MSc - Önálló laboratórium 1

Augmenteted reality using ARKit

Daniel Mark Kiss

2023

Contents

Chapter 1

Introduction

Smartphones have become a defining part of our days. By now we can't even imagine our everyday life without them. We read e-mails, browse the Internet or monitor daily exchange rates. Now we can do all important things with our phones. Listening to music and watching series, but in terms of photography and video capabilities, they are not far behind cameras designed for professional purposes. Their computing capacity has grown exponentially over the years. The quality of camera systems is also undergoing significant progress every year.Perhaps this is why many people say that the development of telephones is starting to slow down. This industry has reached its peak.

At the time, in 2007, the iPhone was considered a huge world sensation and now we know it was a world-changing announcement. At the time of writing this report, there are several reports that Apple is working on the introduction of a similarly large and decisive new product line. The so-called Apple AR/VR glasses. Only time will tell if this really happened until then I thought that it would be a great advance if I started familiarising with this technology. If this will be the next ”touchscreen phone era”, I thought it would be worthwhile to start dealing with it in time. Regardless, I have long been interested in the technology's operating principle and scope of use. From the consumer side, what kind of application possibilities does augmented reality technology have? At that time, I wanted to implement a similar augmented reality-like navigation system in my own high school navigation application. Unfortunately, I did not know about the existence of these technologies at the time.

As I mentioned in the first paragraph, today's phones have a huge computing capacity. This is also why it is possible for such augmented reality applications to run on our phones. Based on the 1 or better case 2 or 3 lenses placed in the phone, it can measure the 3D depth of each scenario. There are phones, such as the iPhone 13 Pro, which have a laser LiDAR sensor placed specifically for this purpose. In the 2010s, many devices with similar technology came out, such as Microsoft's Kinect developed for the X-Box 360 console. It is also worth mentioning Google Glass developed by Google or HoloLens and HoloLens 2 marketed by Microsoft. Unfortunately, the former mentioned Google Glass was not an undivided success and in 2023 the sale of the glasses was discontinued.

Last but not least, it is important to mention the Oculus Quest 2 VR headset introduced by Meta and the computer use and games based on it. In addition, the Metaverse, announced by Meta 2 years ago, is also an important milestone in the life of virtual reality and therefore also in the life of augmented reality.

As the examples above clearly reflect, it will be an important and presumably defining technology in the future. In light of this, during my self-lab, I got to know the ARKit and RealityKit developed and used by Apple. And with the help of the frameworks, I developed an augmented reality application displaying economic metrics. And during the report I used LaTex, because I felt it was time to familiarize myself with this kind of documentation language.

Chapter 2

Current techologies

Before I started the development of the application I have done a research period to have a better overview of the current market. There are several frameworks to choose from to start developing augmented reality applications. The most used AR frameworks are Google ARCore, Apple ARKit and RealityKit, Simple CV and Unity AR Foundation. I ended up using Apple ARKit to build my app. I've been working with iOS development for several years and I'm sure with the Swift language as well.

2.1 Unity

One of the main frameworks that we can use if we want to develope augmented reality application is Unity. Unity is a purpose-build framework for augmented reality development. It's key feature is that with the framework developer can easily deploy it's product across multiple mobile and wearable AR devices. It includes core features from ARKit(Apple), ARCore(Google) and HoloLens(Microsoft) as well unique features.

AR Foundation which is the recommended framework for Unity development there are several features what the developers can use like:

- Device tracking
- Raycast
- Plane detection
- Gestures
- Face tracking

2.2 Apple ARKit and RealityKit

Apple ARKit and RealityKit are two powerful technologies developed by Apple that enable the creation of immersive augmented reality experiences on iOS devices. ARKit is a software framework that provides developers with the tools and resources they need to build augmented reality apps for iPhones and iPads. It allows developers to easily place virtual objects in the real world, track the user's position and orientation, and integrate real-time camera data to create realistic and interactive experiences.

On the other hand, RealityKit is a high-level framework that makes it easy for developers to create 3D content and build AR experiences without requiring extensive knowledge of 3D graphics or game engine programming. RealityKit includes features such as physics simulations, animations, and spatial audio, which can be used to create highly immersive AR experiences.

Together, ARKit and RealityKit provide developers with a comprehensive set of tools for building augmented reality experiences on iOS devices. With these technologies, developers can create innovative and engaging apps that allow users to interact with virtual objects in the real world, opening up new possibilities for gaming, education, and more.

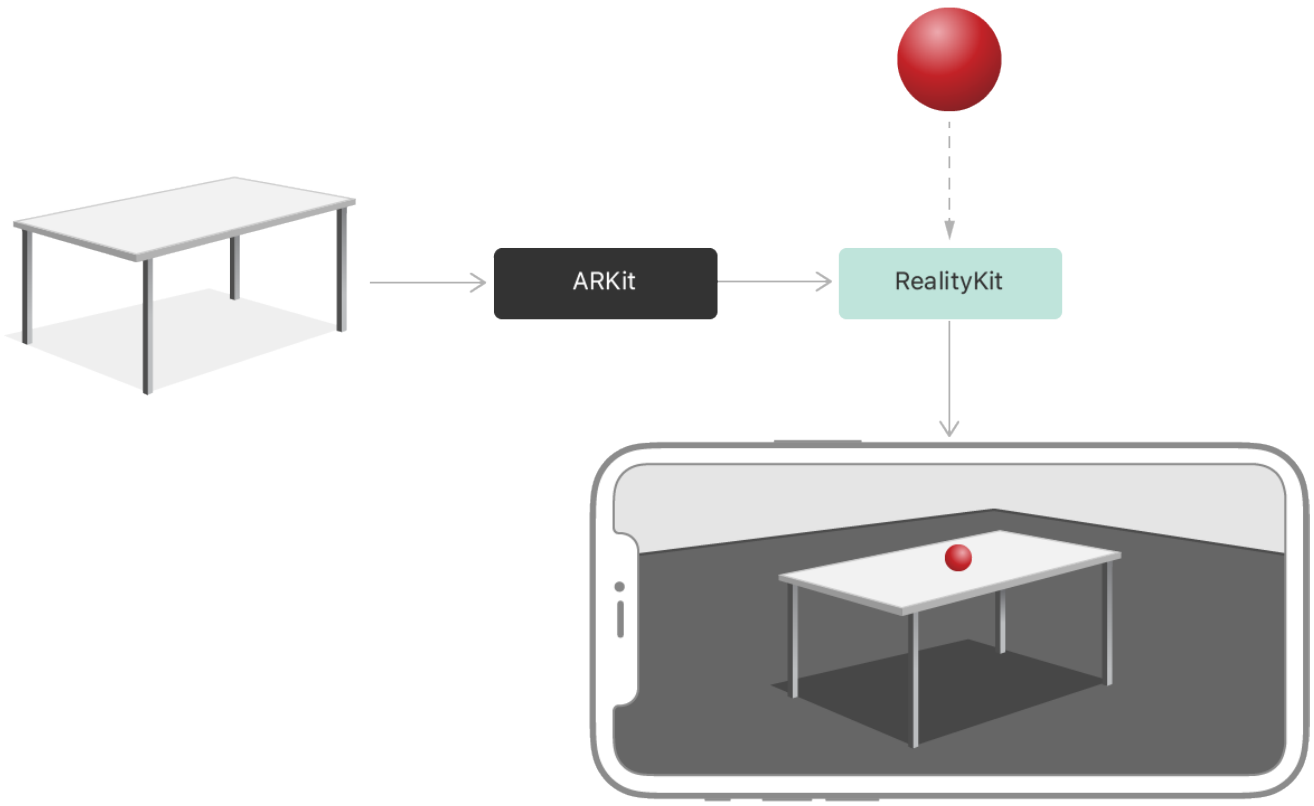

Figure 2.1: RealityKit and ARKit usage.

RealityKit's features:

- Rendering: RealityKit offers a powerful new physically-based renderer built on top of Metal, which is fully optimized for all Apple devices.
- Animation: It has built-in support for Skeletal animation and Transform-based animation. So, if you want, you can animate anithing or you can move, scale and rotate objects with various easing functions.

- Physics: With a powerful physics engine, RealityKit lets you adjust real-world physics properties like mass, drag and restitution, allowing you to fine-tune collisions.
- Audio: Spacial audio understanding and automatic listener configuration let you attach sound effects to 3D objects. You can then track those sounds, making them sound realistic based on their position in the real world.
- ECS: From a coding perspective, RealityKit enforces the Entity Component System design pattern to build objects within the world.
- Synchronization: The framework has built-in support for networking, designed for collaborative experiences. It even offers automatic synchronization of entities between multiple clients.

Chapter 3

Development

3.1 Overview

For the development phase of the project I used Apple's XCode integrated developer environment(IDE) as the main developer platform for iOS and ARKit development. For testing I used an iPhone 11 with dual camera system. I have also version controled the whole development process using git and publishing it on GitHub. Not only the source code can be found there but also the documentation of this project as I have writen it using LaTeX.

3.2 APILayer Rest API

The fundamental part of the application is the data it displays. For retriving the displayed informations I used APILayer's Exchange Rates Data API an open and available for free financial API. The only problem is that you can only do 250 queries per month in the free version. I used 2 endpoints. The first is '/convert'. With this endpoint, we have any amount conversion from one currency to another. The output of this enpoint is the following JSON.

```
{
    "success": true,
    "query": {
        "from": "EUR",
        "to": "HUF",
        "amount": 1
    },
    "info": {
        "timestamp": 1682930463,
        "rate": 373.180303
    },
    "date": "2023-05-01",
    "result": 373.180303
}
```

The other endpoint used is '/fluctuation'. This endpoint returns the fluctuation data between specified dates. The data can be for all available currencies or for a specific set.

```
{
  "base": "EUR",
  "end_date": "2018-02-26",
  "fluctuation": true,
  "rates": {
    "JPY": {
      "change": 0.0635,
      "change_pct": 0.0483,
      "end_rate": 131.651142,
      "start_rate": 131.587611
    },
    "USD": {
      "change": 0.0038,
      "change_pct": 0.3078,
      "end_rate": 1.232735,
      "start_rate": 1.228952
    }
  },
  "start_date": "2018-02-25",
  "success": true
}
```

3.3 SwiftUI

SwiftUI is Apple's brand new framework for building user interfaces for iOS, tvOS, macOS, and watchOS. Apple introduced SwiftUI in 2019 and the framework has been evolving ever since. Unlike UIKit, SwiftUI is a cross-platform framework. The key difference with UIKit and AppKit is that SwiftUI defines the user interface declaratively, not imperatively. What does that mean?

Using UIKit you create views to build the view hierarchy of your application's user interface. That is not how SwiftUI works. SwiftUI provides developers with an API to declare or describe what the user interface should look like. SwiftUI inspects the declaration or description of the user interface and converts it to your application's user interface. SwiftUI does the heavy lifting for you.

One of the most challenging aspects of user interface development is synchronizing the application's state and its user interface. Every time the application's state changes, the user interface needs to update to reflect the change. During the development phase, this was a challenge that had to be overcome. Despite the fact that I have already used and developed an iOS application with SwiftUI, it was excellent practice to deepen my knowledge of user state management. I used ObservableObjects to solve this problem.

I used a common state management technique, the MVC pattern, to control the data and model. MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display. This "separation of concerns" provides for a better division of labor and improved maintenance. Sticking to convention, I created a CurrencyController, CurrencyView and a CurrencyModell class. The CurrencyModel class contains the generated 3D models and their associated values. The task of the CurrencyController class is to query the data and update the information displayed on the View. In the CurrencyView class, it deals with the code defining the appearance of the application and the display of the given dataset.

3.4 ARKit and RealityKit

To operate augmented reality and display the 3D generated graph, I used the ARKit and RealityKit frameworks provided by Apple.

The CurrencyARViewContainer is responsible for displaying the AR view.

```
struct CurrencyARViewContainer: UIViewRepresentable {

    @StateObject var controler:CurrencyController

    func makeUIView(context: Context) -> ARView{
        AR.view = ARView(frame: .zero)
        return AR.view
    }

    func updateUIView(_ uiView: ARView, context: Context) {
        print("updating view - \(controler.timerHappened)")
        uiView.scene.anchors.removeAll()
        ...
    }
}
```

To generate the texts and columns, I used the .generateBox() and .generateText() functions of the built-in MeshResource class. The MeshResource class stores the points defining the shapes. In order for this to become a 3D model, a texture must also be specified. I used the SimpleMaterial() function for this. We also need an

AnchorEntity, which defines the center of our model in the 3D world. After defining these variables, we can create the ModelEntity and place it in the AR world using the AnchorEntity.

```
func updateUIView(_ uiView: ARView, context: Context) {
    uiView.scene.anchors.removeAll()

    let cylinderMeshResource = MeshResource.generateBox(size: SIMD3(x: 1.2, y: 0.01, z:

    let myMaterial = SimpleMaterial(color: .gray, roughness: 0, isMetallic: true)
    let radians = 90.0 * Float.pi / 180.0

    let kozeppont = AnchorEntity(world: SIMD3(x: 0.0, y: 0.0, z: 0.0))
    let axisXEntity = ModelEntity(mesh: cylinderMeshResource, materials: [myMaterial])

    let coneXEntity = ModelEntity(mesh: coneMeshResource, materials: [myMaterial])
    coneXEntity.orientation = simd_quatf(angle: radians, axis: SIMD3(x: 0, y: 0, z: -1))

    axisXEntity.addChild(coneXEntity)
    coneXEntity.setPosition(SIMD3(x: 0.6, y: 0.0, z: 0.0), relativeTo: axisXEntity)



    kozeppont.addChild(axisXEntity)
    uiView.scene.addAnchor(kozeppont)
    ...
}
```

To be able to move the different elements together, all 3D models are children of the axes. Thus, if the axis moves, the connected elements will also move due to the parent-child relationship. In its current version, MeshResource does not support the generation of cones by default, so I was able to achieve this by using an external library package. After importing the RealityGeometries library, I was able to easily generate cones, which I eventually used to draw axes.

Chapter 4

Presentation of finished work

In this section I will present and showcase my finished appliction. I will include screenshots to have a better representation and understanding for the reader. The camera function in the application allows you to capture the current state of the graphs, which in this case came in handy for documentation.

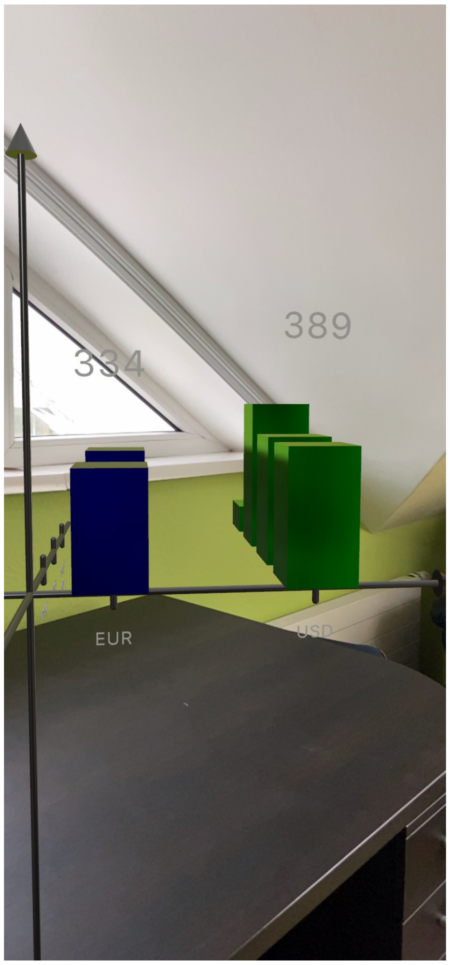

Figure 4.1: The main screen of the application.

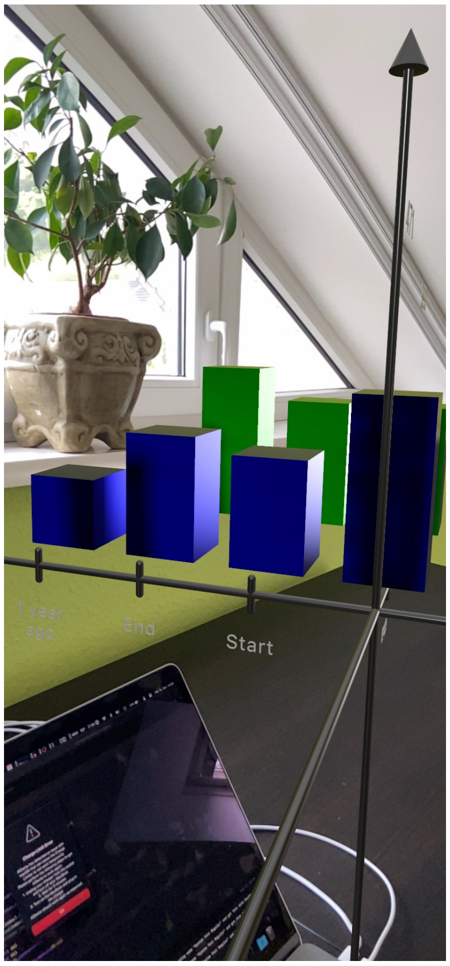

Figure 4.2: The main screen of the application.

As the attached images clearly show, the virtual 3D graph can be easily walked around and viewed from different angles, thereby giving users a new comparative perspective. It is possible to move and rotate the entire graph, as well as move the current exchange rate columns to make it easier to compare with other metrics. The move and rotate functions are only available in Spectate mode, for this you have to stop Live mode (or otherwise start Live mode) by pressing the button located in the upper left corner.

Currently, 2 currencies are available in the application, but of course this can be easily expanded at any time in the future. These two are EUR to HUF and USD to HUF. These can be displayed after selection and confirmation from the bottom bar. If the given exchange rate is already placed, it can no longer be added to the graph twice.

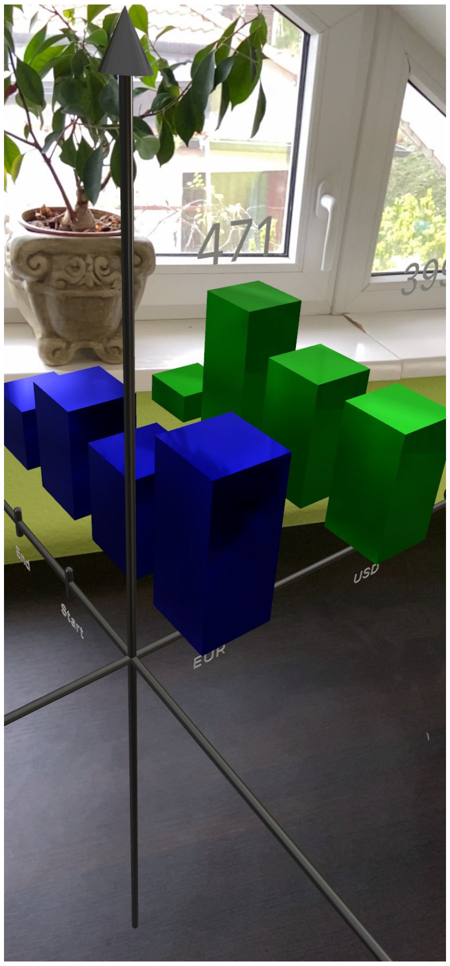

Figure 4.3: The main screen of the application.

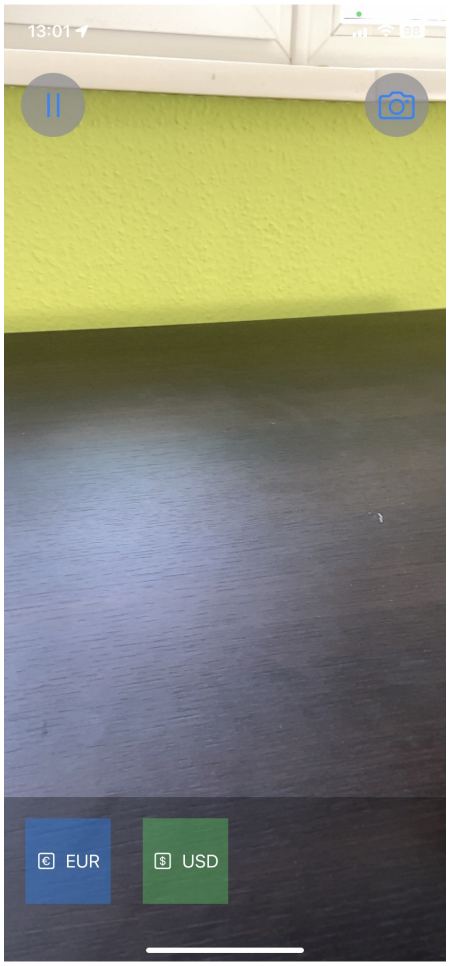

Figure 4.4: The main screen of the application.

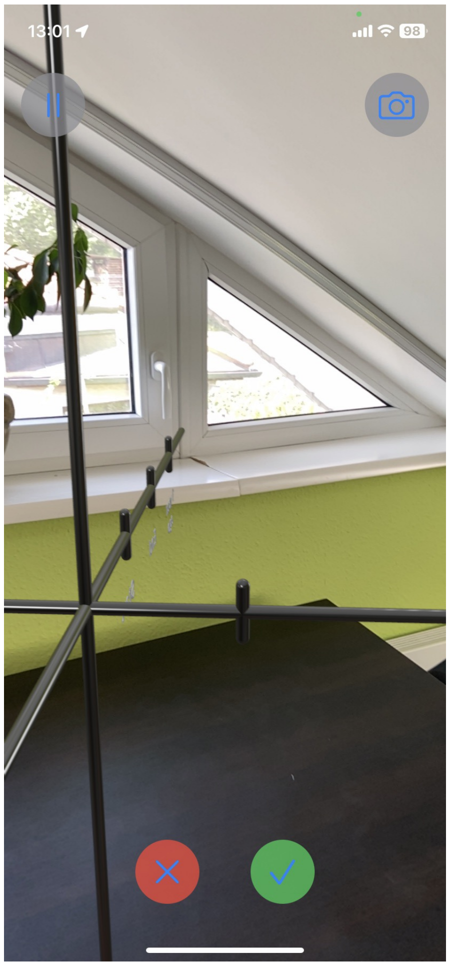

Figure 4.5: The main screen of the application.

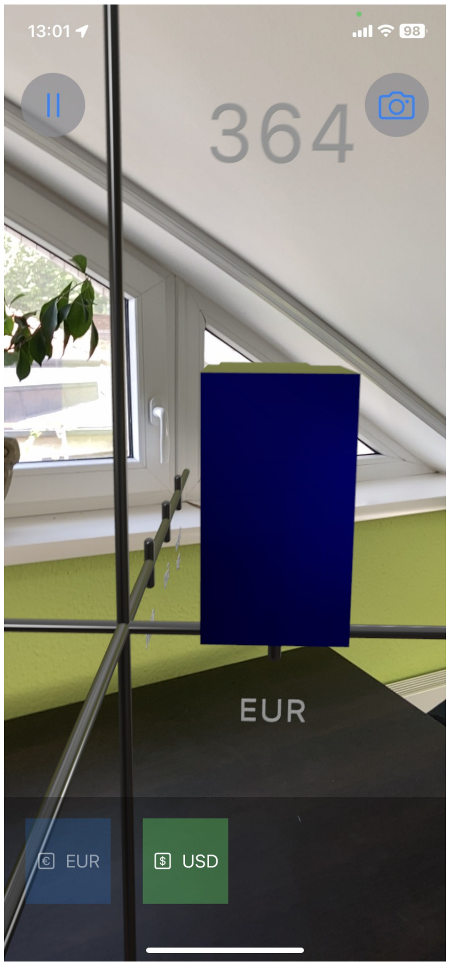

Figure 4.6: The main screen of the application.

Chapter 5

Sources

Below I list the external links and sources used during the project and the report.

5.1 External Links

Microsoft - What is Augmenteted Reality

Apple - Augmented Reality

Apple - RealityKit

Kodeco - RalityKit tutorials

Apple Forum - Where to start ARKit

Apple - Adding procedural assets to a scene

Medium - Adding 3D text to scene

BetterProgramming - Taking AR view snapshot

YouTube - Placing models

GitHub - FocusEntity

BetterProgramming - Update model entity

RapidAPI

Yahoo Financial API Guide

APILayer REST API

RealityGeometries - Kúpinsta

YouTube - RealitySchool: Place, Interact with, and Remove AR Objects in RealityKit

Cocoacast - SwiftUI

Hacking with Swift - ObservedObject

Mozilla - MVC