Cuby
An Augmented Generative Arts Application

Marlene Mayr, Anna Moser

September 05, 2020

Abstract

Cuby is an augmented augmented reality application for mobile devices, designed for offering users a visual snapshot of their environment. The project combines augmented reality, generative arts and device input processing. The application generates a structure from parameters specific to the user's environment. The structure's base is a cube, textured with a height and satellite map according to the users location. Surrounding particles change its shape according to the ambient noise and its color based on colors captured by the camera. Using and adjusting these individual parameters, each user is able to generate a unique and personal *Cuby*. The project is realized with Unity and Unity's AR Foundation[1] package to cover both iOS and Android based devices.

[1] https://unity.com/unity/features/arfoundation

Contents

Chapter 1

Aims and Context

1.1 Initial Focus

The initial focus was to represent the users environment in a meaningful way, where it is to be determined what meaningful might look like. Another main focus was how to realize an AR application on a mobile device, working with the users device data, such as time and location.

1.1.1 Device Data

The main focus of the project is capturing the users environment in an abstract structure in augmented reality. To achieve a unique structure for every user, one goal was to determine which data could be fetched from the user's device and how this data affects the structure. The data should be processed and presented on Cuby in a meaningful way, therefore the users should be able to easily comprehend changes in the data and view them on the structure immediately. Additionally, the user should be given control to alter the parameters to his preferences to some extent. It was decided to base the structure on the user's location, a sound sequence of the ambient noise, the colors in the room (captured by processing a snapshot of the camera) and the creation date and time.

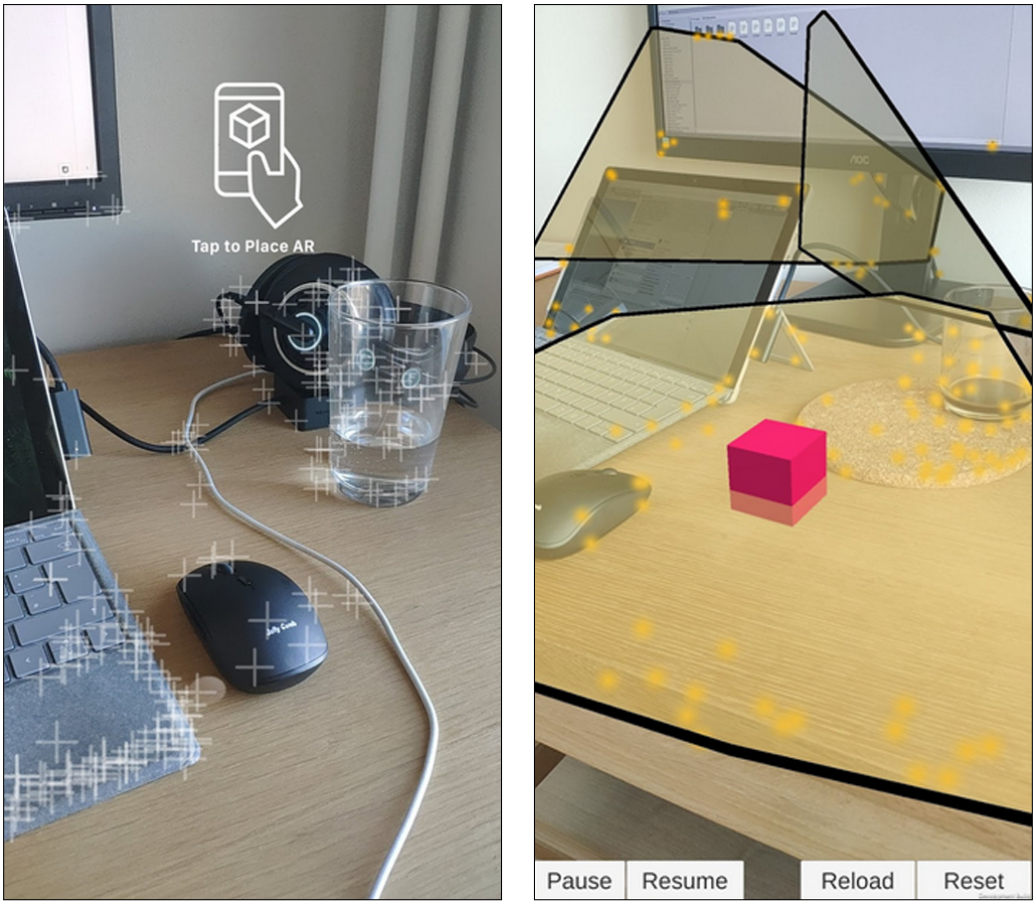
1.1.2 Visual Representation

A frequently asked question was how it is possible to capture a room's vibe and which characteristics it includes. One major difference is the overall color scheme of a room. As represented in image **??** two rooms show completely different color palletes and, therefore, a different mood and feeling. Furthermore the ambient sound of a room changes the mood of an environment. One main aim of this project was to capture all these characteristics of a room and visualize it in a way so that a user gets a proper impression of a Cuby from another person.

1.1.3 Technical Starting Point

As research on similar projects revealed, generative arts is not frequently represented in AR applications, especially not for mobile devices. This fact formed the initial mo-

tivation for the project, which consolidated in generative arts, augmented reality and device specific input processing. Mobile development with Unity and, therefore, addressing limitations of smartphones' performances posed an open issue. During the search for a appropriate plugin for realizing such a mobile AR application, ARKit, ARCore, Vuforia and AR Foundation competed against each other. Since the application should support both, Android and iOS devices, Unity's AR Foundation package was chosen. AR Foundation includes core features from ARKit, ARCore, Magic Leap and HoloLense and enables deploying across multiple mobile devices.



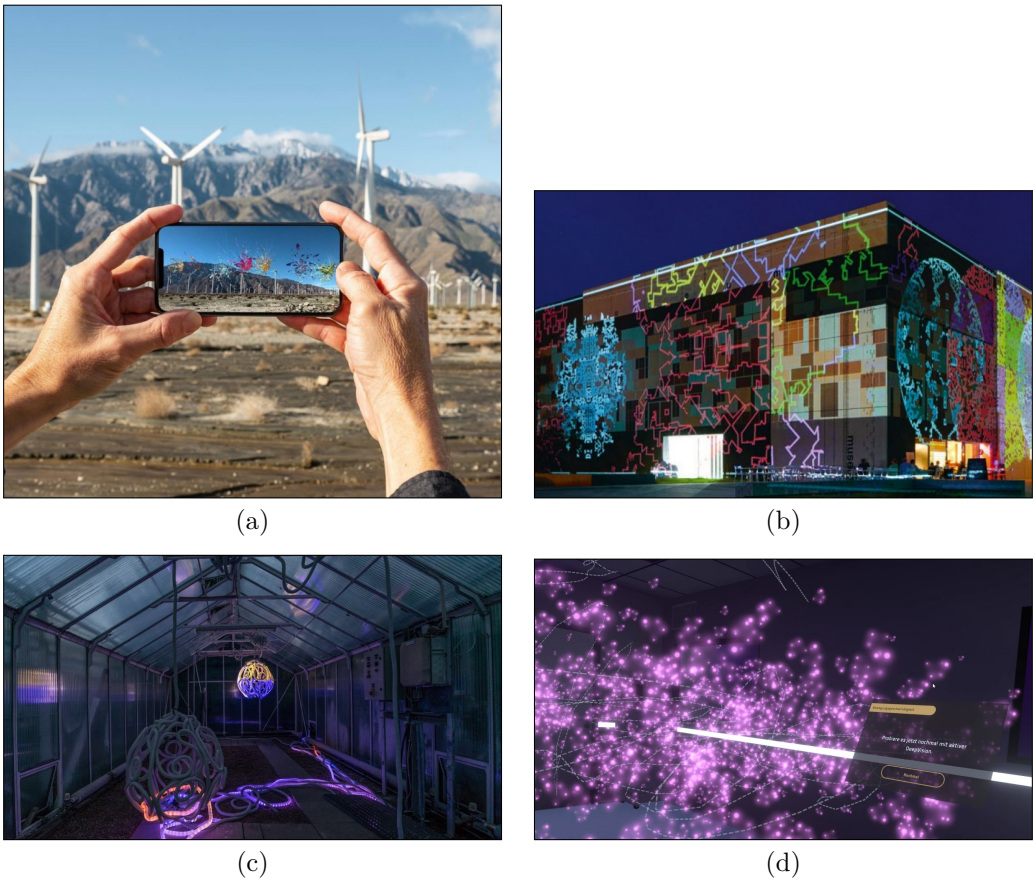
(a) (b)

Figure 1.1: First steps with AR Foundation.

1.2 References

One project references was *ARBrush* [1], which realized a three-dimensional drawing tool with the ARKit plugin for Unity. Another inspiration was an AR art project called *Moto wall* [6] in figure 1.2, which extends a mural wall with different shapes in AR on mobile devices. Especially the user interaction with these application raised awareness and provided a solid discussion base for Cuby.

Figure 1.2: Moto wall is an AR art project, where users can view and walk around 42 digital variations of basic shapes. Image Source: [6]



(a) (b)

(c) (d)

Figure 1.3: (a) shows the 4th Wall AR App, which invites users to discover drawings in AR by Nancy Baker Cahill [2]. Image (b) is a visual concert projected onto a wall [8]. The students project in (c) shows an art installation [7] and (d) a visualization in a medical VR application from Innerspace [4].

Chapter 2

Project Details

Initially, we tried out some AR Foundation examples[1]. Playing around with a few existing scenes provided us with better understanding of the package, for example the plane and feature detection as well as using raycasts for the interaction with these surfaces. Figure 1.1 shows two of these basic applications. Having the project goals in mind, at this point we decided to proactively switch to Unity's newer Universal Render Pipeline[2] (URP) which is compatible with its new Shader Graph[3] and Visual Effects Graph[4] packages. It was said that they reduce performance or on the other hand improve on the visual outcome and URP is optimized for mobile development.

The first technology that was added to the project should assist with demonstrating the coordinates of the smartphone. With the assistance of the Mapbox API[5] for Unity, the user's location is used to query a height and satellite map in various zoom faactors, which are combined into a material used on the main cube. The cube comes with a shiny black, metal like texture by default, which changes as soon as the user enables the location in the menu.

The heart of the structure is covered in two kinds of particles. One particle system gets influenced by the soundscape recorded by the user. A sound wave is generated and the particles react accordingly. The other particle system represents the age of Cuby by extending their trails with increasing age. The latter particles first appear in a default color and change as soon as the user activates the color feature. By processing the current camera image on the screen, the most dominant colors of the capture get extracted and its the users choice which of the colors are added to the structure.

In addition to the age, a sun or moon are placed in the Cuby's orbit depending on the current time when adding the feature. The sun and moon optionally leave a golden glow or blue shine on the cube's surface.

Next to the structure, a card styled like a museum label gives a quick overview about all enabled parameters.

[1] https://github.com/Unity-Technologies/arfoundation-samples

[2] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.0/manual/index.html

[3] https://unity.com/shader-graph

[4] https://unity.com/visual-effect-graph

[5] https://www.mapbox.com

2.1 Features

As already outlined briefly, this section describes the implementation, issues and solutions of these main features.

2.1.1 Location

The user's location changes the material of the cube. With the Mapbox API [6] we get a heightmap and satellite image from the user's current location. The Mapbox API was the only API that provided detailed and consistent elevation data all over the world. This was especially useful since one project partner worked on the project from Finland, where our first tested map API did not have any data for. Using Mapbox for our application was a bit of an overkill, because Mapbox is a huge API that offers extensive map data and we only needed one to two simple maps. Strangely enough, the Unity version of this large tool was highly focused on spherical map representations which only used the satellite image for albedo maps. It was hard to extract a simple feature, namely getting the heightmap with a single web query, but this worked very well in the end. However this also was where the first troubles came up with using Unity's URP. After extensive research we found out that even though the other new render pipeline and the old one both implemented a height map functionality that distorts the geometry, URP did not. This was resolved by using a normal map but was noted that it might be exchanged in the future by our own heightmap implementation.

2.1.2 Sound

Sound does not equal sound. There is a lot you have to consider when you want to capture the volume of a microphone. Should Cuby react to the absolute amplitude or relative to the general noise level of the environment? This means that either the full range of loudness or temporal loudness changes are represented. Or does it need a calibration with user input at the beginning for more accuracy? Depending on the desired recording content, the maximum amplitude could be reduced to provide a wider range for the detected signal and therefore finer sampling. Also the length of the recording plays a big role, since the user can either save small parts or have the surrounding's noise mapped to the structure continuously. We decided to save a recorded clip in our data that can be replayed and not only react to the current sound level. But then even more questions came up since the algorithm worked with the raw microphone data. We had to consider the sampling rate and the channels of the recording microphone as well. This influences the size of the array to which the recorded data will be saved. Additionally, defining parameters like the averaging rate for creating a smooth waveform had to be considered. If the step is rather small, the discrete samples will show rapid changes, however, if it is too big, single spikes in the wave would get lost in the smoothed data. The generated sound wave influences the particle system of Cuby. So, the recorded data also must be playable with the recorded audio clip in sync. This was achieved by implementing a class that broadcasts the volume changes at the same rate they have been recorded before. The recorded audio clip and the smoothed recording data are then al-

[6] https://www.mapbox.com/

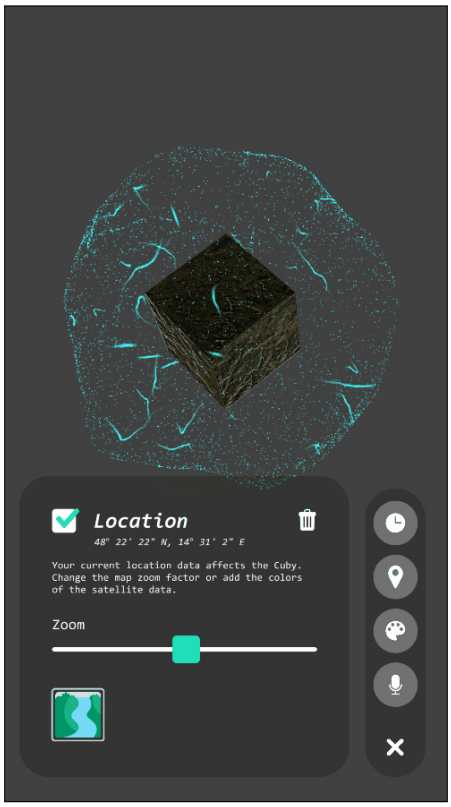
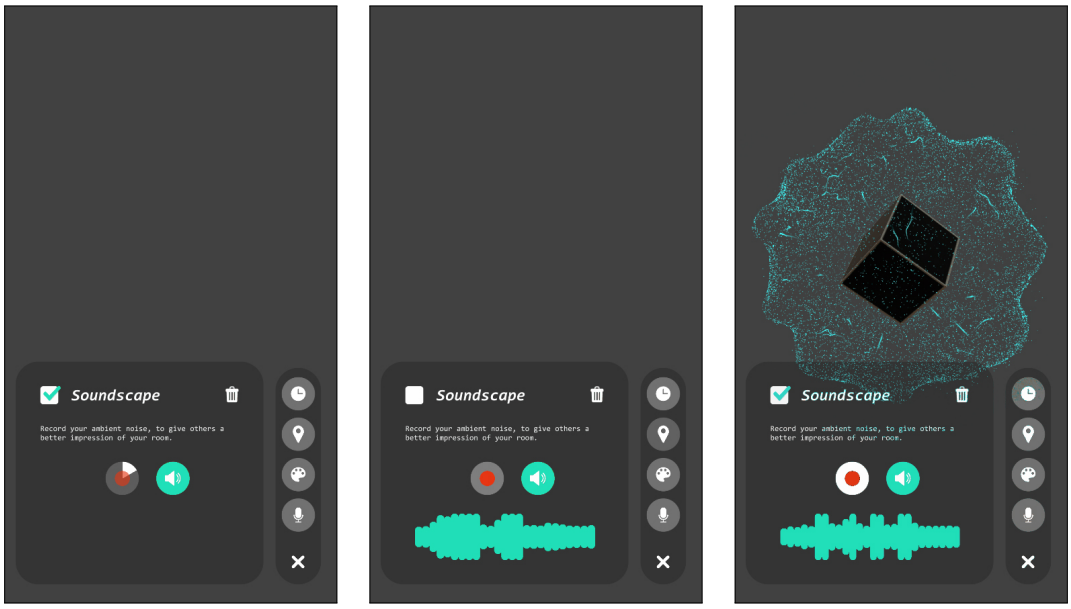


Figure 2.1: This image shows the normal map obtained from the Mapbox API applied to the cube, as well as the user interface to control the map's zoom factor.

ways replayed simultaneously, however, the actual recording can be muted. To represent the sound wave in the menu and on the info card, a texture was created that adjusts to the recording length and maximum amplitude of the data. Although, the sound wave originally was well represented by the Visual Effects Graph, this did not end up in the application. The VFX Graph allowed us to spawn a huge number of particles even on mediocre processing hardware, and it also provides functionality to change these particles' parameters easily after they have been spawned. When everything in the project was put together later on, however, we realized that the combination of ARFoundation, URP and the VFX graph was not yet optimized for smartphone usage and produced bad artifacts and shears of particles across the screen. Temporarily, this representation is replaced with Unity's old particle system, but it is planned to upgrade to the working VFX graph version as soon as its ready from Unity.

2.1.3 Colors

Since the colors were an incentive for this project we wanted to extract them as accurately as possible. AR Foundation offers a way to get the latest camera image on the CPU which was the initial step towards this goal. We then save this raw data to a texture and process it with our custom computer vision algorithm. However, this was harder than expected because color is not simply color. One algorithm that we found



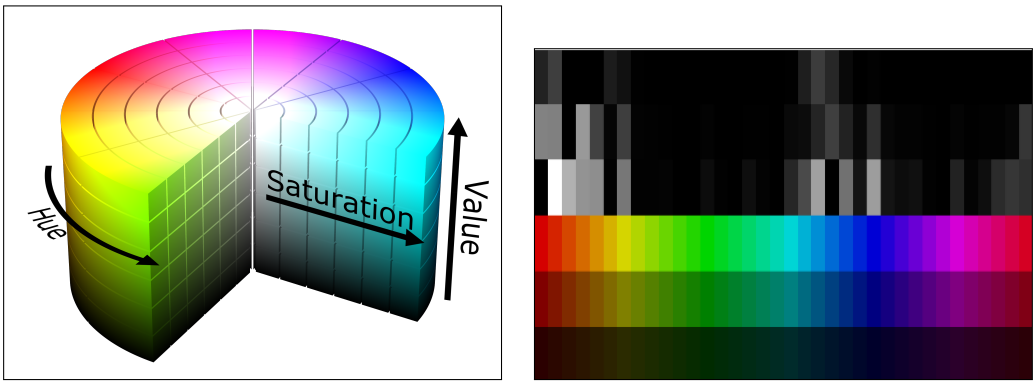
(a) (b) (c)

Figure 2.2: Recording the soundscape (a), the resulting waveform (b) and a different recording which is also applied to the structure (c).

first looked promising on test images with enough colors and contrast. But it did not work at all with dull indoor captures. It was based on plotting all the pixel's accumulated color information in three-dimensional space and then clustered them either by fixed grid steps or by using k-means clustering. Since our mobile phones' cameras did not produce as bright and colorful images, we had to find a different way of creating our color histograms. By combining various research algorithms which all were not quite right for our use case, we then came up with an implementation that creates a histogram from HSV instead of RGB colors. This means that the algorithm fills bins in the cylindrical HSV color space representation with the pixels' values and counts the most populated bins. This way the hues are retained while they were averaged in the previously described implementation. The results are far more accurate and we apply the 7 most dominant colors of the resulting histogram to our structure's particles. Again, the same problem as in 2.1.2 occurred, so we later had to exchange our VFX graph's particle implentation in Unity's old particle system.

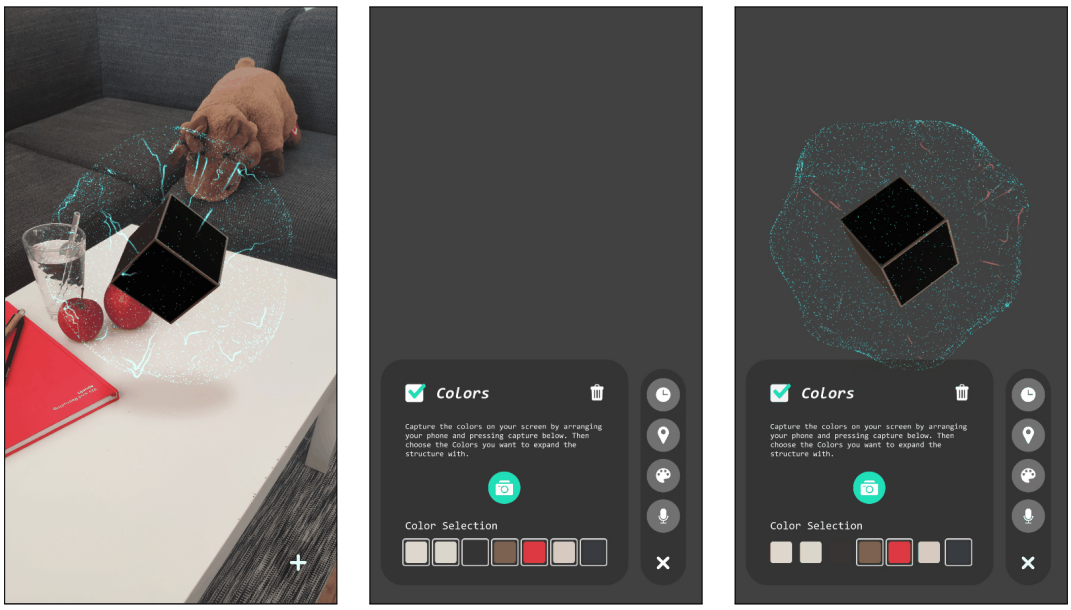
2.1.4 Timestamp

The timestamp feature adds a sun or a moon. It is placed in the Cuby's orbit depending on the current time when adding the feature. The sun and moon optionally leave a golden glow or blue shine on the cube's surface. Deciding whether a time of day is during daylight or in the night time was simply done by a threshold for morning and evening. It was implemented so that twelve hours of sunlight map to 180 degrees in a hemisphere around Cuby. The moon is treated equally, which means that the sun will be in the zenith at noon while the moon will reach its highest point at midnight. To achieve



(a) (b)

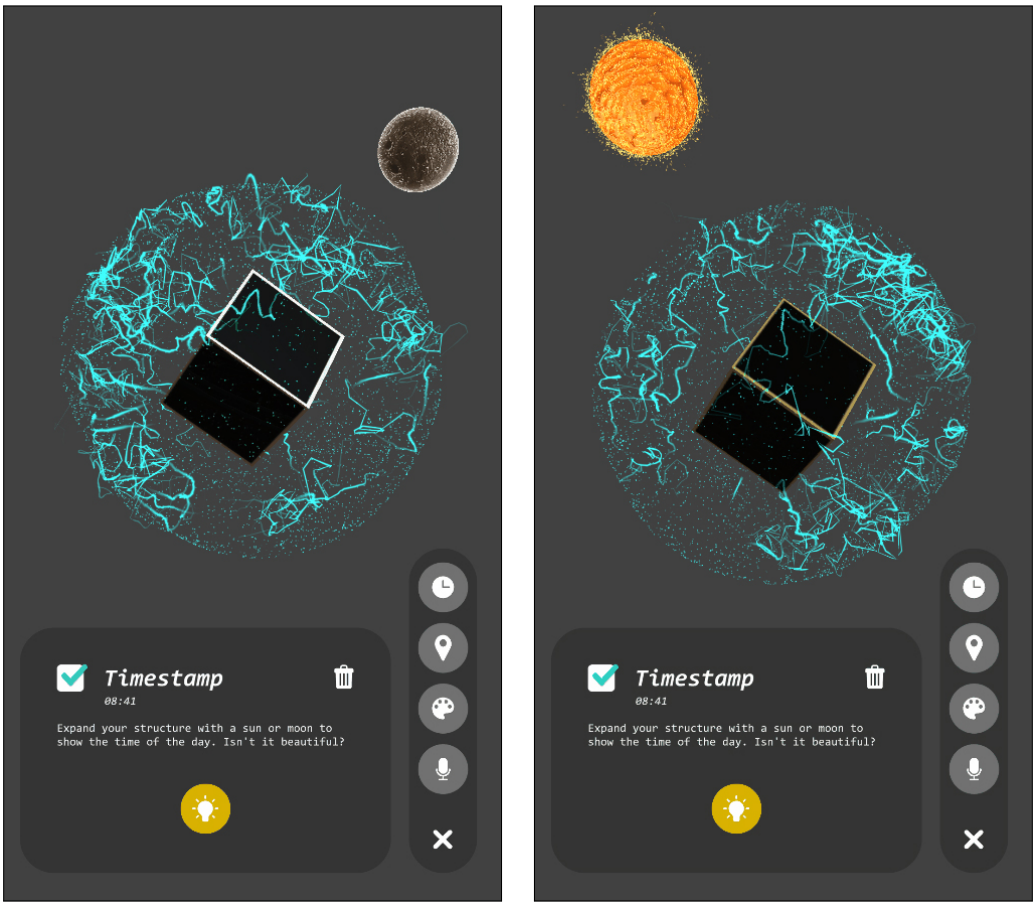
Figure 2.3: The hue-value-saturation space (a) was used to create histograms (b) of the camera's image and extract the dominant colors.



(a) (b) (c)

Figure 2.4: An example image (a) with its dominant colors extracted in (b) and a selection that starts to appear on the structure (c).

the rotation in the orbit as well as the correct orientation of their respective directional lights, the two objects are translated relatively to their empty parents transform. In Unity, this creates a way to treat their parent as the anchor around which the whole object and light can be rotated.



(a) (b)

Figure 2.5: The moon (a) and the sun (b) are instantiated in Cuby's orbit and provide information about the time of day when the structure was created. A bright yellow and white light are optional.

2.1.5 Age

Cuby ages. By opening the application for the first time, Cuby's life starts and its birthday is saved. This value, however, is fixed and cannot be changed by the user. With the days passing by, the spawn rate of the particles increases.

simply get the timestamp of the phone - max age 100 days

2.2 Info card

The info card is designed to resemble a museum label which shows information about the structure. The description changes dynamically when adding features. Thus, at the beginning it is a blank square that can be filled with feature descriptions. When enabling a feature in the menu, the card is automatically updated and shows the according information.

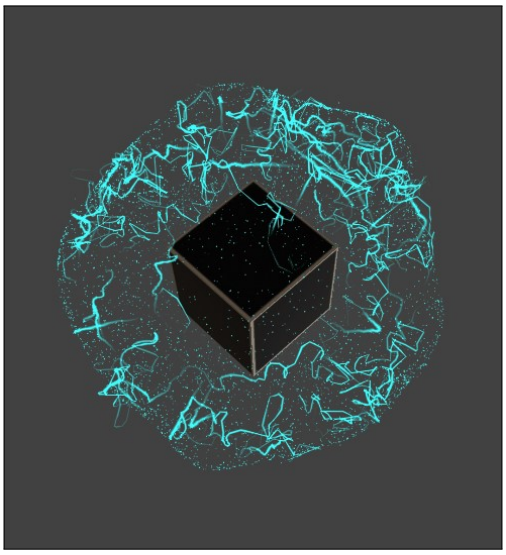


Figure 2.6: An “old” Cuby (with a maximum of 100 days) as it is shown in this image will exhibit a lot more noise and particles then a freshly spawned one.

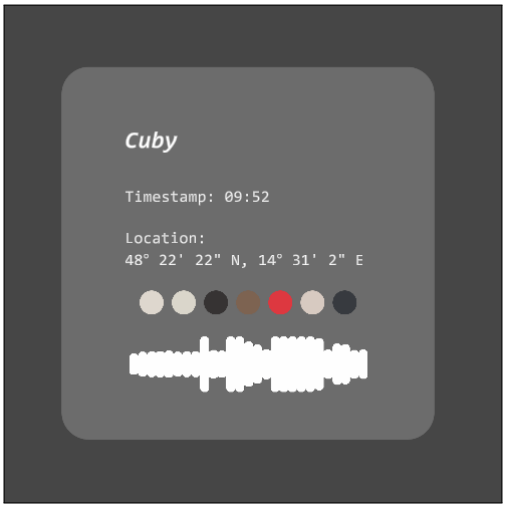


Figure 2.7: The info card summarizes all the information yet applied on the structure.

2.3 Serialization

Since Cuby was designed to be a representation of a surrounding which should be comparable with other Cuby structures, sharing the information as a bundle was desired as well. Therefore, serialization and deserialization of all the data and parameters was necessary. However, in many cases only the absolutely needed information was stored. For instance, instead of including the location’s texture maps in the file and creating some kind of archive, only the coordinates were saved and the textures retrieved by performing another map query on loading the data.

Microsoft strongly advises against using binary serialization due to security reasons in [3]. Therefore, we are currently using Unity's built-in JSON Serialization to convert the feature data to a convenient format. Research showed that there is no uniform way to access a file saving dialog on mobile devices within Unity. So, a freely distributed file handling plugin[7] was used and worked immediately out of the box.

2.4 Sharing

The Cuby's data is saved as a JSON file which can simply be sent to friends like any other file. For this, another great tool from a GitHub repository[8], that enables native file share on both mobile platforms, was used.

2.5 Problems

Time issues were generated through the lack of AR Foundation not providing any remote testing software. Therefore, all AR-related development had to be built from Unity and sent to the phones for testing every time we changed any AR-related feature. We tried to finish as many tasks in the Unity Editor as we could, like building a well guided user interface. But sticking to the PC led to a bigger problem, that we encountered very late in the process. Even worse is the fact that Unity's Shader Graph and Visual Effects Graph from the new Render Pipeline are not compatible with mobile devices yet. The only possible solutions for the time being are to either wait for Unity to release a compatible version or, in the meantime use a fallback version and implement a similar solution with Unity's old capacities. But why use an outdated system when Unity itself is promoting the new Shader/VFX Graph and the project can or even should be updated when it is production-ready? Nevertheless, we wanted a working solution on the smartphone in reasonable time so we had to fall back on the old versions.

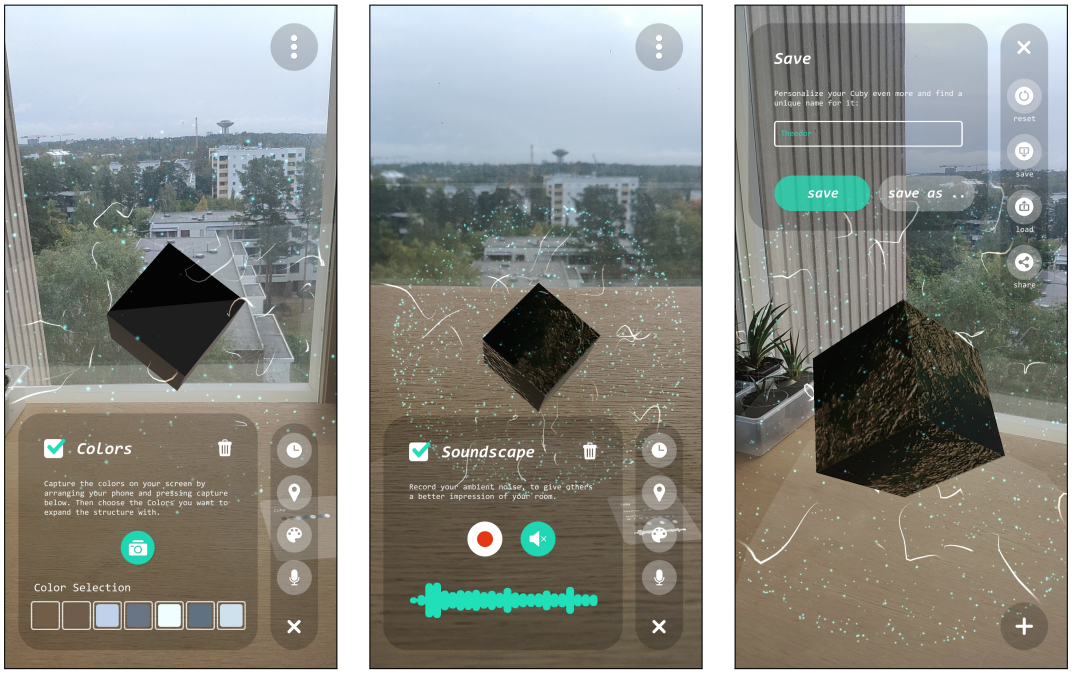
2.6 Android Version

Since we already encountered many time-consuming challenges, coming up with new algorithms and adjusting to Unity's possibilities, one problem remains unsolved: the application is only available for Android for now. Due to us having problems with Apple's development integration and not having a macOS based PC around, we were not able to build and test Cuby on any iOS powered phone yet. However, everything is implemented for both operating systems and built for the common ground, so it should not be too feasible to get Cuby to run on iPhones as well.

The Android version on the other hand is well tested and has been used for all mobile development in this project. Due to the aforementioned issues, the results do not look like they were intended to look like and are replaced by slightly inferior visuals. Nevertheless, it is great to see the application in action through a phone's camera.

[7] https://github.com/yasirkula/UnitySimpleFileBrowser
[8] https://github.com/yasirkula/UnityNativeShare

Figure 2.8: A logo was designed for the app which resembles a simple cube and picks up on the color scheme of the application's UI.



(a) (b) (c)

Figure 2.9: Screenshots of the final Android application showing the Colors feature (a), Soundscape (b) and the saving option's menu (c).

Chapter 3

System Architecture

Apart from the previously mentioned packages, APIs, software and other gimmicks that have been used in making this project possible, there have been a few considerations that shaped the architecture of the application.

3.1 Statecharts

One of the major decisions fell on using statecharts to represent the application's flow. As described in [9] they are an improved version of state machines that allow all project partners to have a common view on what is going on inside of the software. Not only can they be represented visually as a kind of flow diagram, but they also immediately translate to code. Additionally, they provide a highly readable syntax to formulate what should happen when during their execution. Again, using this kind of state handling might be an overkill, but it immediately demonstrated its big advantage when outlining the overall flow (initialization, idle state, feature menu and option menu) of Cuby. Unfortunately, statecharts are only very advanced in the field of javascript and not readily available in .Net development. But thanks to this GitHub repository [5] we were able to use its basics in Unity. This facilitated state handling and made the whole process less error-prone.

The code that is responsible for Cuby's application flow subsequently ended up being rather short:

```
1  public override StatechartDefinition<NoContext> Behaviour => Define.Statechart
2      .WithInitialContext(new NoContext())
3      .WithRootState(
4          "Flow"
5              .AsCompound()
6              .WithInitialState("StartMenu")
7              .WithStates(
8                  "StartMenu"
9                      .WithEntryActions(
10                                 Run(() => startMenu.gameObject.SetActive(true)),
11                                 Run(() => RequestPermissions()))
12                     .WithExitActions(
13                                 Run(() => startMenu.gameObject.SetActive(false)),
14                                 Run(() => HUD.instance.HideMessage()),
15                                 Run(() => Cuby.instance.Resize()))
```

```
16                      .WithTransitions(
17                          On(PlaneTapped).TransitionTo.Sibling("Idle")
18                              .WithActions<NoContext>(PlaceCuby)),
19                  "Idle"
20                      .WithTransitions(
21                          On(ToggleExtensionMenu).TransitionTo.Sibling(
                                "ExtensionMenu"),
22                          On(ToggleOptionMenu).TransitionTo.Sibling("OptionMenu"),
23                          On(PlaneTapped).TransitionTo.Self
24                              .WithActions<NoContext>(PlaceCuby)),
25                  "ExtensionMenu"
26                      .WithEntryActions(
27                          Run(() => extensionMenu.ShowNavigation(true)))
28                      .WithExitActions(
29                          Run(() => extensionMenu.ShowNavigation(false)))
30                      .WithTransitions(
31                          On(ToggleExtensionMenu).TransitionTo.Sibling("Idle")),
32                  "OptionMenu"
33                      .WithEntryActions(
34                          Run(() => optionMenu.ShowNavigation(true)))
35                      .WithExitActions(
36                          Run(() => optionMenu.ShowNavigation(false)))
37                      .WithTransitions(
38                          On(ToggleOptionMenu).TransitionTo.Sibling("Idle"))));
```

This statechart definition just has to be paired with a few event definitions that can be invoked at any other point:

```
public void ExtensionMenuClicked() => SendToStatechart(ToggleExtensionMenu);
private static NamedEvent ToggleExtensionMenu = Define.Event("ToggleExtensionMenu");

public void OptionMenuClicked() => SendToStatechart(ToggleOptionMenu);
private static NamedEvent ToggleOptionMenu = Define.Event("ToggleOptionMenu");

private static NamedDataEventFactory<Pose> PlaneTapped =>
                                        Define.EventWithData<Pose>("PlaneTapped");
```

3.2 Features

The features include the Colors, Location, Timestamp, Soundscape and Age which are all based on the same class. Whenever any feature changes its parameters, asynchronously acquires new data or is removed, events are invoked. This is the basis for all the communication within the application. The different menus, the visualization as well as the info card subscribe to the static events `Feature.onChanged` and `Feature.onRemoved`. The respective feature is passed as the event data from which the subscribers retrieve their new values. This way, every necessary recipient is notified about changes and can decide on their own what to do with the information and how to handle the new data. Additionally, features are described by the two flags `exists` and `enabled` which can be used to exactly determine the status of a feature. The soundscape for example can be enabled (i.e. activated) without having associated data, which would return false for `enabled`.

3.3 Menus

All the feature's menus are derived from the `FeatureMenu` base class which itself is of type `Menu`. The Menu is a simple class that handles fading and activating `CanvasGroup` components. FeatureMenu is interesting because it is itself implemented as a template class which means that `FeatureMenu<T> where T : Feature` keeps a reference to exactly their respective feature with the correct type and only reacts to changes in said feature. The navigation is aligned in automatically adjusting layout groups which facilitate the addition of feature menus and option menus. The navigation button's events are simply handled in Unity's inspector which open the corresponding menu.

3.4 AR Foundation

The necessary elements of ARFoundation to run a scene on a smartphone are the `AR Session`, the `AR Session Origin` and the `AR Camera`. In our setup, these handle all the AR-related tasks such as positioning and plane recognition, as well as the visualization of augmented feature points and planes. Planes are set up to only be detected horizontally.

3.5 Other Classes

A few more helper classes have been implemented to aid in various tasks. Some of them are briefly explained in the following section.

Cuby follows a singleton pattern which provides access to its data at any time.

CubyData is the backbone to the Cuby class where every necessary value is stored and can be retrieved, grouped in nested classes which match with the features. In addition, this class provides a few helping hands for serialization. Some Unity-intern classes like `AudioClip` and `Color` can not be serialized to JSON by default and have been equipped with fitting serialization helpers.

SaveSystem handles the saving and loading dialogs as well as accessing the JSON serialization. This class also invokes and event when loading a Cuby's data is finished which causes the Cuby itself and all the features to update their values to the newly loaded ones.

Permissions take care of mobile phone permissions. If the access to the camera, microphone, file system, location, etc. is not granted, some features will not work. Unity's own Android permission solution is quite error-prone which is why a separate plugin from GitHub[1] has been used.

HUD a head-up display, which is figuratively used in this case for a screen space info-center that consists of a single lined text output. This singleton allows every other class to show the user a message through its static methods.

[1] https://github.com/yasirkula/UnityAndroidRuntimePermissions

Chapter 4

Summary

Experimenting with the device data and the parameters entailed new knowledge. Not only in terms of augmented reality and mobile app development with Unity, but also in each aspect looked into, like audio and image processing. AR foundation is a great base technology to use but led to many issues which had to be resolved and need further attendance in the future. Some feature's visualization can be improved as soon as the Universal Render Pipeline supports the VFX Graph on mobile devices.

It has been challenging to figure out how all components of a room can be captured in one structure. This is definitely the weakest point in the application, which can be extended by lots of features or differently used features. Various methods of representing the surroundings have to be tested by users and can therefore produce a more meaningful and better blended structure. Nevertheless, the well-constructed architecture which has been established is a good foundation for further extension towards the projects initial goal.

References

[1] *ARBrush*. URL: https://github.com/laanlabs/ARBrush (cit. on p. 4).

[2] *4th wall ar app*. 2018. URL: https://nancybakercahill.com/4th-wall-ar-app (visited on 04/02/2020) (cit. on p. 5).

[3] *BinaryFormatter security guide*. URL: https://docs.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide (visited on 04/02/2020) (cit. on p. 13).

[4] *Innerspace*. URL: https://www.innerspace.eu/ (visited on 04/02/2020) (cit. on p. 5).

[5] Bernhard Mayr. URL: https://github.com/bemayr/Statecharts.NET (visited on 06/11/2020) (cit. on p. 15).

[6] *Moto Wall*. 2013. URL: https://www.heavy.io/motowall (visited on 03/25/2020) (cit. on pp. 4, 5).

[7] *Organismus*. 2019. URL: https://www.hbksaar.de/projekte/details/organismus (visited on 03/28/2020) (cit. on p. 5).

[8] *Revolver-Rotationen 2019*. 2018. URL: https://www.hbksaar.de/projekte/details/revolver (visited on 04/01/2020) (cit. on p. 5).

[9] *Welcome to the world of Statecharts*. URL: https://statecharts.github.io (visited on 06/11/2020) (cit. on p. 15).