ObDrop - A Mobile Application Using Augmented Reality

Friday 7th June, 2019 - 15:29

Iris Kremer
University of Luxembourg
Email: iris.kremer.001@student.uni.lu

Jean Botev
University of Luxembourg
Email: jean.botev@uni.lu

***Abstract*—This paper presents the second bachelor semester project of Iris Kremer under the direction of her project academic tutor Jean Botev. The project seeks to develop ObDrop, a mobile application using augmented reality which enables users to place virtual objects in the real space, store and share them with the other users of the app. Therefore, the scientific part of this project deals with the concept of augmented reality, while the technical domain concerns mobile application development and geolocation.**

1. Introduction

Augmented reality (AR) has a great potential for the future, because it can be used in many domains and technologies now have evolved far enough to enable efficient, high quality AR. Whether on the photo-sharing app Snapchat [1] or in their car system, most people nowadays use applications involving this concept, even if not consciously. It not only allows to have some fun by trying out some filters, but it can contribute also in fields like education for instance. Understanding geometry in 3D is probably easier when having the 3D shapes in front of our eyes, in the real space. AR is also used in disciplines like archaeology [3] to ease access to knowledge with 3D models, medicine [4] to enhance medical equipment and many more.

In this project, the aim is to build an AR application called ObDrop, where users can interact with the real world by creating virtual objects and save them in relation to their real-world location. And since according to the definition we build up in Section 4, AR data is bound to real-world locations, using AR together with a georeferencing system makes sense. It even already exists some applications using AR and georeferencing together, like Pokémon Go.

In this paper, we first provide a description of the scientific and technical domains of the project, as well as of the target deliverables to produce. Afterwards, we will give some background knowledge on AR and the technologies used to develop ObDrop. Now, we have everything we need to understand the presentation of the deliverables, which are the next two sections. The first one presents the scientific deliverable, while the second one will describe the technical deliverable. Concretely, for each of these two sections, we will set the requirements, present the design, explain the production and assess the result obtained. Finally, we conclude on this project and discuss some further improvement which could be made.

1. https://www.snapchat.com/

2. Project description

2.1. Domains

This project aims to develop a mobile application using AR. Therefore, the scientific domain concerns AR, while the technical domain is focusing on mobile application development for iOS.

2.1.1. Scientific.

The concept of AR appeared for the first time in the 1960s, together with virtual reality (VR), which means it is now more than 65 years old. However, it only began to become a useful tool for diverse applications around 20 years ago. This is because the technological devices required for AR needed to be improved and much research had to be conducted in this domain before the quality of AR applications developed was high enough to make the virtual objects look realistic in the real world.

AR basically works by capturing the real world through a camera and detecting planes and shapes in the space, as well as the depth. Then, virtual elements, which are bound a certain location, are added to the space. They can simulate physical interaction, and this way, we can program interactions between the user and these virtual elements, either through a screen, or even through gestures in the real world.

The whole project is based on AR, since we developed an app which enables to interact with the real world by adding some virtual objects to it. In our project, the user can only interact with the objects through the screen, because it requires less implementation than direct interaction through gestures. But apart from the implementation, it is important to grasp the concept well and give a definition to which we can refer to, as required for the scientific deliverable.

2.1.2. Technical.

Mobile Application Development.

In mobile application development, the most important thing to take care of is the user interface design. A bad interface hinders the use of the app, because it is the only way of communication between user and app, so it must be designed simple and interactive. Also, it is better to develop universal apps, such that the interface looks fine on any mobile device, no matter the screen dimensions.

Since this project uses AR, the interface will be kept simple, with most of the screen showing the camera view.

The software development environment used in this project to develop the mobile application is XCode10, which enables to test the application directly on a real iOS device. We started programming with Swift 4, then upgraded to Swift 5 at its release.

Geolocation.

Geolocation is the process of finding the position of someone or something on the earth. It can be done through multiple technologies. GPS, wifi, relay antenna etc. and even by combining information gathered from all these different technologies. Depending on the location of the user (for instance inside or outside), the one or the other technology may work better to locate him, but the principle remains the same: triangulation. Triangulate a location means measuring the distance between at least three reference points and the target location. Then, for each point, we draw a virtual sphere around it with the radius being the distance measured between this point and the target location. We know that the target location is on each sphere drawn, so with three or more spheres, we only need to overlap them and see at which location all spheres meet. These are the coordinate of the target location. More than three reference points increases the precision of the measurement.

2.2. Targeted Deliverable

2.2.1. Scientific deliverable.

The scientific deliverable of this project is a detailed, scientific definition of augmented reality. Therefore, research will be conducted on this domain and different approaches explored to finally find a suitable definition for AR.

2.2.2. Technical deliverable.

The technical deliverable of this project is a mobile application using AR, called *ObDrop*. It should enable the user to interact with the real world through the mobile devices camera and add virtual objects to it. The user must then be able to save the objects set in the space, such that when he leaves the app and launches it later again, the saved objects are again visible where they were set previously. Also, the saved objects must be visible to all users of the same app.

3. Background

3.1. Scientific background

The first overlays on the physical world using computers were performed by Ivan Sutherland in the 1960s. This was the starting point of both AR and VR. The first VR system, nicknamed “Sword of Damocles” due to the fact that it had to be suspended from the ceiling because of its weight, was built in 1968. For many years, research and experiences were performed in both the virtual and augmented reality fields without tearing apart these notions. AR became an independent field of research around 1990, and the term ”augmented reality” appeared only in 1992.

More and more AR applications were developed during the 1990s. For example, an application in the field of medicine was developed to enable people to observe a fetus in the womb of the mother. Another application targeted the educational field and was built to teach high school students geometry lessons in 3D. But all these applications required special environments, i.e. laboratories of research and equipment.

The first outdoor AR application, equipped with GPS and orientation tracking systems, appeared 1997, and allowed to display location-based information as overlay on the real world. Soon after that, the first AR game was invented: a first-person shooter game called ARQuake, where the player has to defend himself from zombies.

In 1999, the first open-source software for AR, called ARToolKit, was released, and enabled users to make virtual objects appear in the real world on black and white squares with special patterns, which could be printed easily, as shown in Figure 1. This application became very popular, due to the simplicity of use.

In 2008, AR finally became available on smartphones. Nowadays, AR is accessible to everyone and used in many different applications. [6]

There are some common applications of AR nowadays, which many people are using. For instance, some car sys-

Figure 1: ARToolKit version 2.0 for iOS. Image source: [5]

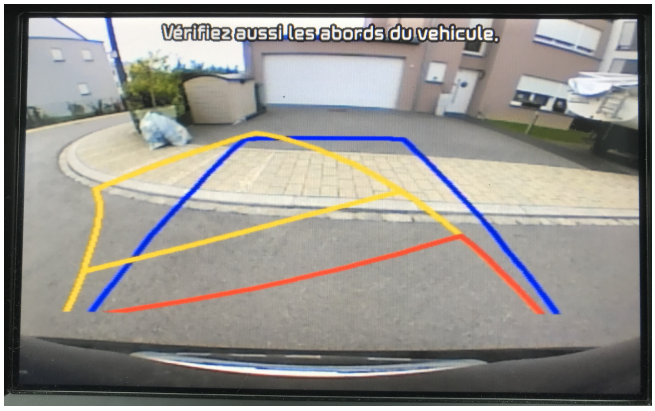


Figure 2: Backdriving assistance overlay.

tems have now a back camera to help the drivers looking behind the car when they drive backwards. On this camera view is a driving assistance overlay showing where the car would drive if the driver continues to turn the wheel the way he does while driving backwards, like in Figure 2. This is a virtual overlay on a real view of the world, which is bound to the car and adjusted according to the orientation of the wheel, so it is AR.

Another common application of AR is in education. For instance, we already mentioned in a previous paragraph an early application of AR in the field of medicine. And apart from the educational aspect, there are great future applications of AR in medicine. The article *"How Augmented Reality Will Make Surgery Safer"* [4] discusses the financial and medical benefits of AR applications displaying data about the health of the patient and advices on the operation (e.g. where to cut the skin) to the doctors during surgeries, using technologies like Microsoft's HoloLens.

These were only a few examples of applications using AR. There exists many others of them, like military training in AR, games and museum apps.

3.2. Technical background

The project is using three frameworks of Apple. The first one is SceneKit, which is used to create and render virtual objects and animations. The second one is ARKit, used to track the real world and render the virtual objects inside, and the last one is Core Location, a framework for determining the device's geolocation using all available technologies and hardware (wifi, GPS etc).

Core Location is only used in this project to get some geographic information about the user's position at the launch of the app. Even though it plays a crucial role in the application, there is no need to know much about this framework to use it. However, there is a lot to learn about SceneKit and ARKit, and it is important to understand how these two frameworks work to fulfill the objectives of this project.

SceneKit.

SceneKit is a 3D graphics framework from Apple [10], which enables to build realistic 3D animated scenes in apps and render them in a virtual view, a so-called SCNView (scene view). It requires at least iOS 8 for iOS devices, but works also on watchOS, TVOS and macOS.

Each scene rendered by SceneKit is built around a "world origin", which is the root node. It is set at position (0, 0, 0) and is the only node which has no parent. All other nodes in the space are directly or indirectly attached to it and they form together the scene graph.

Nodes are structural elements of the scene graph, defined by their position and rotation. Each node has a parent node and a possibly empty array of child nodes. A child node is placed at a certain position relative to its parent, indicated by an SCNVector3 (scene vector). Each node can have a geometry or shape and one or many textures or colours, as well as many other parameters, which define its appearance. They can also have a physicsbody, that enables them to interact with other node's physics bodies, and further properties like weight and friction. Finally, we can make custom animations on the nodes properties.

The rendering of appearance and physics of the nodes works better in appropriate scene environments. SceneKit offers an easy way to set up global parameters for the scene we are rendering. For instance, we can set automatic lightning, such that the objects are shaded and project shade on other objects, and configure the gravity of the scene to match the desired physic effects. [12]

To conclude, the SceneKit framework enables to implement a virtual scene with realistic 3D objects and animations.

ARKit.

ARKit is the framework of Apple for AR [9]. It is only supported on the iOS devices iPhone SE, iPad 2017 and more recent iOS devices, and requires at least iOS 11.

It is tightly related to SceneKit, because it renders 3D objects and animations created using SceneKit on the real world in a special view, called ARSCNView (AR scene view), which is a subclass of the SCNView. This view is actually the camera view, with an overlay to render the virtual 3D objects directly on the real world through the camera. The programmer can choose to implement ARKit for the front or back camera of the device.

ARKit allows to track the real world and detect objects and surfaces. Using this information, it will then set the world origin at the devices location at app launch and set invisible anchors where real surfaces were detected if PlaneDetection is enabled. In the application, the user may implement features to create invisible surfaces or objects where these anchors have been set. While the app is running, it will constantly update these information with newly detected objects when the user moves around in the space. This way, 3D objects can be set directly on detected surfaces

and the user can interact realistically with the real world. For example, when dropping a cube which has a certain weight in the space, the cube will fall down on the surface detected below. [11]

4. Scientific Definition of Augmented Reality

4.1. Requirements

The scientific dimension of this project concerns the domain of AR. In order to learn about this topic, some research has been conducted on this domain.

While some of these research results were already mentioned in previous sections, i.e. the history of AR and some of its applications, the biggest question remained untouched for now: What is AR? Answering this question is the aim of the scientific deliverable in this project. Therefore, a detailed definition of AR will be provided, to be referred to whenever AR is mentioned in this paper.

To ensure the quality of the scientific deliverable, the following requirements have been set:

1) Consider different aspects of the domain of AR and possible criteria for the definition.
2) Search about different definitions given by diverse sources.
3) Basing the argumentation on external references and own justification, build a definition of AR which corresponds to the application of AR used in this project, and possibly others as well.

To these functional requirements, which need to be fulfilled for the scientific deliverable to be complete, we add a comparison between AR and VR as non-functional requirement. As these two topics used to be one unique research field in the past, before it got split in two distinct research domains in the 1990s, comparing them will clarify to which extend these two technologies are similar and in which aspects they differ from each other.

4.2. Design

Definition of Augmented Reality.

The procedure followed for building the definition of AR uses a bottom-up approach. We start from the expression ”augmented reality” itself and build the definition around by adding criteria until the definition matches the applications of AR.

The first step is to look up the definitions of ”augment” and ”reality” in the dictionary, to grasp the global sense of the term and to construct a first, mathematical, definition of AR on that basis.

However, further investigation on the concept is needed to identify all important criteria of AR. Reading from different sources and gather some opinions was the next step to build a more concise definition of the term.

Finally, we can check that the definition obtained describes well the application using AR built in this project, as well as other known applications of AR.

Augmented and Virtual Reality.

To be able to compare the two technologies, considering that we already have the definition of AR at this point, the first thing to do is to define VR. The focus of the project lies on AR and not on VR, so the definition of VR was taken from Wikipedia without deeper research on the topic. Although it is not advised to use Wikipedia as a reference, it is appropriate here, because it should be an acceptable and reasonable compromise between different possible definitions, since many contributors may have adapted and optimized it.

Once we have both definitions, the only task remaining is to compare the properties of both technologies according to their definitions. This way, similarities and differences can be identified.

4.3. Production

Definition of Augmented Reality.

According to the Cambridge dictionary, ”augment” means ”increasing the size or value of something by adding something to it” and ”reality” is ”the actual state of things, or the fact involved in such a state” [7]. In our case, the state mentioned is the real world. We could therefore say that ”taking the real world and add some virtual elements to it” is a definition of AR. But this is not an appropriate definition, because AR describes the state of an environment, not an action. We have to be careful to indeed describe this state. Using the information we have, as a first try to describe AR, we can give a mathematical definition:

$$AR = \{r \mid r \textit{ belongs to the real environment}\} \cup \{v \mid v \textit{ belongs to a virtual environment}\}$$

This definition says that AR is a set, composed by the union of the elements of the real world and virtual objects of a virtual environment. In other terms, it is the physical world extended with virtual elements.

However, the mathematical definition is not complete; it is missing some important aspects of AR which cannot be described easily using mathematics, for instance the relation between the virtual and real objects, or possible interaction. Therefore, we need an improved definition in natural language.

Depending on the definition, a lot of things can be included in AR, like GPS location systems, QR Codes and videos edited by adding some text. There is no true definition which decides on what belongs to AR and what does not - even the boundary between real and virtual is

not clear (e.g. holograms). However, we still need a proper definition to refer to when talking about AR. Therefore, setting some criteria to build the definition is a good way of proceeding.

"Augmented reality is a medium in which information is overlaid on the physical world, that is in both spatial and temporal registration with the physical world and that is interactive in real time." - [8]

This definition given by Alan Craig in his book "Understanding Augmented Reality : Concepts and Applications", published in 2013, takes three criteria into account to define AR:

1) Digital information is overlaid on the physical world.
2) This information is bound to a location of the real world and displayed according to the perspective from which we look at it.
3) Interaction is possible between humans and the digital information, so it can be created, manipulated (maybe sometimes only by moving around to change the perspective) or removed.

The definition of AR given by Apple in the documentation of ARKit is in accordance with these criteria as well:

"Augmented reality (AR) describes user experiences that add 2D or 3D elements to the live view from a device's camera in a way that makes those elements appear to inhabit the real world." [11]

The book to which is referred to before contains other, precise criteria for AR as well, but for the purpose of this project, the three criteria above are sufficient. Therefore, we reformulate them in a nice sentence:

"AR is the concept of overlaying digital information bound to a certain location on the physical world, with the possibility of interaction with this digital information."

From now on, this definition will be the one we refer to when talking about AR, because it suits well the aim of this project to create a mobile application for AR. Indeed, using this definition, we can easily show in which sense the application developed in this project is an AR application, because it is inline with the definition of AR given by Apple in their documentation.

Augmented and Virtual Reality.

People ignoring what AR is may think first of VR as element of comparison. But before comparing both technologies, let us first define VR. Wikipedia says this:

"Virtual reality (VR) is an interactive computer-generated experience taking place within a simulated environment. It incorporates mainly auditory and visual feedback, but may also allow other types of sensory feedback." - [13]

Through this definition, we see that even though both AR and VR involve digital information and somehow an environment different from the real world, there are clearly two key aspects which differ between the two technologies.

The first one is that VR is a completely virtual environment, simulating the whole space, as well as noises and possibly even smells. It is designed to fool our senses and immerse us completely in this virtual world. AR on the other hand does not simulate a whole environment, but uses the existing physical world, extended with virtual elements.

Based on this observation, we can affirm that AR is not a subclass of VR. But recalling from our definition of AR that digital information is overlaid on the physical world, we could say that VR is a special case of AR, because it is overlaying the entire real world.

The second difference is that VR is not bound to a certain location in the real world, since it is not complementary to it, while AR binds the virtual objects to certain places of the real space.

And here is the point where we realize that, according to our definition of AR, VR is actually not a subclass of AR either. To make VR a special case of AR, we would have to remove the spacial requirement from the definition, which we could do. But we chose not to, because the geographical position of virtual objects in AR is an important part of this project, so an important part of the definition as well.

To conclude, although these concepts appear to be very similar and are partly related, there are some important differences between them, which justifies the fact that they were separated in two distinct domains of research since the 1990s. In fact, using our definitions, they are different enough that none of them can be described as a subclass of the other one.

A little remark concerning the technical dimension of the project: In the swift implementation of SceneKit and ARKit, many classes of ARKit are subclasses of SceneKit classes (for instance, the ARSCNView of ARKit is a subclass of the SCNView in SceneKit). This means that the implementation is sort of considering AR as a subclass of VR. However, this is only a practical implementation, probably the most convenient one for Apple developers, which does not need to conform to the theoretical definitions given above.

4.4. Assessment

By using the bottom-up approach described in the design section, the definition of AR is considering the definition of the individual words, a mathematical point of view and many possible criteria. This fulfills the first and second requirements set for the scientific deliverable.

Then, all knowledge gathered during the research was used to construct one definition of AR for this project, with three key criteria. This definition is inline with the definition found in the Apple documentation as well, and explains this way why the application developed in this project is an AR application. This definition is therefore adequate for this project, and can be used as reference in the whole paper. The third requirement is met.

Finally, the non-functional requirement is met as well. The comparison of AR and VR uses definitions of the two technologies to compare them. These are solid references to argument about their similarities and differences. Also, we have seen that none of them is a subclass of the other one, so they deserve to be considered as different research fields, and must not be confused.

5. ObDrop - A Mobile Application using Augmented Reality

5.1. Requirements

The application created in this project, called ObDrop, is an AR app, enabling the users to set virtual objects in the real space by touching the screen and to save these objects, such that when the user quits the application and restarts it later, the objects saved previously are recalled and placed in the space at the location they were set originally. Also, any user of the app must be able to see the objects set by other users.

The application is developed for iOS devices, using the frameworks SceneKit, ARKit and Core Location of Apple. The interface should be designed universally, such that the application can be deployed and used on any mobile device supporting ARKit. Furthermore, the interface must also be kept lightweight, since most of the screen must be free for the camera view.

In order to reach the goal, some functional requirements have been set for ObDrop. These are the criteria needed for the app to fulfill the goal of the project:

1) Design a lightweight and universal user interface.
2) Set a virtual object of the user's choice in the space by touching the real-world position on the screen.
3) Save the objects set in the space in a database on click on a save button, alongside with their positions, heights and rotation parameters.
4) On app launch, recall the saved objects from the database and set them back in the space at their saved real-world position, with the saved rotation and height.
5) Users of the app can see all objects set by all other users as well.

Additionally to these functional requirements, some non-functional requirements were set, to improve the application. These requirements are not necessary for the app to fulfill the goal, but they enhance a lot the user experience:

1) Implement plane detection, to track the surfaces of the real world, and generate invisible plane overlays on them, such that virtual objects can be set on these planes.
2) Implement some physics for the virtual objects of the space, like friction - adherence of the objects to the other ones - and restitution - potential energy kept by the object when it is dropped on the ground, such that it bounces or not.
3) In case an object is dropped at a place where no plane was detected, it will fall down to infinity. In this case, the object must be deleted.

Note that the AR sceneview has by default a gravity field attracting the objects down, and all objects have a default weight of 1kg, so these parameters don't need to be changed.

5.2. Design

The design of ObDrop contained two axes. The first one is the design of the layout of the application and the second one is the database structure.

Layout.

Since the application needs most of the screen to be free for the camera view, the interface has a quite simple layout.

The AR SceneView ("ARSCNView" element in the picture) is spanned over the complete screen and all other elements of the interface are overlaid on it, such that the camera view of the AR scene is the background of the app.

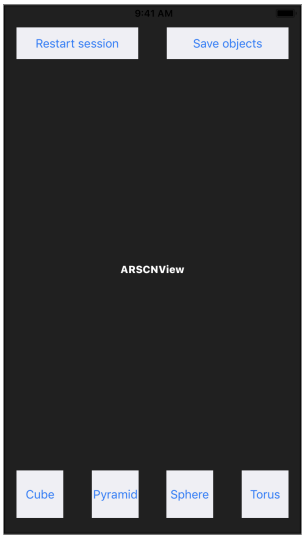
All features of the app are accessible using a few buttons. At the bottom, we find the buttons for selecting the shape we want to set in the space, while at the top, there is the save button and the restart session button. The buttons have an almost white background and a blue script indicating what they are made for.

The app must be designed universally, so most of the constraints were set relative to the other ones. For instance, the width of the buttons on the same line had to be equal to each other, and the space between them and with respect to the borders were fixed, so depending on the width of the screen, the buttons will be larger or thinner. However, some constraints still have fixed sizes, like the spaces between the buttons and the height of the buttons.

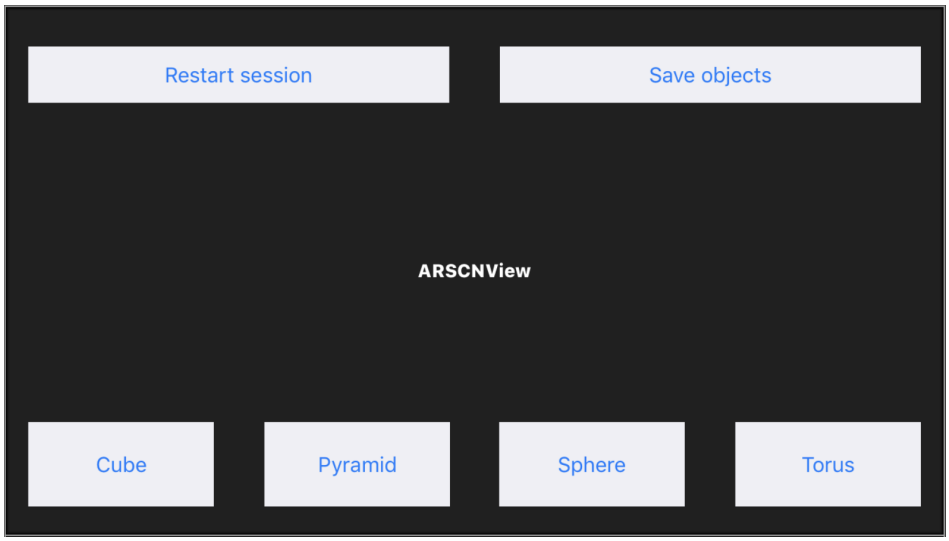
Using this method of scaling the buttons, the interface is adapting itself automatically to the size of the screens. This means that it will look fine on any device, no matter if it is an IPhone or an IPad, as well as regardless of the orientation, as we can observe in figure 3.

Database structure.

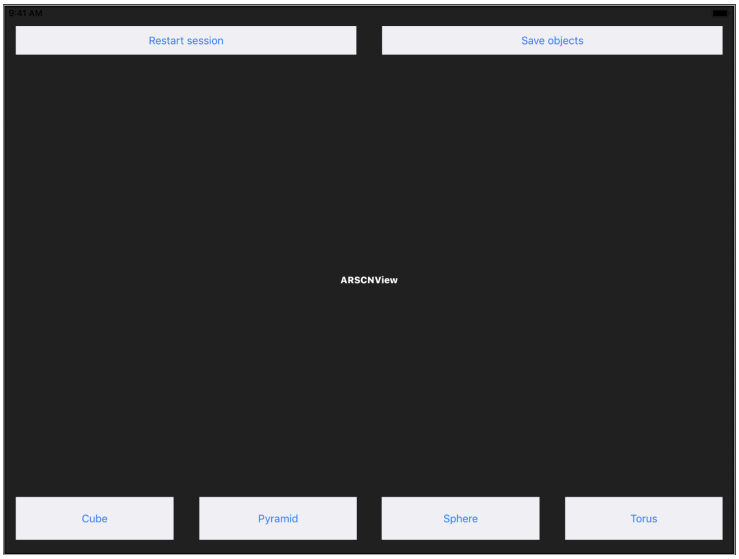
In this project, the best option for saving the objects is to use an SQLite database, because it enables to save a big amount of data. This way, there was no need to set a limit amount of objects to save.



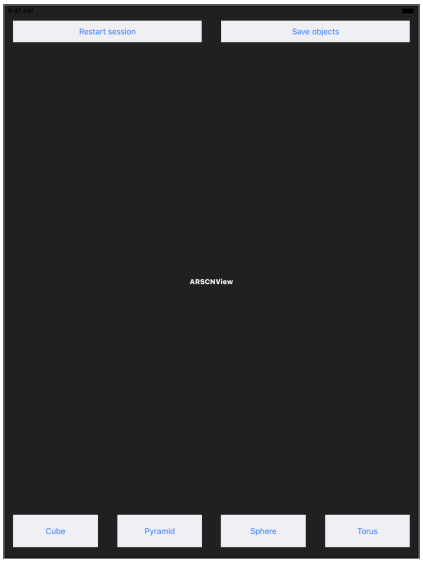
(a) iPhone vertical



(b) iPhone horizontal



(c) iPad horizontal



(d) iPad vertical

Figure 3: Layout of the app on different devices, vertical and horizontal orientation.

The structure of the database has been thought out and adapted many times during the project.

The first version was a database with two tables. One of them would hold the object types which could be set in the space (ObjectTypes) with all the corresponding properties, while the other one would contain all saved objects (PlacedObjects). Each row would indicate the type of the object, referring to an object of the ObjectTypes table, and its geographical location, i.e. latitude, longitude, altitude and orientation. The idea was here to implement a function called on launch of the app, which would recall the saved objects from the PlacedObjects table and, for each row, using the reference key for the type of the saved object, find the object properties in the ObjectTypes table to create a corresponding object set at the given location.

The general idea was fine, but the problem of this approach was that it needed a lot of computations on each launch of the app and each time the user would save an object. Furthermore, due to the lack of precision of the GPS location system, it was even more difficult to have a precise position for the saved object. Therefore, saving each object's position individually could lead to a huge rearrangement of the objects in the space when recalling the objects back later, since none of the object's location was precise enough. For instance, if the user set two cubes next to each other and saved them, he could find them later being 20 meters away from each other, which is absolutely

not convenient. Therefore, we needed to improve the first version of the database and create a second one.

We cannot completely solve the precision issue, because it depends mainly on the current position of the user, the devices hardware and the satellites position in the sky at the time the app is used, and none of these factors can be influenced by the app itself. However, in the second version, we can at least limit the amount of computations and keep very precise positions of objects set during the same session relative to each other.

This improvement was achieved by adding a new table in the database, OriginNodes. Also, the former PlacedObjects table was now renamed into ChildNodes. As we can easily deduce from these names, the OriginNodes table is meant to hold the origins of each session started, and the ChildNodes table will hold all objects set in the space, with indication of the origin node they are bound to.

At the start of each session, the application automatically saves the geographical position of the user into the OriginNodes table: it corresponds to real world position of the world origin for that session. Then, whenever a user saves an object he has set in the space, instead of saving the approximate real position of the object, it will just save the position of the object (child node) inside the 3D coordinate system of the AR session, i.e. the position relative to the current origin. When the user leaves the app and opens it again later, the application will first save the new origin, then calculate the position of all origin nodes saved previously to set them in the space, and finally place all objects of the ChildNodes table relative to their corresponding origin node.

This way, the origins may not be placed very precisely due to the lack of precision of the GPS location system, but at least all objects created during a given session will be placed well relative to each other.

Figure 4 shows a schema of the final database structure. It shows the three tables, with all their columns, as well as the type of data they can contain.

As explained above, the ObjectTypes table contains a list of objects, with their properties. The table is designed to contain the shape itself and the texture. Each object has a unique name, which is used to reference to it. They also have a unique ID, but it is not used in this application.

Apart from the unique ID which is automatically generated, the OriginNodes table contains only latitude, longitude and altitude, no orientation anymore. This last property was removed, because we implemented in the code that the world origin of the AR session should be oriented along to the compass heading and gravity of the earth.

The ChildNodes table contains one column to indicate the origin it belongs to and one to save the object type. Along with these two columns and the unique ID (which is unused), there are other 16 columns in the table. Each of them contains a double value corresponding to one entry of the so-called worldTransform matrix. This 4x4 matrix contains all information about the position and rotation of the object, so the easiest way to save the exact properties of the objects was to save the whole matrix in the database.

5.3. Production

The production of ObDrop can be divided into three parts.

The first part concerns the interactive creation of objects with physical properties, such that for instance, on touch of

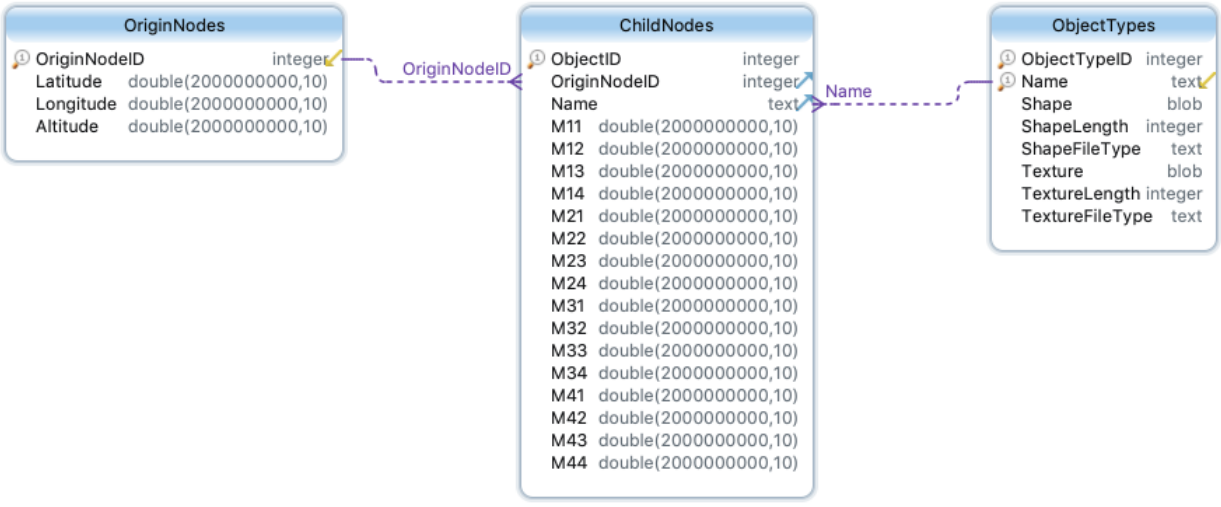


Figure 4: Database schema.

the screen, a virtual object of the chosen type is set in the space. Related to this part comes also the feature to detect surfaces of the world and create overlays for them, to be able to set objects on these planes.

Second, the database must be created and the saving features implemented. Along with this comes the automatic saving of the real world location of the current origin, as well as the saving of the objects set in the space by the user.

Finally, the objects must be retrieved form the database and placed in the AR world dynamically on launch of the app, which is the last feature to implement. This part also includes the real world position calculation of the previously set origins, which uses the Haversine formula and another, closely related formula for the bearing, with some linear algebra.

Objects and physics.

The feature to create objects had to be implemented in such a way that the user could choose the type of the object he wanted to set, and drop it by touching the place on the screen.

As already explained in the design section, the interface has some buttons at the bottom to choose the shape we wanted to create. More concretely referring to the code, when we programmatically want to create an object, it needs a geometry and a texture. In the code, there are some global variables which we use to define the geometry and texture used when creating objects. These variables are changed according to the button tapped on the screen. For instance, touching the button "Sphere" will set the variable "geometry" to a so-called SCNSphere, and an earth texture will be passed to the variable "texture".

Each time the user touches the screen, the program uses the hit results to obtain the SCNVector3, the coordinate vector corresponding to the place touched on the screen, at which a new object will be set. Then, the function "createObject" is called to create it.

This function creates an object using the specified global parameters "geometry" and "texture" and the SCNVector3 obtained. More precisely, it creates a new SCNNode with the specified geometry and texture, which is then attached to an origin node previously created at position $(0, 0, 0)$ (refer to the subsection "Database and saving features" (5.3) for more details). The new node has the name of its corresponding object type in the database, to be able to identify its object type during saving.

As already explained in the technical background on SceneKit, each node has properties which affect it's appearance, but needs other properties to be able to physically interact with other nodes. Therefore, when the node is created, the function also creates a so-called physicsbody for it, otherwise no physical forces could be applied to it. This is the invisible 3D shape for which actions can be defined if another object enters this area. Furthermore, the friction, restitution and rolling friction are set to some values which approach the reality a bit more. Since all nodes have a default weight of 1 kg, this parameter did not need to be changed.

Because the AR sceneview has always a gravity field simulating the real earth gravity, adding physicsbodies to the created nodes automatically caused them to be attracted down, but since there were no virtual surfaces on which the objects could possibly land, they just kept falling down to infinity. This is the reason why we had to implement another feature into the application.

ARKit enables automatic horizontal and vertical planes detection in only one line of code. However, it does not automatically create these planes, it will only set plane anchors where it has detected enough feature points to affirm that there is a surface there. For this reason, we had to implement a function which is called whenever an ARAnchor was added to the space, and creates an invisible plane at this place. These planes also need physicsbodies, to be able to hold the objects dropping down on them, but we don't want the planes to be affected by the gravity field or pushed around by the other objects. Therefore, we specify a property of the physicsboldies of the planes, which is called "kinematic". This will enable the planes to affect other objects behaviour without being themselves affected by any physical forces.

Once a plane is created, it may actually be overlaying only a part of the whole surface, because the device did not detect more of it. Later, when the user moves around, maybe the device will detect more of it, in which case, ARKit will update the ARAnchor. When it is updated, another function is called, that makes sure that the corresponding plane and its physicsbody are immediately extended to the newly detected surfaces.

Finally, when the user is dropping objects in the space, it can happen that some objects don't land on a detected plane and therefore fall down to infinity. In case a non-saved object is situated too far under the world origin, a feature is called which makes the object fade and removes it from the space.

Database and saving features.

On each launch of the application, aside from the standard AR session configuration, a bunch of database related operations are made as well.

First, the app is looking in its document directory to see if the database file ("ObDropObjectsDatabaseVersion3.sqlite") already exists. In case it does not exist yet, it is automatically created and opened. If it already exists, it is just opened, such that we can start using it. Then, the app checks if the tables already exist or not. If they don't exist yet, they are created as well, following exactly the structure described in section 5.2. Finally, the objects in the ObjectTypes table are inserted. In case they already existed (identified by the unique "Name") inside the table, the rows are replaced. This is not always necessary, but it ensured to update the objects to their latest version, since they may have been changed through the code. If the database did not exist previously, it is now in its initial form. And should

it already exist before, it is now updated to the latest version.

The next step is to save the current origin position, because this will be the reference point to which we will save all the objects set in the space during this AR session. At the very first time the application is launched, it will ask the user to allow to use the camera (required for AR) and to access its location. Once authorized, each time the application is launched, it will start to update the location of the user. After each update, a function called "locationManager" is automatically called. In this function, we implemented a check to know whether the location obtained is precise or not, and the function return as long as it is not precise enough. Once we have a good location, the location update is stopped. The last measured latitude, longitude and altitude are saved in the OriginsTable, and the unique ID of this new origin is retrieved, such that it can be inserted as origin reference when saving a child node. This origin will now be referred to as the current origin.

On the interface, there is a button which on press saves all the objects newly set in the space. This button is by default disabled at launch of the app, and is only enabled once the current origin is saved in the database. This way, the user knows that he has to stay in place and wait for a precise localization when it sees that the button to save objects is disabled, and this also avoids to save objects without a corresponding origin.

Once the button is enabled, the user can press it to save all objects he has set in the space that are not already saved in the database. To be able to distinguish between the new objects and the previous ones (as well as the invisible planes created), a simple trick was used. At launch of the application, a node without any visible geometry (the originNode) is created exactly on top of the world origin, so at position $(0, 0, 0)$, to simulate the current origin. All objects that the user is setting in the space by touching the screen will be attached to this originNode, while all previously saved origins (with their corresponding child nodes) will be attached to the world origin (or rootNode) directly (refer to the section "Retrieve and render saved objects" (5.3) for more information). When the user is asking the app to save the objects, the function called will only save the child nodes attached to the originNode, and none of the other nodes in the space.

The function to save the new objects (saveChildNodes) is taking all child nodes of the originNode and saves them one by one by inserting a new row to the ChildNodes table for each of the node. Each row contains the current origin ID retrieved after saving the current origin, the name of the object type (indicated by the name given to the node at its creation) and the 16 enteries of the worldTransform matrix of the node. As a recall, the worldTransform matrix contains all information about the position and rotation of the object relative to the world origin. Here, we had to take care to consider the worldTransform of the presentation of the node, which is the position and rotation of the node we see on the screen after animations and physics have been applied to it. Otherwise, we would save the initial position and orientation of the object, when the user tapped the screen to set it, which is not what we want.

Once a node is saved, its physical properties need to be changed, because we don't want it to be affected by physics anymore. However, it should still be able to affect other object's behaviour. Therefore, we set the same physics settings as for the planes, i.e. "kinematic". Finally, each saved node is removed from the originNode and attached to the rootNode, to avoid saving it again the next time the save button is pressed.

Retrieve and render saved objects.

In the previous subsection, we explained that the current origin was saved in the database as soon as the app got a precise enough location. Immediately after this is done, the app can place all previously saved objects in the space, with respect to the current origin position.

The structure of the database is designed such that we only need to calculate the real world location of the origins. Afterwards, the child nodes will just be attached to their respective origin and placed using their worldTransform matrix.

However, to place an origin at its real world location, we cannot just pass in its latitude and longitude saved in the database, we have to tell the app its coordinates in the 3D vector space of the current AR session. But the world origin, which has coordinates (0, 0, 0) is placed at the devices location at launch time, and therefore changes each time the app is launched or the AR session is restarted. This is the reason why we need the position of the current world origin. Using this position, we can calculate the coordinates of the other origins.

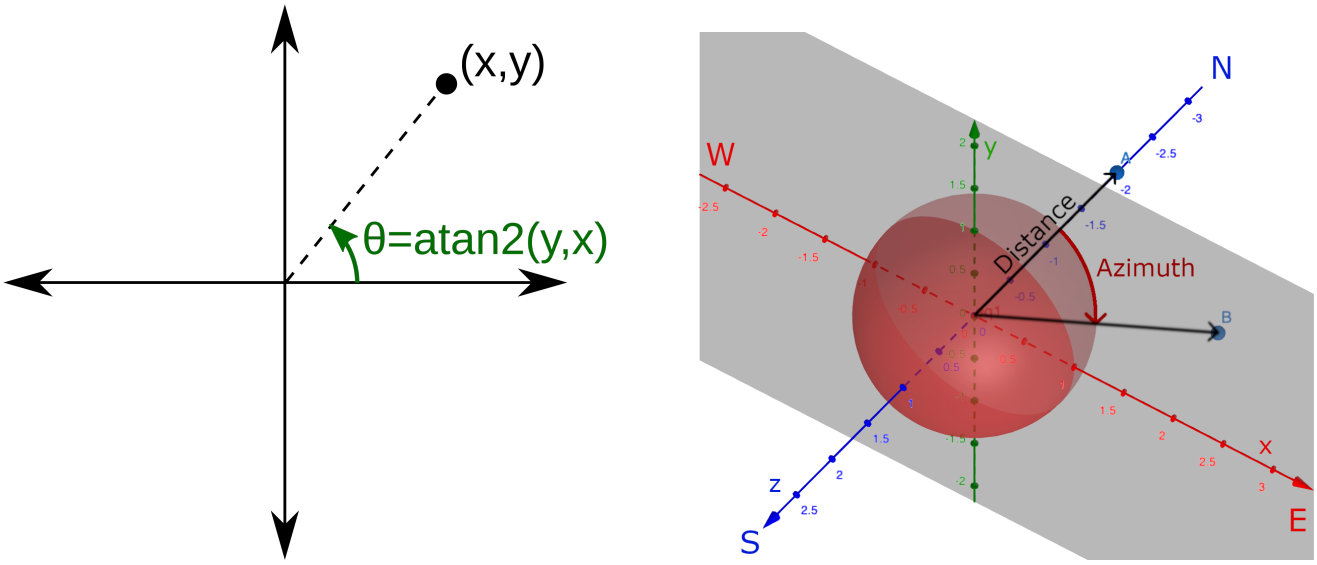
Using an SQL statement, we retrieve all origins but the current one from the database. Then, for each origin, we calculate its distance from the current origin, as well as the azimuth angle, which is the bearing between the current origin and the saved one, with respect to the true north. Here is a little more explanation about these calculations:

The Core Location framework of Apple has a method which calculates the distance between two locations, given their latitudes and longitudes. This method uses the Haversine formula, a formula using trigonometry to calculate the distance between two points indicated each by two angles (the latitude and longitude) on a sphere, including its curvature (the great-circle distance) [15]. The Haversine formula is:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\tfrac{\varphi_2-\varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\tfrac{\lambda_2-\lambda_1}{2}\right)}\right)$$

Where point $1 = (\varphi_1, \lambda_1)$, point $2 = (\varphi_2, \lambda_2)$ (the points are given by (latitude, longitude)) and the distance is d [16].

To calculate the true north based azimuth of the two points, we use another formula that returns this



(a) Arctan2.

(b) Rotation of the origin around the y axis.

Figure 5: Placing the origins in the space.

angle in radians, for which this time there is no default implementation in swift. The formula used is:

$$x = \sin(\lambda_2 - \lambda_1)\cos(\varphi_2)$$
$$y = \cos(\varphi_1)\sin(\varphi_2) - \sin(\varphi_1) * \cos(\varphi_2)\cos(\lambda_2 - \lambda_1)$$
$$\alpha = \arctan 2(x, y)$$

Where point $1 = (\varphi_1, \lambda_1)$, point $2 = (\varphi_2, \lambda_2)$ (all in radians) and the azimuth is α [17]. The arctan2(x,y) (or atan2) function takes two arguments and returns the angle in radians between the x axis and the vector to point (x,y), as showed in figure 5a.

Once we have the distance (d) and azimuth (α), we can create the coordinates for the origin position. Therefore, we create an SCNVector3 with coordinates (0, 0, -d), that is located at the calculated distance from the origin and points to the north (remember that the coordinate system of the AR space is oriented to the cardinal points and gravity). Now, we build a rotation matrix that rotates the space clockwise around the y axis by the angle α. This matrix looks like this:

$$\begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

We just have to multiply the SCNVector3 by this matrix to rotate the point around y and place it correctly with respect to the x and z axes. Figure 5b is a schema of the described computation.

Finally, the last thing to do is to set the origin at the right height. Since the unit used for the AR space is meters, we only need to substract the current altitude from the altitude of the origin we want to place, and set the result as the y parameter of the vector. This vector is set as the position of the origin node and we give its ID as name, to be able to refer to it later.

Note that in case the distance is greater that 50 meters, it is implemented that the app should not place the origin in the space, because it is just too far away to be worth it.

After having placed all origins in the space, we can now retrieve all saved child nodes from the database and place them as well. For each child node retrieved, using the object type indicated, the corresponding geometry and texture of the node is set. Then, we set a kinematic physicsbody of the corresponding shape for each of them, such that they stay in place (since the planes they were set on do not exist anymore) in the space, but can still affect other objects around. Furthermore, we recreate the 4 by 4 worldTransform matrix by plotting in the entries saved in the database and apply it to the node to position and rotate it correctly. Finally, using the ID saved in the database, we bind the node to the origin with the indicated ID.

It may happen that a child node retrieved cannot be bound to its origin, because the origin was not set in the space (because it was more than 50m away). In this case, the object is not attached to any node and won't appear in the space either.

5.4. Assessment

Looking at the requirements set at the beginning of the section, we observe that almost all of them were fulfilled. Points 1) to 4) of the functional requirements, as well as all non-functional requirements were correctly implemented in the application built in this project. The only point which does not seem to be satisfied the fifth functional requirements, because the application creates a local database, which is therefore only accessible locally. This implies that

other users of the app cannot see and recall the objects set by another user, contrary to what was required.

However, to fulfill this last requirement, there is only one more step left to do, which is setting the database on a server. In the code, it would only change one line: the location of the database. Therefore, even if it was not done, we can still consider this requirement to be fulfilled.

Independently from the requirements, a further useful feature was implemented in the app. On the interface, there is a button called "Restart session". To this button is bound a feature which pauses the AR session, removes all nodes from the AR space, resets the variables related to setting objects in the space, then calls all events occurring at launch of the application (database creation/update, updating location, create origin node etc) and reruns the configuration of the AR session. This is equivalent to closing and reopening the app. It is useful to update the world origin and the current location, in case the user has moved more than a few dozens of meters away from his original position. When the user taps this button, an alert is displayed first, asking if he really wants to restart the session, knowing that all non-saved objects set in the space won't be saved. This way, he does not risk lose any information he wants to keep, because he can cancel this event.

Acknowledgment

I, Iris Kremer, would like to thank my project academic tutor Jean Botev for all the time he invested for me and his help during the project.

I would also like to thank my family for their constant support and love.

6. Conclusion

In this paper, we have presented a bachelor semester project which aimed to explore the field of AR and develop a mobile application using AR, called ObDrop. Since all requirements have been reached and we could even implement an additional, useful feature, this goal is fully reached, so the project can be considered as a success.

Nevertheless, both scientific and technical deliverables could be improved further.

The scientific deliverable gives a detailed definition of AR, but does not consider the possibility to include removing/masking real objects from the space in the definition. Another further research axe would be to carefully determine which technologies are part of AR and which ones are not. For example, are holograms AR? Do HoloLenses provide AR experiences, or is this more mixed reality? This last topic mentioned could also be subject of an analysis and comparison to AR.

The technical deliverable could be improved by adding some more useful features, like a feature to delete a saved object by selecting it. Also, there are currently only a few object types available in the database, some more objects could be added to it. Concerning the object types, we also have the possibility to save an scn (SCNSceneSource) file containing a custom shape in the database. There is even a field to indicate the file type, such that it could be handled properly when it is retrieved. However, we did not implement the retrievement of this type of files in the code yet, so this part should be completed as well. Finally, since the precision of the application is still not very good, despite the efforts made to reduce the precision range. There may be ways to increase this precision further and improve the app this way.

References

[1] BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global University of Luxembourg, BiCS - Bachelor in Computer Science (2017).

[2] Bachelor in Computer Science: BiCS Semester Projects Reference Document. Technical report, University of Luxembourg (2018)

[3] B. Jiménez Fernández-Palacios, F. Nex, A. Rizzi, F. Remondino, ARCube – The Augmented Reality Cube for Archaeology. 2015. *Archaeometry*, 57, 250-262. Available online: https://onlinelibrary-wiley-com.proxy.bnl.lu/doi/full/10.1111/arcm.12120?pds=265201913574411543949928213988933 5

[4] Sarah Murthi and Amitabh Varshney. *How Augmented Reality Will Make Surgery Safer*. Harvard Business Review, 2018. Available online: https://hbr.org/2018/03/how-augmented-reality-will-make-surgery-safer

[5] Release 2.0 of ARToolKit Professional for iPhone and iPad2. Retrieved May 2019, from AR Blog: http://arblog.inglobetechnologies.com/?p=397.

[6] Dieter Schmalstieg and Tobias Höllerer. *Augmented Reality: Principles and Practice.* Addison-Wesley Professional, 2016. Available online: https://proquestcombo-safaribooksonline-com.proxy.bnl.lu/book/web-development/usability/9780133153217

[7] Cambridge Dictionary. dictionary.cambridge.com

[8] Craig, Alan B. *"Understanding Augmented Reality : Concepts and Applications"* Elsevier Science & Technology, 2013. Available online: https://ebookcentral.proquest.com/lib/unilu-ebooks/detail.action?docID=1183494.

[9] Apple presentation of ARKit. https://developer.apple.com/arkit

[10] Apple presentation of SceneKit. https://developer.apple.com/scenekit

[11] Apple documentation on ARKit. https://developer.apple.com/documentation/arkit

[12] Apple documentation on SceneKit. https://developer.apple.com/documentation/scenekit

[13] Virtual reality. Retrieved on March 4, 2019, from Wikipedia: https://en.wikipedia.org/w/index.php?title=Virtual_reality&oldid=886005076

[14] Retrieved May 2019, from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Arctangent2.svg

[15] Akshay Upadhyay. Haversine Formula – Calculate geographic distance on earth. Retrieved April 2019, from GisMap: https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/

[16] Haversine formula. Retrieved on May 18, 2019, from Wikipedia: https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=897532696

[17] Akshay Upadhyay. Formula to Find Bearing or Heading angle between two points: Latitude Longitude. Retrieved April 2019, from GisMap: https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/

7. Appendix