesToRad(360.0f/(float) numPoints))*radius;
    }
    else {
        currentX =
            (float) Math.cos(degreesToRad(currentAngle))*radius;
    }
```
Appendix B. Code Snippets

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.common_ui);
    cardboardView = (CardboardView)
        findViewById(R.id.cardboard_view);
    cardboardView.setRenderer(this);
    setCardboardView(cardboardView);
}
```

B.1.2 Cardboard onCreate

```java
currentZ =
    (float)Math.sin(degreesToRad(currentAngle))*radius;
}
}
// Create a triangle for each point of the circle
for (int i = 0; i < numPoints; i++) {
    returnCircle[circleArrayPos] = prevX;
    returnCircle[circleArrayPos+1] = yValue;
    returnCircle[circleArrayPos+2] = prevZ;
    returnCircle[circleArrayPos+3] = middleX;
    returnCircle[circleArrayPos+4] = middleY;
    returnCircle[circleArrayPos+5] = middleZ;
    returnCircle[circleArrayPos+6] = currentX;
    returnCircle[circleArrayPos+7] = yValue;
    returnCircle[circleArrayPos+8] = currentZ;
}
circleArrayPos += 9;
prevX = currentX;
prevZ = currentZ;
currentAngle += 360.0f/(float)numPoints;
if (currentAngle >= 360.0f)
    currentAngle -= 360.0f;
// When we are full circle, make sure the first and last coords are the same
if (i == (numPoints-2)) {
    currentX =
        (float)Math.cos(degreesToRad(360.0f/(float)numPoints))*radius;
    currentZ =
        (float)Math.sin(degreesToRad(360.0f/(float)numPoints))*radius;
} else {
    currentX =
        (float)Math.cos(degreesToRad(currentAngle))*radius;
    currentZ =
        (float)Math.sin(degreesToRad(currentAngle))*radius;
}
return returnCircle;
```
B.1. First Prototype

// Initialize the transformation matrices
mCamera = new float[16];
mView = new float[16];
mModelViewProjection = new float[16];
mModelView = new float[16];
mModelFloorPlan = new float[16];
...

// Detects a long press to swap textures
gestureDetector = new GestureDetector(this, new
    GestureDetector.SimpleOnGestureListener() {
        public void onLongPress(MotionEvent e) {
            textureNum = (textureNum + 1)% NUMTEXTURES;
            cardboardView.onPause();
            cardboardView.onResume();
        }
    });
// Increase view distance
cardboardView.setZPlanes(0.1f, 5000.0f);

B.1.3 Creating Normals and Texture Coordinates for the Circle or Cone

public float[] generateCircleTextureCoords(int numPoints, boolean isCeiling) {
    float[] returnCircleTexCoords = null;
    returnCircleTexCoords = new float[numPoints*6];
    float topStepSize = 1.0f/(float)numPoints;
    float bottomStepSize = topStepSize / 2.0f;
    int circleTexCoordsPos = 0;
    if (!isCeiling) {
        float currentX = 1.0f - topStepSize;
        float currentY = FLOORWALLSENDERELATIVE;
        // Create a triangle for each point of the circle
        for (int i = 0; i < numPoints; i++) {
            returnCircleTexCoords[circleTexCoordsPos] = currentX;
            returnCircleTexCoords[circleTexCoordsPos + 1] = currentY;
            returnCircleTexCoords[circleTexCoordsPos + 2] = currentX -
                bottomStepSize;
            returnCircleTexCoords[circleTexCoordsPos + 3] = 0.0f;
            returnCircleTexCoords[circleTexCoordsPos + 4] = currentX -
                topStepSize;
            returnCircleTexCoords[circleTexCoordsPos + 5] = currentY;
            circleTexCoordsPos += 6;
            currentX -= topStepSize;
        }
    } else {
        float currentX = 1.0f - topStepSize;
    }
float currentY = 1.0f;

// Create a triangle for each point of the circle
for (int i = 0; i < numPoints; i++) {
    returnCircleTexCoords[circleTexCoordsPos] = currentX;
    returnCircleTexCoords[circleTexCoordsPos + 1] = WALLSCEILINGEDGERELATIVE;
    returnCircleTexCoords[circleTexCoordsPos + 2] = currentX - bottomStepSize;
    returnCircleTexCoords[circleTexCoordsPos + 3] = currentY;
    returnCircleTexCoords[circleTexCoordsPos + 4] = currentX - topStepSize;
    returnCircleTexCoords[circleTexCoordsPos + 5] = WALLSCEILINGEDGERELATIVE;
    circleTexCoordsPos += 6;
    currentX -= topStepSize;
}

return returnCircleTexCoords;

public float[] generateCircleNormals(int numPoints, float yDirection) {
    float[] returnNormals = null;
    returnNormals = new float[numPoints*9];
    int normalsPos = 0;

    // Create a triangle for each point of the circle
    for (int i = 0; i < numPoints; i++) {
        returnNormals[normalsPos] = 0.0f;
        returnNormals[normalsPos+1] = yDirection;
        returnNormals[normalsPos+2] = 0.0f;
        returnNormals[normalsPos+3] = 0.0f;
        returnNormals[normalsPos+4] = yDirection;
        returnNormals[normalsPos+5] = 0.0f;
        returnNormals[normalsPos+6] = 0.0f;
        returnNormals[normalsPos+7] = yDirection;
        returnNormals[normalsPos+8] = 0.0f;
        normalsPos += 9;
    }

    return returnNormals;
}
public float[] generateSphereCoords(float radius, float yValue,
   int numPoints, float height) {

   // We draw the bottom and top in another function
   int circlesForSphere = numPoints - 2;
   float heightBetweenCircles = height/((float)numPoints);
   float radiusIncrementPerCircle =
      radius/((float)(numPoints)/2.0f);
   float upRadius = radiusIncrementPerCircle*2.0f;
   float bottomRadius = radiusIncrementPerCircle;
   float currentHeight = yValue + heightBetweenCircles;

   float[] returnSphere = null;
   returnSphere = new float[numPoints*9*2*circlesForSphere];
   int sphereArrayPos = 0;
   boolean halfwayPassed = false;

   for (int circles = 0; circles < circlesForSphere; circles++) {
      // Variables for going around 2 circles by using degrees
      float currentAngle = 0.0f;
      float upCurrentX =
         (float)Math.cos(degreesToRad(currentAngle))*upRadius;
      float upCurrentZ =
         (float)Math.sin(degreesToRad(currentAngle))*upRadius;
      float upPrevX = upCurrentX;
      float upPrevZ = upCurrentZ;
      float bottomCurrentX =
         (float)Math.cos(degreesToRad(currentAngle+360.0f/(float)numPoints/2.0f))*bottomRadius;
      float bottomCurrentZ =
         (float)Math.sin(degreesToRad(currentAngle+360.0f/(float)numPoints/2.0f))*bottomRadius;
      float bottomPrevX = bottomCurrentX;
      float bottomPrevZ = bottomCurrentZ;

      if (circles%2 != 0) {
         currentAngle += 360.0f/(float)numPoints;
         currentAngle += (360.0f/(float)numPoints)/2.0f;
      }
      else
         currentAngle += 360.0f/(float)numPoints;

      upCurrentX =
         (float)Math.cos(degreesToRad(currentAngle))*upRadius;
      upCurrentZ =
         (float)Math.sin(degreesToRad(currentAngle))*upRadius;
      bottomCurrentX = (float)Math.cos(degreesToRad(currentAngle)+
         (360.0f/(float)numPoints/2.0f))*bottomRadius;
      bottomCurrentZ = (float)Math.sin(degreesToRad(currentAngle)+
         (360.0f/(float)numPoints/2.0f))*bottomRadius;

      // Start a few degrees ahead of 0.0 to make the sphere connect better
      for (int i = 0; i < 2; i++) {
         upPrevX = upCurrentX;
         upPrevZ = upCurrentZ;
         bottomPrevX = bottomCurrentX;
         bottomPrevZ = bottomCurrentZ;
currentAngle += 360.0f / (float) numPoints;
if (currentAngle >= 360.0f)
    currentAngle -= 360.0f;

upCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * upRadius;
upCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * upRadius;
bottomCurrentX = (float) Math.cos(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
bottomCurrentZ = (float) Math.sin(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
}

// Create a triangle for each point of the current band between two circles
for (int i = 0; i < numPoints; i++) {
    returnSphere[sphereArrayPos] = upPrevX;
    returnSphere[sphereArrayPos + 1] = currentHeight;
    returnSphere[sphereArrayPos + 2] = upPrevZ;
    returnSphere[sphereArrayPos + 3] = bottomPrevX;
    returnSphere[sphereArrayPos + 4] = currentHeight - heightBetweenCircles;
    returnSphere[sphereArrayPos + 5] = bottomPrevZ;
    returnSphere[sphereArrayPos + 6] = upCurrentX;
    returnSphere[sphereArrayPos + 7] = currentHeight;
    returnSphere[sphereArrayPos + 8] = upCurrentZ;
    returnSphere[sphereArrayPos + 9] = bottomPrevX;
    returnSphere[sphereArrayPos + 10] = currentHeight - heightBetweenCircles;
    returnSphere[sphereArrayPos + 11] = bottomPrevZ;
    returnSphere[sphereArrayPos + 12] = bottomCurrentX;
    returnSphere[sphereArrayPos + 13] = currentHeight - heightBetweenCircles;
    returnSphere[sphereArrayPos + 14] = bottomCurrentZ;
    returnSphere[sphereArrayPos + 15] = upCurrentX;
    returnSphere[sphereArrayPos + 16] = currentHeight;
    returnSphere[sphereArrayPos + 17] = upCurrentZ;
    sphereArrayPos += 18;
    upPrevX = upCurrentX;
    upPrevZ = upCurrentZ;
    bottomPrevX = bottomCurrentX;
    bottomPrevZ = bottomCurrentZ;
    currentAngle += 360.0f / (float) numPoints;
    if (currentAngle >= 360.0f)
        currentAngle -= 360.0f;
upCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * upRadius;
upCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * upRadius;
bottomCurrentX = (float) Math.cos(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
bottomCurrentZ = (float) Math.sin(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
}

// Next band
currentHeight += heightBetweenCircles;

// Set both radiuses for next band if we are passed halfway, the top circle should be getting smaller than the bottom one
if (circles > (circlesForSphere/2)) {
    upRadius -= radiusIncrementPerCircle;
    if (!halfwayPassed) {
        bottomRadius += radiusIncrementPerCircle;
        halfwayPassed = true;
    }
    else
        bottomRadius -= radiusIncrementPerCircle;
} else {
    upRadius += radiusIncrementPerCircle;
    bottomRadius += radiusIncrementPerCircle;
}
return returnSphere;

public float[] generateSphereTextureCoords(int numPoints) {
    float[] returnSphereTexCoords = null;
    int circlesForSphere = numPoints - 2;
    returnSphereTexCoords = new float[numPoints*6*2*circlesForSphere];
    float topStepSize = 1.0f/(float)numPoints;
    float bottomStepSize = topStepSize/2.0f;
    int sphereTexCoordsPos = 0;
    float heightBetweenCircles = 1.0f/numPoints;
    float currentY = heightBetweenCircles;
    for (int circle = 0; circle < circlesForSphere; circle++) {
        float currentX = 1.0f - topStepSize;
        if (circle%2 != 0)
            currentX -= topStepSize/2.0f;
        // Create a triangle for each point of the circle
        for (int i = 0; i < numPoints; i++) {
            returnSphereTexCoords[sphereTexCoordsPos] = currentX;
            returnSphereTexCoords[sphereTexCoordsPos + 1] = currentY;
        }
    }
returnSphereTexCoords[sphereTexCoordsPos + 2] = currentX - bottomStepSize;
returnSphereTexCoords[sphereTexCoordsPos + 3] = currentY - heightBetweenCircles;
returnSphereTexCoords[sphereTexCoordsPos + 4] = currentX - topStepSize;
returnSphereTexCoords[sphereTexCoordsPos + 5] = currentY;
returnSphereTexCoords[sphereTexCoordsPos + 6] = currentX - bottomStepSize;
returnSphereTexCoords[sphereTexCoordsPos + 7] = currentY - heightBetweenCircles;
returnSphereTexCoords[sphereTexCoordsPos + 8] = currentX - topStepSize - bottomStepSize;
returnSphereTexCoords[sphereTexCoordsPos + 9] = currentY - heightBetweenCircles;

sphereTexCoordsPos += 12;
currentX -= topStepSize;
}
}
currentY += heightBetweenCircles;
}
return returnSphereTexCoords;

public float[] generateSphereNormals(float middleX, float middleZ, float middleY, float radius, float yValue, int numPoints, float height) {
float[] returnNormals = null;
int circlesForSphere = numPoints - 2;
returnNormals = new float[numPoints*9*2*circlesForSphere];
int normalsPos = 0;

// We draw the bottom and top in another function
float heightBetweenCircles = height/((float)circlesForSphere);
float radiusIncrementPerCircle = radius/((float)(numPoints)/2.0f);
float upRadius = radiusIncrementPerCircle;
float bottomRadius = upRadius + radiusIncrementPerCircle;
float currentHeight = yValue + heightBetweenCircles;
for (int circles = 0; circles < circlesForSphere; circles++) {
    // Variables for going around 2 circles by using degrees
    float currentAngle = 360.0f/(float)numPoints;
```java
float upCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * upRadius;
float upCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * upRadius;
float upPrevX = upCurrentX;
float upPrevZ = upCurrentZ;
float bottomCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * bottomRadius;
float bottomCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * bottomRadius;
float bottomPrevX = bottomCurrentX;
float bottomPrevZ = bottomCurrentZ;
boolean halfwayPassed = false;

currentAngle += 360.0f / (float) numPoints;
upCurrentX = (float) Math.cos(degreesToRad(360.0f / (float) numPoints)) * upRadius;
upCurrentZ = (float) Math.sin(degreesToRad(360.0f / (float) numPoints)) * upRadius;
bottomCurrentX = (float) Math.cos(degreesToRad(360.0f / (float) numPoints)) * bottomRadius;
bottomCurrentZ = (float) Math.sin(degreesToRad(360.0f / (float) numPoints)) * bottomRadius;

// Create a triangle for each point of the current band between two circles
for (int i = 0; i < numPoints; i++) {
    returnNormals[normalsPos] = middleX - upPrevX;
    returnNormals[normalsPos + 1] = middleY - currentHeight;
    returnNormals[normalsPos + 2] = middleZ - upPrevZ;
    returnNormals[normalsPos + 3] = middleX - bottomPrevX;
    returnNormals[normalsPos + 4] = middleY - currentHeight - heightBetweenCircles;
    returnNormals[normalsPos + 5] = middleZ - bottomPrevZ;
    returnNormals[normalsPos + 6] = middleX - upCurrentX;
    returnNormals[normalsPos + 7] = middleY - currentHeight;
    returnNormals[normalsPos + 8] = middleZ - upCurrentZ;
    returnNormals[normalsPos + 9] = middleX - bottomPrevX;
    returnNormals[normalsPos + 10] = middleY - currentHeight - heightBetweenCircles;
    returnNormals[normalsPos + 11] = middleZ - bottomPrevZ;
    returnNormals[normalsPos + 12] = middleX - bottomCurrentX;
    returnNormals[normalsPos + 13] = middleY - currentHeight - heightBetweenCircles;
    returnNormals[normalsPos + 14] = middleZ - bottomCurrentZ;
    returnNormals[normalsPos + 15] = middleX - upCurrentX;
    returnNormals[normalsPos + 16] = middleY - currentHeight;
    returnNormals[normalsPos + 17] = middleZ - upCurrentZ;
    normalsPos += 18;
    upPrevX = upCurrentX;
```
Appendix B. Code Snippets

upPrevZ = upCurrentZ;
bottomPrevX = bottomCurrentX;
bottomPrevZ = bottomCurrentZ;

currentAngle += 360.0f / (float) numPoints;
if (currentAngle >= 360.0f)
currentAngle -= 360.0f;

// When we are full circle, make sure the first and last coords are the same
if (i == (numPoints - 2)) {
upCurrentX = (float) Math.cos(degreesToRad(360.0f / (float) numPoints)) * upRadius;
upCurrentZ = (float) Math.sin(degreesToRad(360.0f / (float) numPoints)) * upRadius;
bottomCurrentX = (float) Math.cos(degreesToRad(360.0f / (float) numPoints)) * bottomRadius;
bottomCurrentZ = (float) Math.sin(degreesToRad(360.0f / (float) numPoints)) * bottomRadius;
} else {
upCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * upRadius;
upCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * upRadius;
bottomCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * bottomRadius;
bottomCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * bottomRadius;
}
}
currentHeight += heightBetweenCircles;
// Set both radiuses for next band
if (circles > (circlesForSphere/2)) {
if (!halfwayPassed) {
bottomRadius += radiusIncrementPerCircle;
halfwayPassed = true;
}
else {
bottomRadius -= radiusIncrementPerCircle;
}
else {
upRadius += radiusIncrementPerCircle;
bottomRadius += radiusIncrementPerCircle;
}
}
return returnNormals;

B.1.6 On Surface Created Part 1

public void onSurfaceCreated(EGLConfig config) {
// Dark background
GLES20.glClearColor(0.1f, 0.1f, 0.1f, 0.5f);
Generate a cone and the information needed to draw and texture it to fill the bottom hole in the sphere
float [] groundCoords = generateCircleCoords(0.0f, (-1.0f * RADIUS), 0.0f, RADIUS/CIRCLEPOINTS/2.0f, (-1.0f * RADIUS)*RADIUS/CIRCLEPOINTS*6, CIRCLEPOINTS);
float [] groundColors = generateCircleColors(CIRCLEPOINTS);
float [] groundTexCoords = generateCircleTextureCoords(CIRCLEPOINTS, false);
float [] groundNormals = generateCircleNormals(CIRCLEPOINTS, 1.0f);

ByteBuffer bbGroundVertices = ByteBuffer.allocateDirect(groundCoords.length * 4);
bbGroundVertices.order(ByteOrder.nativeOrder());
mGroundVertices = bbGroundVertices.asFloatBuffer();
mGroundVertices.put(groundCoords);
mGroundVertices.position(0);

ByteBuffer bbGroundColors = ByteBuffer.allocateDirect(groundColors.length * 4);
...  
ByteBuffer bbGroundTextureCoords = ByteBuffer.allocateDirect(groundTexCoords.length * 4);
...  
ByteBuffer bbGroundNormals = ByteBuffer.allocateDirect(groundNormals.length * 4);
...  
// Generate a cone and the information needed to draw and texture it to fill the top hole of the sphere

float [] wallsCoords = generateSphereCoords(RADIUS, (-1.0f * RADIUS + RADIUS*2.0f/CIRCLEPOINTS), CIRCLEPOINTS, 20.0f-2.0f*RADIUS*2/CIRCLEPOINTS);
float [] wallsColors = generateSphereColors(CIRCLEPOINTS);
float [] wallsTexCoords = generateSphereTextureCoords(CIRCLEPOINTS);
float [] wallsNormals = generateSphereNormals(0.0f, 0.0f, 0.0f, RADIUS, (-1.0f * RADIUS), CIRCLEPOINTS, 20.0f-2.0f*RADIUS*2/CIRCLEPOINTS);

ByteBuffer bbWallsVertices = ByteBuffer.allocateDirect(wallsCoords.length * 4);
bbWallsVertices.order(ByteOrder.nativeOrder());
mWallsVertices = bbWallsVertices.asFloatBuffer();
mWallsVertices.put(wallsCoords);
mWallsVertices.position(0);

ByteBuffer bbWallsColors = ByteBuffer.allocateDirect(wallsColors.length * 4);
...  
ByteBuffer bbWallsNormals = ByteBuffer.allocateDirect(wallsNormals.length * 4);
...  
ByteBuffer bbWallsTextureCoords = ByteBuffer.allocateDirect(wallsTexCoords.length * 4);
B.1.7 On Surface Created Part 2

```java
// Lines for the floor-plan
LatLonToPoint latLonToGallPeeters = new LatLonToPoint();
float [] GallPeetersCoords = latLonToGallPeeters.coordsArrayToGallPeeters
     (SURROUNDMAPDATA.EDMANDROMEMAP_COORDS);
ByteBuffer bbLinesSurroundMapVertices =
     ByteBuffer.allocateDirect(GallPeetersCoords.length * 4);
bbLinesSurroundMapVertices.order(ByteOrder.nativeOrder());
mLinesSurroundMapVertices =
     bbLinesSurroundMapVertices.asFloatBuffer();
float [] tempEDMMapCoordsBuffer =
     remapCoordsAroundCenterPos(GallPeetersCoords, myPointFloat);
int numColors = (SURROUNDMAPDATA.EDMANDROMEMAP_COORDS.length/3)*4;
numLineEdges = numColors/4;
ByteBuffer bbLinesSurroundMapColors =
     ByteBuffer.allocateDirect(numColors * 4);
... float [] colorsForLines = new float[numColors];
for (int i = 0; i < colorsForLines.length; i++) {
    ...
    mLinesSurroundMapColors.put(colorsForLines);
    mLinesSurroundMapColors.position(0);
}
float [] normalsForLines = new float[SURROUNDMAPDATA.EDMANDROMEMAP_COORDS.length];
for (int i = 0; i < normalsForLines.length; i++) {
}
ByteBuffer bbLinesSurroundMapNormals =
     ByteBuffer.allocateDirect(normalsForLines.length * 4);
... int vertexShader = loadGLShader(GLES20.GL_VERTEX_SHADER,
     R.raw.light_vertex);
int gridShader = loadGLShader(GLES20.GL_FRAGMENT_SHADER,
     R.raw.grid_fragment);
GLES20.glAttachShader(mGlProgram, vertexShader);
GLES20.glAttachShader(mGlProgram, gridShader);
GLES20.glLinkProgram(mGlProgram);
GLES20.glEnable(GLES20.GL_DEPTH_TEST);
Matrix.setIdentityM(mModelSurroundMap, 0);
Matrix.translateM(mModelSurroundMap, 0, 0, 3, 0);
... // Set up textures
setNextTexture();
```

B.1.8 Transform Google Maps Coordinates To X/Y Coordinates
// Code based on: https://developers.google.com/maps/documentation/javascript/examples/map-projection-simple

public class LatLonToPoint {
    // Note: this value is exact as the map projects a full 360 degrees of longitude
    private final int GALL_PETERS_RANGE_X = 800;
    // Note: this value is inexact as the map is cut off at ~ +/- 83 degrees.
    // However, the polar regions produce very little increase in Y range, so
    // we will use the tile size.
    private final int GALL_PETERS_RANGE_Y = 510;
    private static double worldCoordinatePerLonDegree_ = 0;
    private static double worldCoordinateLatRange = 0;
    private static MyPoint worldOrigin;

    LatLonToPoint() {
        // This projection has equidistant meridians, so each longitude degree is a linear mapping.
        worldCoordinatePerLonDegree_ = GALL_PETERS_RANGE_X / 360;
        // This constant merely reflects that latitudes vary from +90 to -90 degrees.
        worldCoordinateLatRange = GALL_PETERS_RANGE_Y / 2;
        worldOrigin = new MyPoint(GALL_PETERS_RANGE_X * 400 / 800,
                                   GALL_PETERS_RANGE_Y / 2);
    }

    static double degreesToRadians(double deg) {
        return deg * (Math.PI / 180);
    }

    double radiansToDegrees(double rad) {
        return rad / (Math.PI / 180);
    }

    public MyPoint LatLonToPointF(double lat, double lon) {
        double xOrigin = worldOrigin.lat;
        double yOrigin = worldOrigin.lon;

        double x = xOrigin + worldCoordinatePerLonDegree_ * lon;
        // Note that latitude is measured from the world coordinate origin
        // at the top left of the map.
        double latRadians = degreesToRadians(lat);
        double y = xOrigin - worldCoordinateLatRange * Math.sin(latRadians);

        MyPoint rPoint = new MyPoint(x, y);
        return (rPoint);
    }
}
// Takes an array of lon, some y-value, lat and converts it to Gall_peeters
public float[] coordsArrayToGallPeeters(float[] coords) {
    float[] gallPeetersArray = null;

    gallPeetersArray = new float[coords.length];
    for (int i = 0; i < coords.length; i+=3) {
        float lon = coords[i];
        float lat = coords[i+2];
        MyPoint currentPoint = LatLonToPointF(lat, lon);
        gallPeetersArray[i] = (float)currentPoint.lon;
        gallPeetersArray[i+2] = (float)currentPoint.lat;
    }

    return gallPeetersArray;
}

public static class MyPoint {
    public double lat;
    public double lon;
    MyPoint(double lat, double lon) {
        this.lat = lat;
        this.lon = lon;
    }
}

B.1.9 Set New Center Position

public void setNewCenterPos(float[] center) {
    LatLonToPoint latLonToGallPeeters = new LatLonToPoint();
    float[] GallPeetersCoords =
        latLonToGallPeeters.coordsArrayToGallPeeters(SURROUNDMAPDATA.EDMANDROMEMAP_COORDS);
    ByteBuffer bbLinesSurroundMapVertices =
        ByteBuffer.allocateDirect(GallPeetersCoords.length * 4);
    bbLinesSurroundMapVertices.order(ByteOrder.nativeOrder());
    mLinesSurroundMapVertices.clear();
    mLinesSurroundMapVertices =
        bbLinesSurroundMapVertices.asFloatBuffer();

    float[] tempEDMMapCoordsBuffer =
        remapSurroundMapWithGeoreferences(GallPeetersCoords,
            SURROUNDMAPDATA.EDMANDROMEMAP_COORDS,
            SURROUNDMAPDATA.EDMSTREETVIEWIMAGE1GEOREFERENCE,
            myPointFloat, 2560, 1280);
    mLinesSurroundMapVertices.put(tempEDMMapCoordsBuffer);
    mLinesSurroundMapVertices.position(0);
}

B.1.10 Find OpenGL Position Based On Texture Coordinate
public float[] getOpenGLPosFromTextureCoords(int x, int y, int
   imageWidth, int imageHeight, int numPoints, float height,
   float radius, float yValue) {
    float[] openGLPos = {0.0f, 0.0f, 0.0f};

    // Texture coordinates of the position on the image
    // We need to find out where on the sphere this is located
    float textureX = (float)(x)/(float)(imageWidth);
    float textureY = (float)(y)/(float)(imageHeight);

    int circlesForSphere = numPoints - 2;

    float topStepSize = 1.0f/(float)numPoints;
    float bottomStepSize = topStepSize/2.0f;
    int sphereTexCoordsPos = 0;

    float heightBetweenTextureCircles = 1.0f/numPoints;
    float currentTextureY = heightBetweenTextureCircles;

    float heightBetweenCircles = height/((float)numPoints);
    float radiusIncrementPerCircle =
        radius/((float)(numPoints)/2.0f);
    float upRadius = radiusIncrementPerCircle*2.0f;
    float bottomRadius = radiusIncrementPerCircle;
    float currentHeight = yValue + heightBetweenCircles;

    boolean halfwayPassed = false;
    float tempWorldX = 0.0f;
    float tempWorldY = 0.0f;
    float tempWorldZ = 0.0f;

    float tempTextureX = 0.0f;
    float tempTextureY = 0.0f;

    boolean positionFound = false;
    float currentFoundTextureY = 1.0f;
    float currentFoundTextureX = 1.0f;
    float currentFoundDistance =
        (float)Math.sqrt((currentFoundTextureX-0.0f)*(currentFoundTextureX-0.0f) +
        (currentFoundTextureY-0.0f)*(currentFoundTextureY-0.0f));
    float currentDistance = 0.0f;

    for (int circles = 0; circles < circlesForSphere &&
        !positionFound; circles++) {
      // Variables for going around 2 circles by using degrees
      float currentAngle = 0.0f;
      float upCurrentX =
          (float)Math.cos(degreesToRad(currentAngle))*upRadius;
      float upCurrentZ =
          (float)Math.sin(degreesToRad(currentAngle))*upRadius;
      float upPrevX = upCurrentX;
      float upPrevZ = upCurrentZ;
      float bottomCurrentX =
          (float)Math.cos(degreesToRad(currentAngle))*bottomRadius;
      float bottomCurrentZ =
          (float)Math.sin(degreesToRad(currentAngle))*bottomRadius;
```java
float bottomPrevX = bottomCurrentX;
float bottomPrevZ = bottomCurrentZ;

if (circles % 2 != 0) {
    currentAngle += 360.0f/(float)numPoints;
    currentAngle += (360.0f/(float)numPoints)/2.0f;
} else {
    currentAngle += 360.0f/(float)numPoints;
}

upCurrentX = (float)Math.cos(degreesToRad(currentAngle))*upRadius;
upCurrentZ = (float)Math.sin(degreesToRad(currentAngle))*upRadius;
bottomCurrentX = (float)Math.cos(degreesToRad((currentAngle+(360.0f/(float)numPoints)/2.0f)))*bottomRadius;
bottomCurrentZ = (float)Math.sin(degreesToRad((currentAngle+(360.0f/(float)numPoints)/2.0f)))*bottomRadius;

float currentTextureX = 1.0f - topStepSize;
// Every other circle of the sphere, it has to be slightly
// rotated a bit to get better triangles
if (circles % 2 != 0) currentTextureX -= topStepSize/2.0f;

// Start a few degrees ahead of 0.0 to make the sphere connect
// better
for (int i = 0; i < 2; i++) {
    upPrevX = upCurrentX;
    upPrevZ = upCurrentZ;
    bottomPrevX = bottomCurrentX;
    bottomPrevZ = bottomCurrentZ;

    upCurrentX = (float)Math.cos(degreesToRad(currentAngle)) * upRadius;
    upCurrentZ = (float)Math.sin(degreesToRad(currentAngle)) * upRadius;
    bottomCurrentX = (float)Math.cos(degreesToRad((currentAngle+(360.0f/(float)numPoints)/2.0f)))*bottomRadius;
    bottomCurrentZ = (float)Math.sin(degreesToRad((currentAngle+(360.0f/(float)numPoints)/2.0f)))*bottomRadius;
    currentTextureX -= topStepSize;
}

// Create a triangle for each point of the current band between
two circles
for (int i = 0; i < numPoints&&!positionFound; i++) {
    tempWorldX = upPrevX;
    tempWorldY = currentHeight;
    tempWorldZ = upPrevZ;

    tempTextureX = currentTextureX;
    tempTextureY = currentTextureY;
    currentDistance =
        (float)Math.sqrt((tempTextureX-textureX)*(tempTextureX-textureX)
                        + (tempTextureY-textureY)*(tempTextureY-textureY));
```

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
    openGLPos[0] = tempWorldX;
    openGLPos[1] = tempWorldY;
    openGLPos[2] = tempWorldZ;
    currentFoundTextureX = tempTextureX;
    currentFoundTextureY = tempTextureY;
    currentFoundDistance = currentDistance;
}

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
    openGLPos[0] = tempWorldX;
    openGLPos[1] = tempWorldY;
    openGLPos[2] = tempWorldZ;
    currentFoundTextureX = tempTextureX;
    currentFoundTextureY = tempTextureY;
    currentFoundDistance = currentDistance;
}

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
    openGLPos[0] = tempWorldX;
    openGLPos[1] = tempWorldY;
    openGLPos[2] = tempWorldZ;
    currentFoundTextureX = tempTextureX;
    currentFoundTextureY = tempTextureY;
    currentFoundDistance = currentDistance;
}
currentDistance =
  (float)Math.sqrt((tempTextureX-textureX)*(tempTextureX-textureX)
  + (tempTextureY-textureY)*(tempTextureY-textureY));

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
  openGLPos[0] = tempWorldX;
  openGLPos[1] = tempWorldY;
  openGLPos[2] = tempWorldZ;
  currentFoundTextureX = tempTextureX;
  currentFoundTextureY = tempTextureY;
  currentFoundDistance = currentDistance;
}

tempWorldX = bottomCurrentX;
tempWorldY = currentHeight - heightBetweenCircles;
tempWorldZ = bottomCurrentZ;

tempTextureX = currentTextureX - topStepSize -
    bottomStepSize;
tempTextureY = currentTextureY - heightBetweenTextureCircles;
currentDistance =
  (float)Math.sqrt((tempTextureX-textureX)*(tempTextureX-textureX)
  + (tempTextureY-textureY)*(tempTextureY-textureY));

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
  openGLPos[0] = tempWorldX;
  openGLPos[1] = tempWorldY;
  openGLPos[2] = tempWorldZ;
  currentFoundTextureX = tempTextureX;
  currentFoundTextureY = tempTextureY;
  currentFoundDistance = currentDistance;
}

tempWorldX = upCurrentX;
tempWorldY = currentHeight;
tempWorldZ = upCurrentZ;

tempTextureX = currentTextureX - topStepSize;
tempTextureY = currentTextureY;
currentDistance =
  (float)Math.sqrt((tempTextureX-textureX)*(tempTextureX-textureX)
  + (tempTextureY-textureY)*(tempTextureY-textureY));

if (tempTextureX >= textureX && tempTextureY >= textureY &&
    currentDistance < currentFoundDistance) {
  openGLPos[0] = tempWorldX;
  openGLPos[1] = tempWorldY;
  openGLPos[2] = tempWorldZ;
  currentFoundTextureX = tempTextureX;
  currentFoundTextureY = tempTextureY;
  currentFoundDistance = currentDistance;
}

sphereTexCoordsPos += 12;
currentTextureX -= topStepSize;
B.1. First Prototype

```
   upPrevX = upCurrentX;
   upPrevZ = upCurrentZ;
   bottomPrevX = bottomCurrentX;
   bottomPrevZ = bottomCurrentZ;

   currentAngle += 360.0f / (float) numPoints;
   if (currentAngle >= 360.0f)
       currentAngle -= 360.0f;

   upCurrentX = (float) Math.cos(degreesToRad(currentAngle)) * upRadius;
   upCurrentZ = (float) Math.sin(degreesToRad(currentAngle)) * upRadius;
   bottomCurrentX = (float) Math.cos(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
   bottomCurrentZ = (float) Math.sin(degreesToRad(currentAngle + (360.0f/(float) numPoints)/2.0f)) * bottomRadius;
}

// Next band
currentHeight += heightBetweenCircles;
currentTextureY += heightBetweenTextureCircles;

// Set both radiiuses for next band if we are passed halfway, the top circle should be getting smaller than the bottom one
if (circles > (circlesForSphere /2)) {
    upRadius -= radiusIncrementPerCircle;
    if (!halfwayPassed) {
        bottomRadius += radiusIncrementPerCircle;
        halfwayPassed = true;
    }
    else {
        bottomRadius -= radiusIncrementPerCircle;
    }
} else {
    upRadius += radiusIncrementPerCircle;
    bottomRadius += radiusIncrementPerCircle;
}
return openGLPos;
```

B.1.11 Set Next Texture

```
public void setNextTexture() {
    // Swap textures
    if (textureNum == 0) {
        // resourceId = R.drawable.edm2;
        if (surroundLinesHeight != 0.0f) {
            float translationAmount = 0.0f - surroundLinesHeight;
            Matrix.translateM(mModelLines, 0, 0, translationAmount, 0);
            surroundLinesHeight = 0.0f;
        }
        if (surroundLinesAngle != 44f) {
            float rotationAmount = 44f - surroundLinesAngle;
            Matrix.rotateM(mModelLines, 0, rotationAmount, 0, 1.0f, 0);
            surroundLinesAngle = 44f;
        }
    }
    ```
14 }
15 MAPSIZE = 550;
16 resourceId = R.drawable.edm2;
17 setNewCenterPos(SURROUNDMAPDATA.EDM2USERPOS);
18 }
19 else if (textureNum == 1) {
20 ... 
}
21 }
22 ... 
23 mTextureDataHandle = loadTexture(this, resourceId);
24 checkGLError("" + GLES20.GL_MAX_TEXTURE_SIZE);
25 }

B.1.12 Remap Floor Plan With Georeferences

```java
public float[] remapSurroundMapWithGeoreferences(float[] surroundMapCoords, float[] originalMapCoords, float[] georeferences, float[] center, int imgWidth, int imgHeight) {
    float[] newSurroundMap = null;
    int numGeoreferences = 0;
    int[] georeferencesInUse = null;

    for (int i = 0; i < originalMapCoords.length && georeferences != null; i += 3) {
        float currentX = originalMapCoords[i];
        float currentY = originalMapCoords[i + 1];
        float currentZ = originalMapCoords[i + 2];
        float[] currentSetOfCoords = new float[3];
        currentSetOfCoords[0] = currentX;
        currentSetOfCoords[1] = currentY;

        int position = geoReferencesContainPosition(georeferences, currentSetOfCoords);

        if (position != -1) {
            numGeoreferences++;
            if (georeferencesInUse == null) {
                georeferencesInUse = new int[2];
                georeferencesInUse[0] = i;
                //The point after this one is the same coordinate
                if (i+3 < surroundMapCoords.length)
                    georeferencesInUse[1] = i+3;
                else
                    georeferencesInUse[1] = 0;
            } else {
                int[] tempReferenceBuffer = new int[georeferencesInUse.length+2];
                tempReferenceBuffer[georeferencesInUse.length] = i;
                if (i+3 < surroundMapCoords.length)
                    tempReferenceBuffer[georeferencesInUse.length+1] = i+3;
                else
                    tempReferenceBuffer[georeferencesInUse.length+1] = 0;
                for (int j = 0; j < georeferencesInUse.length; j++)
                    tempReferenceBuffer[j] = georeferencesInUse[j];
```
georeferencesInUse = new int[georeferencesInUse.length+2];
for (int j = 0; j < georeferencesInUse.length; j++)
    georeferencesInUse[j] = tempReferenceBuffer[j];

float textureX = georeferences[position+2];
float textureY = georeferences[position+3];

float[] position3D =
    getOpenGLPosFromTextureCoords((int)textureX,
        (int)textureY, imgWidth, imgHeight, CIRCLEPOINTS,
        20.0f-2.0f*RADIUS*2/CIRCLEPOINTS, RADIUS, (-1.0f*
            RADIUS + RADIUS*2.0f/CIRCLEPOINTS));

// Remap all the coordinates using the information of the
// first point
if (numGeoreferences == 1) {
    float[] shiftedSurroundMap =
        remapCoordsAroundCenterPos(surroundMapCoords, center);
    float verschilX = position3D[0] - shiftedSurroundMap[i];
    float verschilY = position3D[1] - shiftedSurroundMap[i + 1];
    float verschilZ = position3D[2] - shiftedSurroundMap[i + 2];
    newSurroundMap = new float[shiftedSurroundMap.length];
    for (int j = 0; j + 2 < shiftedSurroundMap.length; j += 3) {
        newSurroundMap[j] = shiftedSurroundMap[j] + verschilX;
        newSurroundMap[j + 1] = shiftedSurroundMap[j + 1] +
            verschilY;
        newSurroundMap[j + 2] = shiftedSurroundMap[j + 2] +
            verschilZ;
    }
    i += 3;
}
// Rotate around the first found point and scale until the
// map fits on the second reference
//3 because all coordinates are listed twice
else if (numGeoreferences == 2) {
    float[] firstRefPosition3D =
        {newSurroundMap[georeferencesInUse[0]],
            newSurroundMap[georeferencesInUse[0]+1],
            newSurroundMap[georeferencesInUse[0]+2]};

    // Rotate and scale the map while keeping the position of
    // this point
    float[] secondRefPosition3D =
        getOpenGLPosFromTextureCoords((int)textureX,
            (int)textureY, imgWidth, imgHeight, CIRCLEPOINTS,
            20.0f-2.0f*RADIUS*2/CIRCLEPOINTS, RADIUS, (-1.0f*
                RADIUS + RADIUS*2.0f/CIRCLEPOINTS));
    float[] currentSecondRefPosition3D = {newSurroundMap[i],
        newSurroundMap[i+1], newSurroundMap[i+2]};
// We get the vector as it is now, between the first
// referenced point and the second
// After that we calculate the vector between the first
// referenced point and the new position of the second
// point
// Then we apply this rotation and scaling to all the
// points of the map
// Vector A
float vectorX = currentSecondRefPosition3D[0] -
    firstRefPosition3D[0];
float vectorY = currentSecondRefPosition3D[1] -
    firstRefPosition3D[1];
float vectorZ = currentSecondRefPosition3D[2] -
    firstRefPosition3D[2];
float [] vectorOldPos = {vectorX, vectorY, vectorZ};
float currentDist =
    distBetweenPoints(currentSecondRefPosition3D, firstRefPosition3D);
float newDist = distBetweenPoints(secondRefPosition3D, firstRefPosition3D);
float distFactor = newDist / currentDist;
float magnitudeOldVector =
    (float)Math.sqrt(Math.pow(vectorOldPos[0], 2) +
        Math.pow(vectorOldPos[1], 2) +
        Math.pow(vectorOldPos[2], 2));

// Vector B
vectorX = secondRefPosition3D[0] - firstRefPosition3D[0];
vectorY = secondRefPosition3D[1] - firstRefPosition3D[1];
float [] vectorNewPos = {vectorX, vectorY, vectorZ};
float magnitudeNewVector =
    (float)Math.sqrt(Math.pow(vectorNewPos[0], 2) +
        Math.pow(vectorNewPos[1], 2) +
        Math.pow(vectorNewPos[2], 2));
float dotProduct = vectorOldPos[0] * vectorNewPos[0] +
    vectorNewPos[2];
float angle =
    (float)Math.acos((dotProduct)/(magnitudeOldVector *
        magnitudeNewVector));
// http://www.wikihow.com/Find-the-Angle-Between-Two-Vectors
// C = AxB = (a_y b_z - a_z b_y , a_z b_x - a_x b_z, a_x
// b_y - a_y b_x),
float [] crossProduct1 = new float[3];
crossProduct1[0] = vectorOldPos[1]*vectorNewPos[2] -
    vectorOldPos[2]*vectorNewPos[1];
crossProduct1[1] = vectorOldPos[2]*vectorNewPos[0] -
    vectorOldPos[0]*vectorNewPos[2];
crossProduct1[2] = vectorOldPos[0]*vectorNewPos[1] -
    vectorOldPos[1]*vectorNewPos[0];
float crossProduct1Magnitude =
    (float)Math.sqrt(Math.pow(crossProduct1[0], 2) +
Math.pow(crossProduct1[1], 2) +
Math.pow(crossProduct1[2], 2));

// Normalize it
crossProduct1[0] /= crossProduct1Magnitude;
crossProduct1[1] /= crossProduct1Magnitude;
crossProduct1[2] /= crossProduct1Magnitude;

// F = CxA
float[] crossProduct2 = new float[3];
crossProduct2[0] = crossProduct1[1]*vectorOldPos[2] -
crossProduct1[2]*vectorOldPos[1];
crossProduct2[1] = crossProduct1[2]*vectorOldPos[0] -
crossProduct1[0]*vectorOldPos[2];
crossProduct2[2] = crossProduct1[0]*vectorOldPos[1] -
crossProduct1[1]*vectorOldPos[0];

// The new position is calculated by now rotating the old
// vector and changing the distance
// G = cos(angle) A + sin(angle) F
float[] rotatedVector = new float[3];
rotatedVector[0] =
(float)(Math.cos(angle)*vectorOldPos[0] +
Math.sin(angle)*crossProduct2[0]);
rotatedVector[1] =
(float)(Math.cos(angle)*vectorOldPos[1] +
Math.sin(angle)*crossProduct2[1]);
rotatedVector[2] =
(float)(Math.cos(angle)*vectorOldPos[2] +
Math.sin(angle)*crossProduct2[2]);

float rotatedVectorMagnitude =
(float)Math.sqrt(Math.pow(rotatedVector[0], 2) +
Math.pow(rotatedVector[1], 2) +
Math.pow(rotatedVector[2], 2));
rotatedVector[0] =
(rotatedVector[0]/rotatedVectorMagnitude)*magnitudeNewVector +
firstRefPosition3D[0];
rotatedVector[1] =
(rotatedVector[1]/rotatedVectorMagnitude)*magnitudeNewVector +
firstRefPosition3D[1];
rotatedVector[2] =
(rotatedVector[2]/rotatedVectorMagnitude)*magnitudeNewVector +
firstRefPosition3D[2];

// Apply the scaling and rotation to all coordinates
// (except for the ones that are already in the right position)
for (int j = 0; j < newSurroundMap.length; j+=3) {
if (j != georeferencesInUse[0] && j !=
georeferencesInUse[1]) {
float[] currentVector = {newSurroundMap[j],
newSurroundMap[j+1], newSurroundMap[j+2]};
float oldDistance = distBetweenPoints(currentVector,
firstRefPosition3D);
float newDistance = oldDistance * distFactor;

// Vector A
currentVector[0] = (currentVector[0] - firstRefPosition3D[0]);

// Create an imaginary point to rotate towards. For the georeference this is a real point (the newly found 3D-position for that georeference)
float rotateTowardsX = (newSurroundMap[j] + (secondRefPosition3D[0]+currentSecondRefPosition3D[0]));
float rotateTowardsY = (newSurroundMap[j+1] + (secondRefPosition3D[1]+currentSecondRefPosition3D[1]));

float[] rotateTowardsPoint = {rotateTowardsX, rotateTowardsY, rotateTowardsZ};

// Vector B
float[] newVector =
{rotateTowardsX - firstRefPosition3D[0],
rotateTowardsY - firstRefPosition3D[1],
rotateTowardsZ - firstRefPosition3D[2]};
magnitudeNewVector = (float)Math.sqrt(Math.pow(newVector[0], 2) + Math.pow(newVector[1], 2) + Math.pow(newVector[2], 2));

// C = AxB = (a_y b_z - a_z b_y, a_z b_x - a_x b_z,
a_x b_y - a_y b_x),
float[] crossProductC = new float[3];
float crossProductCMagnitude =
(float)Math.sqrt(Math.pow(crossProduct1[0], 2) + Math.pow(crossProduct1[1], 2) + Math.pow(crossProduct1[2], 2));

// Normalize it
crossProductC[0] /= crossProductCMagnitude;
crossProductC[1] /= crossProductCMagnitude;
crossProductC[2] /= crossProductCMagnitude;

// F = CxA
float[] crossProductF = new float[3];
// G = cos(angle) A + sin(angle) F
float [] newRotatedVector = new float [3];

newRotatedVector[0] =
((float)(Math.cos(angle)*currentVector[0] + 
Math.sin(angle)*crossProductF[0]));
newRotatedVector[1] =
((float)(Math.cos(angle)*currentVector[1] + 
Math.sin(angle)*crossProductF[1]));
newRotatedVector[2] =
((float)(Math.cos(angle)*currentVector[2] + 
Math.sin(angle)*crossProductF[2]));

float newRotatedVectorMagnitude =
(float)Math.sqrt(Math.pow(newRotatedVector[0], 2) 
+ Math.pow(newRotatedVector[1], 2) 
+ Math.pow(newRotatedVector[2], 2));

newRotatedVector[0] =
(newRotatedVector[0]/newRotatedVectorMagnitude)*magnitudeNewVector 
+ firstRefPosition3D[0];
newRotatedVector[1] =
(newRotatedVector[1]/newRotatedVectorMagnitude)*magnitudeNewVector 
+ firstRefPosition3D[1];
newRotatedVector[2] =
(newRotatedVector[2]/newRotatedVectorMagnitude)*magnitudeNewVector 
+ firstRefPosition3D[2];

newSurroundMap[j] = newRotatedVector[0];
newSurroundMap[j+1] = newRotatedVector[1];
newSurroundMap[j+2] = newRotatedVector[2];

if (numGeoreferences == 0) {
    newSurroundMap = remapCoordsAroundCenterPos(surroundMapCoords ,
    center);
}

return newSurroundMap;

B.1.13 Remap Coordinates Around Center Position

public float [] remapCoordsAroundCenterPos(float [] coords, float []
centerPos) {
    // The array should contain an x and y value for every edge of a
    // line
    if (coords.length%3 != 0) {
        Log.e("", "Wrong input for surround map coordinates");
        return null;
    } else {
        // Scaling the map down by finding the top-right most
        // coordinate (if the coordinates would be put on a rectangle)
        float topRightX = 0.0f;
    }
float topRightZ = 0.0f;

float [] newCoords = new float[coords.length];
for (int i = 0; i < coords.length; i++) {
    newCoords[i] = coords[i]*1000.0f;
    i++;
    newCoords[i] = coords[i];
    i++;
    newCoords[i] = coords[i]*1000.0f;
}

for (int i = 0; i < newCoords.length; i++) {
    if (newCoords[i] > topRightX) {
        topRightX = newCoords[i];
    }
    i += 2;
    if (newCoords[i] > topRightZ) {
        topRightZ = newCoords[i];
    }
}

// Finding the bottom left most coordinate for a first shift of the coordinates so the origin is in the bottom left corner
float bottomLeftX = topRightX;
float bottomLeftZ = topRightZ;
for (int i = 0; i < newCoords.length; i++) {
    if (newCoords[i] < bottomLeftX) {
        bottomLeftX = newCoords[i];
    }
    i += 2;
    if (newCoords[i] < bottomLeftZ) {
        bottomLeftZ = newCoords[i];
    }
}

for (int i = 0; i < newCoords.length; i++) {
    newCoords[i] = newCoords[i] - bottomLeftX;
    i += 2;
    newCoords[i] = newCoords[i] - bottomLeftZ;
}

topRightX -= bottomLeftX;
topRightZ -= bottomLeftZ;

// Finding the factor by which to divide all coordinates to make them fit on a map of chosen size MAPSIZE
float divFactor = 1.0f;

// Coordinates of the lines are too big to fit on a map of given size
if (MAPSIZE <= topRightX || MAPSIZE <= topRightZ) {
    while (topRightX > MAPSIZE || topRightZ > MAPSIZE) {
        if (topRightX > MAPSIZE) {
            topRightZ = (topRightZ / topRightX) * MAPSIZE;
            divFactor = divFactor * MAPSIZE / topRightX;
            topRightX = MAPSIZE;
        }
        if (topRightZ > MAPSIZE) {
            topRightX = (topRightX / topRightZ) * MAPSIZE;
            divFactor = divFactor * (MAPSIZE / topRightZ);
        }
    }
}
B.1. First Prototype

```java
    topRightZ = MAPSIZE;
    }
}
// Coordinates of the lines are smaller than the map of given
// size and we need to scale them up
else {
    while (topRightX < MAPSIZE || topRightZ < MAPSIZE) {
        if (topRightX < MAPSIZE) {
            topRightZ = (topRightZ / topRightX) * MAPSIZE;
            divFactor = divFactor * MAPSIZE / topRightX;
            topRightX = MAPSIZE;
        }
        if (topRightZ < MAPSIZE) {
            topRightX = (topRightX / topRightZ) * MAPSIZE;
            divFactor = divFactor * (MAPSIZE / topRightZ);
            topRightZ = MAPSIZE;
        }
    }
    while (topRightX > MAPSIZE || topRightZ > MAPSIZE) {
        if (topRightX > MAPSIZE) {
            topRightZ = (topRightZ / topRightX) * MAPSIZE;
            divFactor = divFactor * MAPSIZE / topRightX;
            topRightX = MAPSIZE;
        }
        if (topRightZ > MAPSIZE) {
            topRightX = (topRightX / topRightZ) * MAPSIZE;
            divFactor = divFactor * (MAPSIZE / topRightZ);
            topRightZ = MAPSIZE;
        }
    }
    float newCenterX =
        (centerPos[0]*1000.0f-bottomLeftX)*divFactor;
    float newCenterZ =
        (centerPos[1]*1000.0f-bottomLeftZ)*divFactor;
    for (int i = 0; i < newCoords.length; i++) {
        // Shift and scale the coordinates
        newCoords[i] = ((newCoords[i]*divFactor)-newCenterX);
        i += 2;
        newCoords[i] = (-1.0f)*((newCoords[i]*divFactor)-newCenterZ);
    }
    return newCoords;
}
```

B.1.14 Draw Scene

```java
@Override
public void onDrawEye(EyeTransform transform) {
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
        GLES20.GL_DEPTH_BUFFER_BIT);
    mPositionParam = GLES20.glGetAttribLocation(mGlProgram,
        "a_Position");
    mNormalParam = GLES20.glGetAttribLocation(mGlProgram,
        "a_Normal");
```
mColorParam = GLES20.glGetAttribLocation(mGlProgram, "a_Color");
mTextureCoordinateHandle =
    GLES20.glGetAttribLocation(mGlProgram, "a_TexCoordinate");
GLES20.glEnableVertexAttribArray(mPositionParam);
GLES20.glEnableVertexAttribArray(mNormalParam);
GLES20.glEnableVertexAttribArray(mColorParam);
checkGLError("mColorParam");

// Apply the eye transformation to the camera.
Matrix.multiplyMM(mView, 0, transform.getEyeView(), 0, mCamera, 0);

// Set the position of the light
Matrix.multiplyMV(mLightPosInEyeSpace, 0, mView, 0, mLightPosInWorldSpace, 0);
GLES20.glUniform3f(mLightPosParam, mLightPosInEyeSpace[0],
    mLightPosInEyeSpace[1],
    mLightPosInEyeSpace[2]);

// Set the active texture unit to texture unit 0.
GLES20.glActiveTexture(GLES20.GL_TEXTURE0);

// Bind the texture to this unit.
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, mTextureDataHandle);

// Tell the texture uniform sampler to use this texture in the
// shader by binding to texture unit 0.
GLES20.glUniform1i(mTextureUniformHandle, 0);
Matrix.multiplyMM(mModelView, 0, mView, 0, mModelWalls, 0);
Matrix.multiplyMM(mModelViewProjection, 0,
    transform.getPerspective(), 0, mModelView, 0);
drawWalls();

// Build the ModelView and ModelViewProjection matrices
Matrix.multiplyMM(mModelView, 0, mView, 0, mModelFloor, 0);
Matrix.multiplyMM(mModelViewProjection, 0,
    transform.getPerspective(), 0, mModelView, 0);
drawGround();

// Build the ModelView and ModelViewProjection matrices
Matrix.multiplyMM(mModelView, 0, mView, 0, mModelCeiling, 0);
Matrix.multiplyMM(mModelViewProjection, 0,
    transform.getPerspective(), 0, mModelView, 0);
drawCeiling();

// Set the active texture unit to texture unit 0.
GLES20.glActiveTexture(GLES20.GL_TEXTURE0);

// Bind the texture to this unit.
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, mMapTextureDataHandle);

// Disable depth test so lines get drawn even if they're
// obstructed
GLES20.glDisable(GLES20.GL_DEPTH_TEST);

// Build the ModelView and ModelViewProjection matrices
Matrix.multiplyMM(mModelView, 0, mView, 0, mModelLines, 0);
Matrix.multiplyMM(mModelViewProjection, 0, transform.getProjection(), 0, mModelView, 0);
drawLines();
GLES20.glEnable(GLES20.GL_DEPTH_TEST);
}

@Override
public void onFinishFrame(Viewport viewport) {
}

public void drawGround() {
    // 2f for texture without transparancy
    GLES20.glUniform1f(mIsSurroundMapParam, 2f);
    // Set the Model in the shader, used to calculate lighting
    GLES20.glUniformMatrix4fv(mModelParam, 1, false, mModelFloor, 0);
    // Set the ModelView in the shader, used to calculate lighting
    GLES20.glUniformMatrix4fv(mModelViewParam, 1, false, mModelView, 0);
    // Set the position of the plane
    GLES20.glVertexAttribPointer(mPositionParam, COORDS_PER_VERTEX, GLES20.GL_FLOAT, false, 0, mGroundVertices);
    // Set the ModelViewProjection matrix in the shader.
    GLES20.glUniformMatrix4fv(mModelViewProjectionParam, 1, false, mModelViewProjection, 0);
    // Set the normal positions of the plane, again for shading
    GLES20.glVertexAttribPointer(mNormalParam, 3, GLES20.GL_FLOAT, false, 0, mGroundNormals);
    GLES20.glVertexAttribPointer(mColorParam, 4, GLES20.GL_FLOAT, false, 0, mGroundColors);
    GLES20.glVertexAttribPointer(mTextureCoordinateHandle, mTextureCoordinateDataSize, GLES20.GL_FLOAT, false, 0, mGroundTextureCoords);
    GLES20.glEnableVertexAttribArray(mTextureCoordinateHandle);
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, (CIRCLEPOINTS * 3));
    GLES20.glDisableVertexAttribArray(mTextureCoordinateHandle);
}

public void drawCeiling() {
    // 2f for texture without transparancy
    GLES20.glUniform1f(mIsSurroundMapParam, 2f);
    // Set the Model in the shader, used to calculate lighting
    GLES20.glUniformMatrix4fv(mModelParam, 1, false, mModelCeiling, 0);
// Set the ModelView in the shader, used to calculate lighting
GLES20.glUniformMatrix4fv(mModelViewParam, 1, false, mModelView, 0);

// Set the position of the plane
GLES20.glVertexAttribPointer(mPositionParam, COORDS_PER_VERTEX,
    GLES20.GL_FLOAT,
    false, 0, mCeilingVertices);

// Set the ModelViewProjection matrix in the shader.
GLES20.glUniformMatrix4fv(mModelViewProjectionParam, 1, false,
mModelViewProjection, 0);

// Set the normal positions of the plane, again for shading
GLES20.glVertexAttribPointer(mNormalParam, 3, GLES20.GL_FLOAT,
    false, 0, mCeilingNormals);
GLES20.glVertexAttribPointer(mColorParam, 4, GLES20.GL_FLOAT,
    false, 0, mCeilingColors);
GLES20.glVertexAttribPointer(mTextureCoordinateHandle,
    mTextureCoordinateDataSize, GLES20.GL_FLOAT, false,
    0, mCeilingTextureCoords);
GLES20.glEnableVertexAttribArray(mTextureCoordinateHandle);
GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, (CIRCLEPOINTS * 3));
GLES20.glDisableVertexAttribArray(mTextureCoordinateHandle);
}

public void drawWalls() {
    // 2f for texture without transparency
    GLES20.glUniform1f(mIsSurroundMapParam, 2f);
    // Set the Model in the shader, used to calculate lighting
    GLES20.glUniformMatrix4fv(mModelParam, 1, false, mModelWalls, 0);
    // Set the ModelView in the shader, used to calculate lighting
    GLES20.glUniformMatrix4fv(mModelViewParam, 1, false,
mModelView, 0);
    // Set the position of the sphere
    GLES20.glVertexAttribPointer(mPositionParam, COORDS_PER_VERTEX,
        GLES20.GL_FLOAT,
        false, 0, mWallsVertices);
    // Set the ModelViewProjection matrix in the shader.
    GLES20.glUniformMatrix4fv(mModelViewProjectionParam, 1, false,
mModelViewProjection, 0);
    // Set the normal positions of the sphere, again for shading
    GLES20.glVertexAttribPointer(mNormalParam, 3, GLES20.GL_FLOAT,
        false, 0, mWallsNormals);
GLES20.glVertexAttribPointer(mColorParam, 4, GLES20.GL_FLOAT, 
   false, 
   0, mWallsColors);
GLES20.glVertexAttribPointer(mTextureCoordinateHandle, 
   mTextureCoordinateDataSize, GLES20.GL_FLOAT, false, 
   0, mWallsTextureCoords);
GLES20.glEnableVertexAttribArray(mTextureCoordinateHandle);
GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, 
   (CIRCLEPOINTS*(CIRCLEPOINTS-2)*6));
GLES20.glDisableVertexAttribArray(mTextureCoordinateHandle);
checkGLError("Drawing walls");
}

public void drawLines() {
   // if for regular colors
   GLES20.glUniform1f(mIsSurroundMapParam, 1.0f);
   // Set the Model in the shader, used to calculate lighting
   GLES20.glUniformMatrix4fv(mModelParam, 1, false, mModellines, 0);
   // Set the ModelView in the shader, used to calculate lighting
   GLES20.glUniformMatrix4fv(mModelViewParam, 1, false, mModelView, 0);
   // Set the position of the sphere
   GLES20.glVertexAttribPointer(mPositionParam, COORDS_PER_VERTEX, 
       GLES20.GL_FLOAT, false, 0, mLinesSurroundMapVertices);
   // Set the ModelViewProjection matrix in the shader.
   GLES20.glUniformMatrix4fv(mModelViewProjectionParam, 1, false, 
       mModelViewProjection, 0);
   // Set the normal positions of the lines, again for shading
   GLES20.glVertexAttribPointer(mNormalParam, 3, GLES20.GL_FLOAT, 
       false, 0, mLinesSurroundMapNormals);
   GLES20.glVertexAttribPointer(mColorParam, 4, GLES20.GL_FLOAT, 
       false, 
       0, mLinesSurroundMapColors);
   GLES20.glLineWidth(LINEWIDTH);
   GLES20.glDrawArrays(GLES20.GL_LINES, 0, numLineEdges);
   checkGLError("Drawing line");
}

//loadTexture - code from:
   http://www.learnopengles.com/android-lesson-four-introducing-basic-texturing/
public static int loadTexture(final Context context, final int resourceId)
{
   final int[] textureHandle = new int[1];
   GLES20.glGenTextures(1, textureHandle, 0);
if (textureHandle[0] != 0)
{
final BitmapFactory.Options options = new BitmapFactory.Options();
options.inScaled = false;  // No pre-scaling

// Read in the resource
final Bitmap bitmap =
    BitmapFactory.decodeResource(context.getResources(), resourceId, options);

// Bind to the texture in OpenGL
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textureHandle[0]);

// Set filtering
GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
    GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_NEAREST);
GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
    GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_NEAREST);

// Load the bitmap into the bound texture.
GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, bitmap, 0);

// Recycle the bitmap, since its data has been loaded into OpenGL.
bitmap.recycle();
}

if (textureHandle[0] == 0)
{
throw new RuntimeException("Error loading texture.");
}

return textureHandle[0];

B.2 Second Prototype

B.2.1 Custom Shader

Shader "Custom/UnlitTexture" {
Properties {
    _MainTex ("Base (RGB)", 2D) = "white" {} 
    _Color("Color", Color) = (1,1,1,1)
}

SubShader {
    Tags { "RenderType"="Opaque" }
    LOD 100

    Pass {
        CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #pragma multi_compile_fog

B.2 Second Prototype

B.2.1 Custom Shader

Shader "Custom/UnlitTexture" {
Properties {
    _MainTex ("Base (RGB)", 2D) = "white" {} 
    _Color("Color", Color) = (1,1,1,1)
}

SubShader {
    Tags { "RenderType"="Opaque" }
    LOD 100

    Pass {
        CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #pragma multi_compile_fog

3  _MainTex ("Base (RGB)", 2D) = "white" {}
4  _Color("Color", Color) = (1,1,1,1)
5 }

7 SubShader {
8 Tags { "RenderType"="Opaque" }
9 LOD 100
10
11 Pass {
12 CGPROGRAM
13 #pragma vertex vert
14 #pragma fragment frag
15 #pragma multi_compile_fog
16
```cpp
#include "UnityCG.cginc"

struct appdata_t {
    float4 vertex : POSITION;
    float2 texcoord : TEXCOORD0;
};

struct v2f {
    float4 vertex : SV_POSITION;
    half2 texcoord : TEXCOORD0;
    UNITY_FOG_COORDS(1)
};

sampler2D _MainTex;
float4 _MainTex_ST;
fixed4 _Color;

v2f vert (appdata_t v) {
    v2f o;
    o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
    o.texcoord = TRANSFORM_TEX(v.texcoord, _MainTex);
    UNITY_TRANSFER_FOG(o, o.vertex);
    return o;
}

fixed4 frag (v2f i) : SV_Target {
    fixed4 col = tex2D(_MainTex, i.texcoord) * _Color;
    UNITY_APPLY_FOG(i.fogCoord, col);
    UNITY_OPAQUE_ALPHA(col.a);
    return col;
}
ENDCG

B.2.2 Inverted Sphere

using UnityEngine;
using UnityEditor;

// Code from 
public class InvertedSphere : EditorWindow {
    private string st = "1.0";

    [MenuItem("GameObject/Create Other/Inverted Sphere...")]
    public static void ShowWindow() {
        EditorWindow.GetWindow(typeof(InvertedSphere));
    }

    public void OnGUI() {
        GUILayout.Label("Enter sphere size: ");
        
```
st = GUILayout.TextField(st);

float f;
if (!float.TryParse(st, out f))
    f = 1.0f;
if (GUILayout.Button("Create Inverted Sphere")) {
    CreateInvertedSphere(f);
}

private void CreateInvertedSphere(float size) {
    GameObject go =
        GameObject.CreatePrimitive(PrimitiveType.Sphere);
    MeshFilter mf = go.GetComponent<MeshFilter>();
    Mesh mesh = mf.sharedMesh;
    GameObject goNew = new GameObject();
    goNew.name = "Inverted Sphere";
    MeshFilter mfNew = goNew.AddComponent<MeshFilter>();
    mfNew.sharedMesh = new Mesh();

    // Scale the vertices;
    Vector3[] vertices = mesh.vertices;
    for (int i = 0; i < vertices.Length; i++)
        vertices[i] = vertices[i] * size;
    mfNew.sharedMesh.vertices = vertices;

    // Reverse the triangles
    int[] triangles = mesh.triangles;
    for (int i = 0; i < triangles.Length; i += 3) {
        int t = triangles[i];
        triangles[i] = triangles[i + 2];
        triangles[i + 2] = t;
    }
    mfNew.sharedMesh.triangles = triangles;

    // Reverse the normals;
    Vector3[] normals = mesh.normals;
    for (int i = 0; i < normals.Length; i++)
        normals[i] = -normals[i];
    mfNew.sharedMesh.normals = normals;

    mfNew.sharedMesh.uv = mesh.uv;
    mfNew.sharedMesh.uv2 = mesh.uv2;
    mfNew.sharedMesh.RecalculateBounds();

    // Add the same material that the original sphere used
    MeshRenderer mr = goNew.AddComponent<MeshRenderer>();
    mr.sharedMaterial =
        go.GetComponent<Renderer>().sharedMaterial;

    DestroyImmediate(go);
}
B.2.3 Mega Map

```csharp
void Update () {
    RaycastHit hit;
    Vector3 CameraCenter = camera.ScreenToWorldPoint(new Vector3(Screen.width / 2f, Screen.height / 2f, camera.nearClipPlane));
    //If the raycast hits something
    if (Physics.Raycast(CameraCenter, camera.transform.forward, out hit, 500, layerMask)) {
        LookingAtObject = true;
        //Display the name of the object if it's not already being displayed
        //Find object user is looking at
        Vector3 hitObjectPos = hit.transform.position;
        if (hit.transform.name != objectName.text) textCreated = false;
        objectName.font = font;
        objectName.text = hit.transform.name;
    }
    //Check whether it has been scaled up (or down)
    //This is important because this means the middle of
    //the object is shifted if it's scaled
    Vector3 scaleFactor = hit.transform.localScale;
    if (scaleFactor.x == 1f) {
        scaleFactor = transform.localScale;
    }
    //Find the middle of the object so the line
    //(from text to object) is in the middle of the object
    MeshFilter m = hit.transform.gameObject.GetComponent<MeshFilter>();
    Vector3 middlePoint = new Vector3(0f, 0f, 0f);
    //Calculate the average of all vertices
    //to find the middle (this is only done once) when
    //a user is looking at an object. Not at every frame.
    if (!textCreated) {
        int verticesSize = m.mesh.vertexCount;
        for (int i = 0; i < verticesSize; i++) {
            middlePoint.x += m.mesh.vertices[i].x;
            middlePoint.y += m.mesh.vertices[i].y;
            middlePoint.z += m.mesh.vertices[i].z;
        }
        middlePoint.x /= verticesSize;
        middlePoint.y /= verticesSize;
        middlePoint.z /= verticesSize;
        objectMiddle = middlePoint;
    }
    //Go from local point to world (Meshes all point towards
    //the same point so to find an object, use the local
    //position of vertices of its mesh added to the world position)
    Vector3 buildingOffset = new Vector3(objectMiddle.x * scaleFactor.x, objectMiddle.y * scaleFactor.y, objectMiddle.z * scaleFactor.z);
```
Appendix B. Code Snippets

buildingOffset = hitObjectPos + buildingOffset;
buildingOffset = RotatePointAroundPivot(buildingOffset, hitObjectPos, transform.localRotation.eulerAngles);
hitObjectPos = buildingOffset;

// Render a line from the object to its name to clarify what object the name is referencing
LineRenderer lineRenderer = GetComponent<LineRenderer>();
if (lineRenderer == null) {
    lineRenderer = gameObject.AddComponent<LineRenderer>();
    lineRenderer.material = new Material(Shader.Find("Particles/Additive"));
    lineRenderer.SetColors(Color.red, Color.yellow);
    lineRenderer.SetWidth(0.05f, 0.1f);
    lineRenderer.SetVertexCount(2);
}
lineRenderer.SetPosition(0, objectName.transform.position);
lineRenderer.SetPosition(1, hitObjectPos);

textCreated = true;
}
// User is no longer looking at an object -> remove line
else {
    textCreated = false;
    LookingAtObject = false;
    objectName.text = "";
    currentText = null;
    LineRenderer lineRenderer = GetComponent<LineRenderer>();
    Destroy(lineRenderer);
    lineRenderer = null;
}

B.2.4 Radar

void Update () {
    // transform.parent.eulerAngles = startRotation;
    RectTransform thisRectTransform = GetComponent<RectTransform>();
    Vector3 userRotation =
        centerEyeAnchor.transform.rotation.eulerAngles;
    // Vector of what the user is looking at
    Vector3 userLookAt = centerEyeAnchor.transform.forward;
    userLookAt.y = 0f;
    userLookAt.Normalize();
    // Find the vector between the user and the destination
    Vector3 pointAtDirectionVector = new Vector3
        (pointAtDirection.x -
        centerEyeAnchor.transform.position.x, 0f,
        pointAtDirection.z - centerEyeAnchor.transform.position.z);
    pointAtDirectionVector.Normalize();
    // Calculate the angle between what users are looking it
    // and what they should be looking at
B.2. Second Prototype

```csharp
float rotationAngle = 360 - Vector3.Angle(userLookAt, pointAtDirectionVector);
Vector3 cross = Vector3.Cross(userLookAt, pointAtDirectionVector);
if (cross.y < 0)
    rotationAngle = -rotationAngle;
Vector3 newRotation = new Vector3(0f, 0f, rotationAngle);
thisRectTransform.localEulerAngles = newRotation;

B.2.5 Mini-map (Exploration) Activation Script

public class InitMiniMap : MonoBehaviour {
    public Texture mapTexture;
    public GameObject minimap;

    void Start () {
        EnableMiniMap ();
        visualizationController vC = GameObject.Find("visualizationController").GetComponent<visualizationController>();
        vC.decreaseMinimapHelpers();
    }

    void OnEnable () {
        EnableMiniMap();
    }

    void OnDisable () {
        DisableMiniMap();
    }

    public void EnableMiniMap () {
        minimap.GetComponent<RawImage>().texture = mapTexture;
        minimap.SetActive(true);
        visualizationController vC = GameObject.Find("visualizationController").GetComponent<visualizationController>();
        vC.increaseMinimapHelpers();
    }

    public void DisableMiniMap () {
        GameObject findVC = GameObject.Find("visualizationController");
        visualizationController vC = null;
        if (findVC != null) {
            vC = findVC.GetComponent<visualizationController> ();
            vC.decreaseMinimapHelpers();
            if (vC != null & vC.getNumMinimapHelpersActive () < 1)
                minimap.SetActive(false);
        }
    }
}
B.2.6 Show Navigation Helper

```csharp
public void ShowNavigationHelper(Helper helperName, string navName) {
    GameObject[] navigationObjects = null;
    Transform chosenHelper = null;

    NavigationVisualization nV = new NavigationVisualization (navName, helperName);
    bool navFound = false;
    for (int p = 0; p < activeHelpers.Count && !navFound; p++) {
        if (activeHelpers[p] == nV) {
            navFound = true;
            activeHelpers.RemoveAt(p);
        }
    }
    if (!navFound)
        activeHelpers.Add (new NavigationVisualization (navName, helperName));

    // Load correct list of helpers
    if (helperName == Helper.ARNavigation)
        chosenHelper = ARNavigations.transform;
    else if (helperName == Helper.AROrientation)
        chosenHelper = AROrientations.transform;
    else if (helperName == Helper.ARAnnotation)
        chosenHelper = ARAnnotations.transform;
    else if (helperName == Helper.ARExploration)
        chosenHelper = ARExplorations.transform;
    else if (helperName == Helper_WidgetNavigation)
        chosenHelper = WidgetNavigations.transform;
    else if (helperName == Helper_WidgetOrientation)
        chosenHelper = WidgetOrientations.transform;
    else if (helperName == Helper_WidgetAnnotation)
        chosenHelper = WidgetAnnotations.transform;
    else if (helperName == Helper_WidgetExploration)
        chosenHelper = WidgetExplorations.transform;

    navigationObjects = new GameObject[chosenHelper.transform.childCount];

    if (navName == "HeatVision")
        showHeatVision();

    int i=0;
    bool helperActive = false;
    // Go through list of current category
    foreach(Transform navigationObject in chosenHelper) {
        navigationObjects[i] = navigationObject.gameObject;
        if (navigationObjects[i].name == navName) {
            // Go through list of helpers (there is a helper for each streetview position)
            GameObject[] helpers = new GameObject[navigationObjects[i].transform.childCount];
            int currentPos = 0;
            foreach(Transform helper in navigationObjects[i].transform) {
                helpers[currentPos] = helper.gameObject;
                // Other code...
            }
            helperActive = true;
        }
    }
}
```
//Activate or deactivate chosen helper
if (currentPos == currentPosition) {
    //The HUD-visualizations can be put in a different
corner of the screen. This was used during user
tests.
    if (navName == "MiniMapNav" || navName ==
        "2DPointingArrow" || navName == "Radar" || navName
        == "MiniMapAnnotations" || navName == "MiniMapIcons"
        || navName == "MiniMap" || navName == "Compass")
        SetWidgetUIPosition (navName, helpers[currentPos]);
    if (helpers[currentPos].activeSelf) {
        helpers[currentPos].SetActive (false);
    }
    else {
        helpers[currentPos].SetActive (true);
        helperActive = true;
    }
    break;
} else {
    currentPos ++;
}
if (helperActive)
    break;
i ++;

B.2.7 Menu

A “menu-screen” was created for each
category of visualizations. The screens
are controlled by their gameobjects. The
HelperSelectionMenu gameobject will en-
able the right “main-menu”-gameobject
(Widgets or AR). This gameobject will
then allow the user to select an item in
the menu (using the XBOX-controller)
and opening a lowest layer of the menu
where actual visualizations are enabled
or disabled.

Fig. B.1: Structure of the menu-
buttons.

```c
void Update () {
    if (canOpenMenu) {
        //Check state of the buttons on the controller
        bool leftShoulderButton = OVRGamepadController.GPC_GetButton
            (OVRGamepadController.Button.LeftShoulder);
        bool rightShoulderButton = OVRGamepadController.GPC_GetButton
            (OVRGamepadController.Button.RightShoulder);
```
Appendix B. Code Snippets

```csharp
if (leftShoulderButton && (leftShoulderButton !=
    lastLeftShoulder)) {
    // Close widget menu if active
    if (HelperSelectionMenuWidgets.activeSelf) {
        HelperSelectionMenuWidgets.SetActive (false);
    }
    HelperSelectionMenuWidgetItemScript menuScript =
        HelperSelectionMenuAR.GetComponent<HelperSelectionMenuWidgetItemScript> ();
    // Open or close AR menu
    if (HelperSelectionMenuAR.activeSelf && !menuScript.enabled)
        menuScript.enabled = true;
    else
        HelperSelectionMenuAR.SetActive (!HelperSelectionMenuAR.activeSelf);
    // If the right button is pressed (and kept pressed)
} else if (rightShoulderButton && (rightShoulderButton !=
    lastRightShoulder)) {
    // Close AR menu if active
    if (HelperSelectionMenuAR.activeSelf) {
        HelperSelectionMenuAR.SetActive (false);
    }
    HelperSelectionMenuWidgetItemScript menuScript =
        HelperSelectionMenuWidgets.GetComponent<HelperSelectionMenuWidgetItemScript> ();
    // Open or close widget menu
    if (HelperSelectionMenuWidgets.activeSelf && !menuScript.enabled)
        menuScript.enabled = true;
    else
        HelperSelectionMenuWidgets.SetActive (!HelperSelectionMenuWidgets.activeSelf);
}
```

When either the AR- or Widget-menu is enabled, they will run a script of their own to show actual menu-items that can be selected. Their Start()- and OnEnable()-functions show the items.

```csharp
void Start () {
    // Load gameobjects references in a list
    itemLists = new GameObject[5];
    itemLists[0] = MainButtons;
    itemLists[1] = NavItems;
    itemLists[2] = OrientItems;
    itemLists[4] = ExploItems;
    itemLists[0].SetActive (true);
    for (int i = 1; i < 5; i++)
        itemLists[i].SetActive (false);
    pos = 0;
}
```
B.2. Second Prototype

Their update function will listen to button presses and the pushes of the analog sticks on the XBOX-controller to change the selected item or swap to its corresponding menu. The menu-categories have a Helper-type that is used in the ShowNavigationHelper()-function. The position in the list is also the same position as the items have in the Hierarchy of gameobjects. This way we know which visualization currently needs to be enabled.

Fig. B.2: Manipulating the options in a menu.
if ((OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.LeftYAxis) > 0.5) && !analogUp) {
    pos--;  
    if (pos < 0)  
        pos = itemLists[currentItemList].transform.childCount - 1;
    setItemSelected(itemLists[currentItemList], pos);  
    analogUp = true;
}
else if (OVRGamepadController.GPC_GetAxis(OVRGamepadController.Axis.LeftYAxis) <= 0.5)  
    analogUp = false;

//A-button pressed on an item -> set the item as selected

if (OVRGamepadController.GPC_GetButton
    (OVRGamepadController.Button.A) && !aDown) {
    if (currentItemList == 0) {
        if (itemLists[pos+1].transform.childCount > 0) {
            itemLists[currentItemList].SetActive(false);
            itemLists[pos+1].SetActive(true);
            currentItemList = pos+1;
            pos = 0;
            //Changes the text of the selected menu-item to yellow so 
            //users know which item they currently has the focus
            setItemSelected(itemLists[currentItemList], pos);
        }
    }
}

//Item selected and pressed A-button
else {
    GameObject selectedObject = itemLists[currentItemList].transform.GetChild(pos).gameObject;
    visualizationController vc = GameObject.Find("visualizationController").GetComponent<visualizationController>();
    HelperType type = selectedObject.transform.parent.gameObject.GetComponent<HelperType>();
    vc.ShowNavigationHelper(type.getHelper(), selectedObject.name);
    //Change background color of the menu-item to unselected 
    // (blue) or selected (green)
    Image img = selectedObject.GetComponent<Image>();
    if (img.color != selectedImgColor)
        setImgSelected(img);
    else
        setImgUnselected(img);
    aDown = true;
}

//A-button no longer pressed
else if (!OVRGamepadController.GPC_GetButton
    (OVRGamepadController.Button.A) && aDown)  
    aDown = false;

//B-button pressed so close menu or go back a menu
if (OVRGamepadController.GPC_GetButton
    (OVRGamepadController.Button.B) && !bDown) {
if (currentItemList == 0) {
    itemLists[0].SetActive(false);
    enabled = false;
} else {
    itemLists[0].SetActive(true);
    itemLists[currentItemList].SetActive(false);
    currentItemList = 0;
    pos = 0;
    setItemSelected(itemLists[currentItemList], pos);
}

bDown = true;
} else if (!OVRGamepadController.GPC_GetButton(OVRGamepadController.Button.B) && bDown)
    bDown = false;
}
Appendix C

USER TEST SURVEY
Please fill in the following survey.

Sex: male  female

Age: ......

Have you ever used a Google Glass or a different head-mounted display?

Yes  No

How experienced are you with Google Glass or different head-mounted displays?

No experience at all

Very experienced

0  1  2  3  4  5  6  7  8  9  10

How often do you use a mobile device for helping you with navigation?

Never  Yearly  Monthly  Weekly  Daily

□  □  □  □  □  □  □  □  □  □
1. For each of the previous Navigation-helpers, rate them based on how much you liked the visualizations helping you complete the tasks. You will be given some pictures in case you forgot what the visualization looked like.

<table>
<thead>
<tr>
<th>Mini-map with marked path (1)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>completely unhelpful</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Would never use it</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D-arrow on HUD (2)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>completely unhelpful</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Would never use it</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
### Fog Path (3)

<table>
<thead>
<tr>
<th>completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

### Object Path (4)

<table>
<thead>
<tr>
<th>completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

### Highlighted Path (5)

<table>
<thead>
<tr>
<th>completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>Feature</td>
<td>完全</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>Moving Arrow (6)</td>
<td></td>
</tr>
<tr>
<td>全部视觉上不吸引人</td>
<td></td>
</tr>
<tr>
<td>Would never use it</td>
<td></td>
</tr>
<tr>
<td>3D-arrow next to user (7)</td>
<td></td>
</tr>
<tr>
<td>Static Arrows (8)</td>
<td></td>
</tr>
<tr>
<td>全部视觉上不吸引人</td>
<td></td>
</tr>
<tr>
<td>Would never use it</td>
<td></td>
</tr>
</tbody>
</table>

注：数字代表评分，0-10分，分数越高表示评价越好。
2. If you have any extra comments, you can write them down here.

Mini-map with marked path (1)

2D-arrow on HUD (2)

Fog Path (3)

Object Path (4)

Highlighted Path (5)

Moving Arrow (6)

3D-arrow next to user (7)

Static Arrows (8)
3. For each of the previous Orientation-helpers, rate them based on how much you liked the visualizations helping you complete the tasks. You will be given some pictures in case you forgot what the visualization looked like.

### Compass (9)

<table>
<thead>
<tr>
<th></th>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>visual appeal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>use</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Radar (10)

<table>
<thead>
<tr>
<th></th>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>visual appeal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>use</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Heat Vision (11)

<table>
<thead>
<tr>
<th></th>
<th>completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>Would never use it</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

---

4. **If you have any extra comments, you can write them down here.**

**Compass (9)**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Radar (10)**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Heat Vision (11)**

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
5. For each of the previous Annotation-helpers, rate them based on how much you liked the visualizations helping you complete the tasks. You will be given some pictures in case you forgot what the visualization looked like.

### Mini-map Annotations (12)

<table>
<thead>
<tr>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

### Mini-map Icons/Logos (13)

<table>
<thead>
<tr>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ □ □ □ □ □ □ □ □</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>
### Object Names (14)

<table>
<thead>
<tr>
<th></th>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Very visually appealing</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Would never use it</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Would use it all the time</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
</tbody>
</table>

### Object Icons/Logos (15)

<table>
<thead>
<tr>
<th></th>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Very visually appealing</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Would never use it</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Would use it all the time</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
</tbody>
</table>

### Highlighting (16)

<table>
<thead>
<tr>
<th></th>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Not visually appealing</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Very visually appealing</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>Would never use it</td>
<td>□ □ □ □ □ □ □ □ □ □</td>
<td>Would use it all the time</td>
</tr>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
</tbody>
</table>
6. If you have any extra comments, you can write them down here.

Mini-map annotations (12)

Mini-map Icons/Logos (13)

Object Names (14)

Object Icons/logos (15)

Highlighting (16)
7. For each of the previous Exploration-helpers, rate them based on how much you liked the visualizations helping you complete the tasks.
You will be given some pictures in case you forgot what the visualization looked like.

Mini-map (17)

<table>
<thead>
<tr>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

Mega Map (18)

<table>
<thead>
<tr>
<th>Completely unhelpful</th>
<th>Very helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not visually appealing</th>
<th>Very visually appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Would never use it</th>
<th>Would use it all the time</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

8. If you have any extra comments, you can write them down here.

Mini-map (17)

Mega Map (18)
9. For each category, mark your favourite Augmented Reality- and Widget-visualization and cross out/overstrike your least favourite visualization. You will be given some pictures in case you forgot what the visualizations for each category looked like.

Widget Navigation

- Mini-map with marked path (1)
- 2D-arrow on HUD (2)

Augmented Reality Navigation

- Fog Path (3)
- Object Path (4)
- Highlighted Path (5)
- Moving Arrow (6)
- 3D arrow next to user (7)
- Static Arrows (8)

Widget Orientation

- Compass (9)
- Radar (10)

Augmented Reality Orientation

(Mark this one if you like it, cross this one if you do not like it, or ignore it you have no opinion.)

- Heat Vision (11)

Widget Annotation

- Mini-map Annotations (12)
- Mini-map Icons/Logos (13)

Augmented Reality Annotation

- Object Names (14)
- Object Icons/Logos (15)
- Highlighting (16)
Widget Exploration
(Mark this one if you like it, cross this one if you do not like it, or ignore it you have no opinion.)

☐ Mini-map (17)

Augmented Reality Exploration
(Mark this one if you like it, cross this one if you do not like it, or ignore it you have no opinion.)

☐ Mega map (18)

10. Consider the following categories: Navigation, Orientation, Annotation and Exploration. Which category do you think is best?
   You will be given some pictures in case you forgot what the visualizations for each category looked like.

☐ Navigation
☐ Orientation
☐ Annotation
☐ Exploration

11. Of all the visualizations, mark the three visualizations you like best.
   You will be given some pictures in case you forgot what the visualizations for each category looked like.

☐ Mini-map with marked path (1)
☐ 2D-arrow on HUD (2)
☐ Fog Path (3)
☐ Object Path (4)
☐ Highlighted Path (5)
☐ Moving Arrow (6)
☐ 3D arrow next to user (7)
☐ Static Arrows (8)
☐ Compass (9)
☐ Radar (10)
☐ Heat Vision (11)
☐ Mini-map Annotations (12)
☐ Mini-map Icons/Logos (13)
☐ Object Names (14)
☐ Object Icons/Logos (15)
☐ Highlighting (16)
☐ Mini-map (17)
☐ Mega map (18)
12. Build your own Head-mounted display navigation application! Pick up to four widgets and up to three different Augmented Reality-visualizations that you think would form a good combination. Then try it out. You will be given some pictures in case you forgot what the visualization looked like.

13. Thank you for participating. If you have any extra comments, you can write them down here.

..................................................................................................................................................
Appendix C. User Test Survey
Appendix D

MEETINGS

This section contains a summary of all the meetings with Prof. dr. Johannes Schöning.
1 Meeting 1: 03/07/2014 13.30h - 14h

1.1 Points discussed
General information about the master thesis:

- Basics
- Timing
- Meeting
- How to find papers

1.2 Action points
Finding and reading 15 - 20 papers about:

- how to deal with usability of Google Glass
- different scenarios where Google Glass is used

2 Meeting 2: 26/08/2014 10h - 11h

2.1 Points discussed

- Improvements to abstract and references
- Related work
- Divided the related work into groups (medical, culture, ...)
- General structure of the master thesis

2.2 Action points

- Improve document + add discussed structure
- Hardware overview
- Think about three different “cool” ideas
3 Meeting 3: 2/09/2014 10h - 11h

3.1 Points discussed

- Hardware overview
- Ideas
- A few improvements about some images in the thesis (positions and captions)

3.2 Action points

- Elaborate on ideas (scenarios, functions, situations) while keeping the needed hardware in mind

4 Meeting 4: 10/09/2014 10h - 11h

4.1 Points discussed

- Ideas
- Improvements for the thesis

4.2 Action points

- Detail the idea better
- Write and/or improve sections 2 to 2.4 of the thesis
- Use Google VR cardboard box to create a static position in the EDM and main building

5 Meeting 5: 7/10/2014 15h - 16h

5.1 Points discussed

- Google cardboard prototype (with a texture as the surroundmap) of EDM and UHasselt main building

5.2 Action points

- Build a Google Cardboard
- Align the map to the building
- Draw the map by using “vectors” in stead of an image
- Think of a zoom-function which draws lines smaller, the further they are away
- Add a simple navigation
6 Meeting 6: 12/11/2014 13.30h - 14.30h

6.1 Points discussed

- Incorrect mapping of the “Surround-map” to the buildings on the ODV-images and how to possibly correct it.

6.2 Action points

- Georeference positions on the ODV-images with real-world coordinates and use those for drawing the “Surround-map” with OSM-coordinates.
- Collect examples of AR-“Helpers” in computer games.

7 Meeting 7: 2/12/2014 16h - 17h

7.1 Points discussed

- Google streetview image as 360° picture
- Coordinate system for the “surround-map” (google maps coordinates looked distorted)
- AR-helpers in computer games

7.2 Action points

- Use google maps coordinates for the application
- Get the georeferences working
- Go through the “top 100 greatest open-world games” (http://gamingbolt.com/100-greatest-open-world-games) and write about AR-helpers as well as non-AR helpers in those games
- Write about AR navigation systems (Chapter 1: introduction, chapter 2: real-world AR-research, chapter 3: game implementations) for tasks: navigation, exploration, orientation and “annotation + labels”
8 Meeting 8: 9/1/2015 14h - 15h

8.1 Points discussed
- Slides for in-between presentation
- List of 100 Open World Games
- Thesis text about AR visualisation techniques (and widgets) as aids for navigation in Open World Games and the real world

8.2 Action points
- Improve discussed points of the slides for the in-between presentation
- Merge all documents for the thesis into one document
- Improve discussed sections of the thesis
- Write conclusions for the results of the 100 studied Open World Games
- Find and write about 25 real world applications that use AR as a navigation aid

9 Meeting 9: 27/2/2015 13h - 14h

9.1 Points discussed
- List of 100 Open World Games
- Different kinds of helpers used in these games
- Real world AR-navigation research

9.2 Action points
- Create more graphs for displaying information about the helpers used in OWG and research
- Formulate ten “lessons learned” using this information
- Formulate 5 “lessons learned” about the real world AR-navigation research

10 Meeting 10: 16/3/2015 15h - 16h

10.1 Points discussed
- Brief discussion of overall paper structure
- Visualizations for navigation

10.2 Action points
- Recreate visualizations for navigation with the help of the Oculus Rift
11 Meeting 11: 7/04/2015 10.30h - 11.30h

11.1 Points discussed
- Oculus Rift demo

11.2 Action points
- Create a user test to find out which visualizations are preferred

12 Meeting 12: 20/04/2015 14h - 15h

12.1 Points discussed
- Pilot test of user test

12.2 Action points
- Improve on the user test with more tasks so that every helper is tested multiple times with a 45°, 90°, 120° and 180° turn
- Create a survey for users to fill in with at the end a “build your own application”-opportunity

13 Meeting 13: 5/05/2015 14h - 15h

13.1 Points discussed
- Pilot test of user tests + survey

13.2 Action points
- Test 12 to 16 users

14 Meeting 14: 27/05/2015 14h - 15h

14.1 Points discussed
- Thesis text

14.2 Action points
- Improve the thesis text based on the feedback
15 Meeting 15: 10/06/2015 11h - 12h

15.1 Points discussed
- Thesis text with feedback from dr. Dirk Wenig

15.2 Action points
- Improve the thesis text based on the feedback
BIBLIOGRAPHY


[29] Nguyen, E., Modak, T., Dias, E., Yu, Y., and Huang, L. Fitnamo: Using bodydata to encourage exercise through google glass. In CHI ’14, CHI EA ’14, ACM, pp. 239–244.


Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling: *Exploring different visualization techniques for Augmented Reality navigation helpers inspired by Open World Games*

Richting: *master in de informatica-Human-Computer Interaction*  
Jaar: *2015*

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen -, aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtredt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

* Lodts, Cedric  

Datum: *14/06/2015*