IceAndFiAR
Augmenting Game of Thrones

Felix Gruber*

University of Applied Sciences Hagenberg, Mobile Computing

Abstract

This document describes the design and implementaiton of a 3D augmented reality visualization for handheld mobile devices using Unity in combination with Vuforia. It uses marker based tracking to present virtual overlays on mentioned markers with the data sourced from a fan-made website that allows tracking characters of the popular TV show "Game of Thrones" by HBO and shows them on said map.

Index Terms: aumgented reality—unity—vuforia—game of thrones

1 Introduction

Augmented reality (AR) is gaining popularity and foothold in the mainstream market and with companies like Apple and Google releasing ARKit[1] and ARCore[2], respectively, an increasingly large number of people will potentially become interested in the topic and look into it. This being stated, the adaption of the technology can make sense in various situations and under certain conditions, where it can be easy to augment already existing information with digital aids. One area lies in the field of maps and their visualization. Using a conventional, static map as a base layer to improve and build upon, the additional information can be presented in a more interactive, modern way to allow for improved user interaction. For this project HBO's successful TV series "Game of Thrones"[3] with their fictional world was chosen due the authors personal interest in the show, the already existing, fan-made website to track the characters over the episodes, shown in Fig. 1, and the suitability of the mentioned map for 3D augmentation.

2 Design and Requirements

Starting the project firstly required analyzing the website, shown in Fig. 1, to find out how user interaction is supposed to take place and which elements of the GUI will be considered to be reused in the AR app. The large, beige sidebar with check-boxes for each character was going to be mostly irrelevant for the app, since for the first prototype a selection of the individual characters was not proposed. Alternatively the behavior of adding paths to already existing ones was changed to instead only show the path of the current episode, skipping the ones beforehand, but always showing those of all characters, as mentioned. One prominent feature was adopted to also be used in the app, that is also present on the website: the slider for selecting the episodes and therefore the paths to display.

Besides the paths of the characters it was chosen as a requirement to put a small, unobtrusive 3D object on the map for each of the clickable locations, which should open some form of text label once touched, which should also be possible for the paths.

*e-mail: iceandfiar@felixgruber.net

[1] ARKit — Apple Developer — developer.apple.com/arkit
[2] ARCore — Google Developers —developers.google.com/ar/discover
[3] Game of Thrones — HBO — hbo.com/game-of-thrones

Figure 1: Screenshot of quartermaester.info [1] with its user interface.

This project was developed alongside the lecture "Augmented Reality", in which Unity[4] in combination with Vuforia[5] was taught, so both of the mentioned technologies were used.

3 Implementation

3.1 Data acquisition and processing

Starting with the implementation the first step was to find out what the data sources for the information concerning locations, characters and the paths were and if they could easily be reused in the app. Luckily the developers stored all the required information in a rather large, plain JavaScript (JS) file that can be accessed publicly. In order to bring the data to Unity and therefore C# it was decided to take an intermediate step and use JSON as a data format. Mentioned file was used to extract all the required information before processing it any further, which turned out to be easy for some parts, like the episodes[6], but way more complicated for others, like the paths[7]. JS is able to output data in a JSON compliant format natively, but since the project is implemented in C# a framework, capable of executing JS code, was needed. The chosen library for this task was Jurassic[8], which later turned out not to be implemented for cross-platform use,

[4] Unity3D — unity3d.com
[5] Vuforia — Augmented Reality — vuforia.com
[6] `JSON.stringify(episodes)`
[7] github.com/grubFX/IceAndFiAR/blob/master/Assets/Resources/pathScript
[8] Jurrassic — github.com/paulbartrum/jurassic

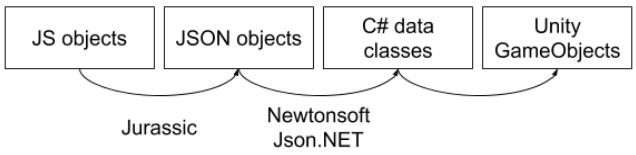

Figure 2: Proposed workflow for acquisition, conversion and presentation of data.

so the conversion at runtime was abandoned and instead performed once, saving it's output to use for further processing. The proposed workflow can be seen in Fig. 2 where the described aspect covers the transition from the first to the second box.

Advancing further in the workflow meant finding a way to bring the data to C# and store it in what will further be referred to as "data classes", whose sole purpose is containing data that was previously converted to JSON. Creating these was done with the help of an online tool [9] which takes JSON formatted data as input and will output the appropriate code in the chosen language. These data classes represent the third box in Fig. 2 which meant that the next step was finding a framework to convert the actual information from JSON and fill them. This was accomplished by using Newtonsoft's Json.NET library[10] which is also available for use on Android, which will be the primary platform the final app shall be run on, amongst other platforms.

3.2 Data visualization

Using Vuforia's image tracking capabilities it was decided to use the actual, fictional map, depicted in Fig. 3, as a tracker. So in order to visualize the acquired data and show paths and locations as an overlay it was first tried to arrange a scaling value for the `GameObject`[11] called "ImageTarget" inside Unity, that would match the locations printed on the map as closely as possible to the digitally added ones. It was discovered that the coordinate system uses some kind of nonlinear scaling or map projection, resulting in unwanted distortion, mainly along the vertical axis. Accounting for this lead to the development of an appropriate scaling function as described in Sect. 4.

Having tackled the scaling problem the next steps in drawing the actual overlay data mainly consisted of two parts that needed to be drawn.

1. **Static locations**: these always need to be drawn, as soon as the marker is detected. These objects should not distract from the main objective of showing the paths but nevertheless be selectable when chosen.

2. **Paths of the characters**, according to the chosen episode: these will be selected by the slider on the right side. To highlight the actual path it was decided to draw spheres, that were bigger than the objects representing static locations, and further also colored. A `LineRenderer`[12] was used to draw a line between each of the points along the path, also colored the same way as the related points of one single character, to highlight their correspondence.

3.3 POI selection via Raycast

In order to find out if a user touched a virtual GameObject on the screen the use of Raycasts looked like a viable solution. Whenever a touch is registered on screen a virtual ray is cast from the point on the screen/camera in its looking direction. When this ray hits an object that is not part of the label itself, it will further display the name of the hit GameObject on the label. Rotation of the label is explained in the following Sect. 3.4.

[9] quicktype.io
[10] Json.NET — Newtonsoft — newtonsoft.com/json
[11] docs.unity3d.com/Manual/class-GameObject
[12] docs.unity3d.com/Manual/class-LineRenderer

Figure 3: "The Lands of Ice and Fire" – Speculative World Map [2] used as marker.

3.4 Automatic label rotation

Label rotation was implemented with the goal of the label automatically facing camera direction of the viewport. Since the label is shown after touching an object in virtual space, as described in Sect. 3.3, it will keep its rotation locked, relative to the marker, until the screen is tapped and the sent Raycast did *not* hit a GameObject. This triggers the rotation of the label to be set accordingly, with a 45° "backwards" tilt between it and the marker for increased readability.

The final result of the implementation can be seen in the screenshots depicted in Fig. 4.

4 Challenges

As briefly mentioned in Sect. 3.2 one challenge was to figure out what kind of coordinate system, map projection and scaling function was used by the developers of quartermaester.info and apply it in order to achieve corresponding matches between locations printed on the map and the digitally added ones. After trying out a uniform scaling factor and quickly realizing that the results were far away from optimal another approach was chosen. Therefore simple, static cubes were added on top the "ImageTarget" in Unity for a few chosen locations and their coordinates where entered into Table 1. The first two columns represent latitude and longitude of the locations, taken from the JS source file, columns `z` and `x` show the respective coordinates inside Unity while the values in last two columns were calculated from them.

Looking at both of these factors reveals the nonlinear nature of the scaling factor for the z-direction inside Unity, affecting scaling of latitude values. Plotting an x-/y-diagram for `factor0` over `latitude`, depicted in Fig. 5, further allowed to add a trendline which reveals the equation, that the scaling-factor is following. This being stated there are some small inaccuracies, since the positioning of cubes

Figure 4: Screenshots of the final prototype.

lat	long	z	x	fact0	fact1
-73.503	-130.928	-153.300	-183.400	2.086	1.401
-58.700	71.792	-101.100	99.520	1.722	1.386
-45.491	80.449	-70.800	112.000	1.556	1.392
-18.021	71.836	-25.760	100.000	1.429	1.392
4.083	-59.165	5.660	-82.500	1.386	1.394
4.478	-5.508	6.100	-7.700	1.362	1.398
29.420	-84.214	43.100	-117.200	1.465	1.392
56.680	-114.097	95.900	-158.800	1.692	1.392
75.995	8.467	167.300	11.550	2.201	1.364
78.721	-97.156	184.660	-135.120	2.346	1.391

Table 1: A table of various locations with their “real” latitude and longitude, their coordinates inside Unity along with the resulting scaling factors for both axes.

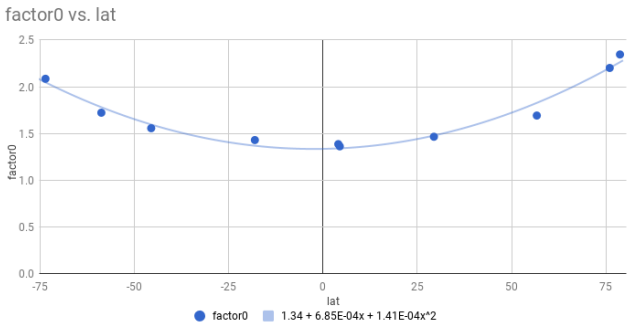

Figure 5: A plot showing factor0 over latitude with a trendline through the existing data revealing the approximate scaling function.

on the markers is done by hand, which affects finding the “correct trendline” to a certain degree resulting in slightly more misaligned location objects, the further away their position is from the origin in z-direction. This is one of the possible improvements, mentioned in Sect. 5.

5 Possible Improvements

The following points mark possible improvements in the future:

- Repositioning the points on the map more precisely to recalculate the scaling function for a more accurate match.
- Enabling left-handed mode with the slider on the left side of the screen.
- Finding bounding boxes around each connected path and centering the label to its relative midpoint.

The latest source code can be found on the authors GitHub profile[13] where the whole project is published under a permissive open source license.

Acronyms

AR augmented reality. 1

GUI graphical user interface. 1

JS JavaScript. 1–3

JSON JS object notation. 1, 2

References

[1] quartermaester.info.
[2] A song of ice and fire – speculative world map.

[13] github.com/grubFX/IceAndFiAR