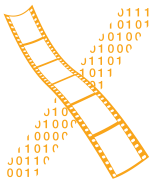


Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Pen-based Drawing in Augmented Reality on Mobile Phones

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Felix Wehnert

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Bastian Leibe

Registration date: 2018-01-16
Submission date: 2018-04-09

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Contents

List of Figures

List of Tables

Abstract

In 2017, ARKit and ARCore arrived on mobile devices, enabling markerless augmented reality experiences on everyday smartphones. However, the potential for creating new input methods offered by this has not yet been fully realized. For this Bachelor's thesis we will examine the viability of creating an input device capable of capturing movements in a three-dimensional space, which works in conjunction with every ARKit-capable device.

Our first step will consist of designing the input device. For this we have decided on a pen due to its intuitive use. The pen will be designed in a 3D modeling program and produced using a 3D printer. Due to the fact that we are going to use a low-cost production method and will require no special hardware our approach will be affordable to most consumers.

To facilitate the tracking functionality the pen must include surface areas on which markers can be affixed, as these are required to determine the pen tip's position and orientation. In addition the pen will contain buttons to perform custom actions. Combining the proven and well-known technology of marker detection with the innovative ARKit framework, we will implement an application to determine the pen's position and orientation, and compute the pen tip's location from this data.

Once we have completed the implementation and have a working pen, we will start evaluating this prototype by conducting a user study. From the participants we want to know how the pen behaves in terms of tracking and ergonomics.

Überblick

Im Jahr 2017 kamen ARKit und ARCore auf mobile Endgeräte, wodurch markerlose Augmented Reality-Erlebnisse auf alltäglichen Smartphones ermöglicht wurden. Das damit verbundene Potenzial zur Schaffung neuer Eingabemethoden ist jedoch noch nicht voll ausgeschöpft. Für diese Bachelorarbeit werden wir die Machbarkeit eines Eingabegerätes untersuchen, das Bewegungen in einem dreidimensionalen Raum erfasst und mit jedem ARKit-fähigen Gerät zusammenarbeiten kann.

Unser erster Schritt besteht darin, das Eingabegerät zu entwerfen. Hierfür haben wir uns aufgrund der intuitiven Bedienung für einen Stift entschieden. Der Stift wird in einem 3D-Modellierungsprogramm entworfen und mit einem 3D-Drucker gedruckt. Aufgrund der Tatsache, dass wir eine kostengünstige Produktionsmethode verwenden und keine spezielle Hardware benötigen, wird unser Ansatz für die meisten Verbraucher bezahlbar sein.

Um die Tracking-Funktionalität zu erleichtern, muss der Stift Flächen besitzen, auf denen Marker angebracht werden können, da diese zur Bestimmung der Position und Orientierung des Stiftes benötigt werden. Zusätzlich enthält der Stift Knöpfe, um eigene Aktionen durchzuführen. Durch die Kombination der bereits bewährten Technologie der Markererkennung mit dem ARKit-Framework werden wir eine Anwendung zur Bestimmung der Position und Orientierung des Stiftes implementieren und aus diesen Daten die Position der Stiftspitze berechnen.

Sobald wir mit der Implementierung fertig sind und einen funktionierenden Stift haben, werden wir mit der Evaluierung dieses Prototyps beginnen, indem wir eine Anwenderstudie durchführen. Von den Teilnehmern wollen wir wissen, wie sich der Stift in Bezug auf Tracking und Ergonomie verhält.

Acknowledgements

First I want to thank the chair Media Computing Group and professor Jan Borchers who give me the opportunity and the support to make this thesis possible.

Furthermore I want to thank my advisor Philipp Wacker who supported me during this thesis.

I would like to thank Prof. Dr. Leibe for agreeing to be the second examiner.

Last but not least I want to thank Jan Thar, Christian Schmidt and Oliver Novak who helped me operating the 3D printers in the FabLab.

Chapter 1

Introduction

With the release of the new Augmented Reality (AR) frameworks "ARKit" for iOS and "ARCore" for Android in 2017, new ways of interaction with the mobile phone have arisen [Apple][Google, a]. Instead of displaying content flat on the screen, three-dimensional content can now be placed in the user's direct environment. The created virtual objects can then be inserted into a live camera feed. Creating a user-friendly interface that works well with augmented reality is definitely a challenge. We take up this challenge and would like to go one step further by creating an interaction device that not only allows the user to interact with the existing augmented reality but even to create new three-dimensional content.

In Virtual Reality (VR) systems like the HTC Vive or the Oculus Rift, controllers that are tracked and with which a user can interact in space are an integral part of the system. This technology cannot simply be transferred to mobile augmented reality systems because special sensors are required to track the controllers, which are not available in mobile phones. Our vision is to develop an input device for mobile augmented reality, which is qualitatively comparable with the existing ones from VR.

3D content is nowadays created on computers with 2D screens. Even if there are some approaches to create 3D content in AR and VR that we will discuss in more detail

later, there is definitely still a need for research in this area [Billinghurst et al., 2009]. This is where we want to start and develop a technique with which three-dimensional objects can also be created or modified using three dimensions. In the context of this Bachelor's thesis we create a pen with which the user can freely draw in her Augmented Reality. We develop an iOS app that tracks this pen accordingly, so that the pen is directly connected to ARKit. Unfortunately, ARKit alone is not able to track small objects in space. To do this we need an additional framework that works together with ARKit to achieve our goal. In order to track the pen, we need identifiable markers on the pen, which can be used to calculate the pen's location and orientation. Since such a pen does not yet exist, we first have to model it and then have it printed by a 3D printer. In this work we first design the pen and set up requirements for our app. To track our pen we connect ARKit and another framework so that it is possible to recognize it and paint with its tip, the position of which has to be calculated from the markers. To let the pen communicate with the app via Bluetooth we embed a chip and some push buttons into the pen, to allow further input possibilities. After we have met all the objectives, we carry out a user study in which we ask the participants how they rate the pen in relation to various criteria.

Chapter 2

Background

In this chapter we give a brief overview about topics that will be mandatory to understand for this thesis. We will begin with the theory of Augmented Reality and follow with implementations that will make Augmented Reality possible on mobile phones. We close this chapter with an introduction to personal fabrication.

2.1 Augmented Reality

Augmented reality is a view of reality which is enriched with computer-generated content

In Augmented Reality, a live view of a real-world environment is enriched with computer-generated content [Schueffel, 2017]. This view may then be enhanced using sound, video or haptics. The first development of an AR-like system happened in the early 1990s at the U.S. Air Force's Armstrong Labs [Rosenberg, 1993]. They created Virtual Fixtures, which consisted of a helmet with an integrated display and two controllers, each of which was connected to a robot arm. Their purpose was to allow operators to remotely perform tasks with better spatial understanding. Not long after that, AR also became interesting for the gaming and entertainment industry who hoped for a deeper user experience [Molla and Lepetit, 2010] [Oda et al., 2008].

The first release of an open-source AR framework that

AR on mobile phones already existed in 1999

could be used to develop an arbitrary AR application was ARToolKit in 1999 [ARToolKit]. This framework could detect the position and orientation of computer-readable markers or planar images and was used for real-time augmented reality applications.

2.2 Marker Tracking Frameworks

In the early 2000s, ARToolKit and other frameworks were adapted to make AR on smartphones possible [ARToolKit]. ARToolKit was ported in 2005 to Nokia's Symbian OS, which is a mobile operating system and the AR gaming industry made their first mobile AR games.

Multiple tracking frameworks with different focuses were developed. In addition to markers and other computer-readable patterns, images of any kind can now also be used for tracking [Sörös et al., 2011].

Marker tracking frameworks perform a graphical analysis of the camera image to detect the marker

To calculate the position and orientation of any marker the framework has to perform a graphical analysis of the camera feed or input image [Garrido-Jurado et al., 2014]. The framework first begins with an adaptive thresholding to separate the markers from other striking objects in the field of vision. Then those markers which are not rectangular, too small or too big are filtered out. After all candidates are determined, the framework tries to decode the information that is encoded in the marker. This information may consist of a simple ID, but may also include additional data [Garrido-Jurado et al., 2014].

In contrast to detecting the marker's position, detecting the orientation in space is more complex

If the decoding was erroneous, it is likely that the candidate was not a marker but only a falsely recognized object. From a successfully recognized marker one can easily compute the direction of the object relative to the camera by calculating the mid point of the rectangle. However, compared to determining the position of a marker, it is more difficult to calculate its orientation. To do so, the framework has to know the position of the marker's corners and the optical parameters of the user's camera. With this information, it is possible to estimate the object pose given a set

of object points. At this point, we still do not know the distance of the object from the camera. For this, the real-world size of the marker must also be included in the calculation [Garrido-Jurado et al., 2014].

Marker tracking alone does not meet our requirements

The limitations of marker tracking is that the tracking is only relative to the camera. With marker tracking it is only possible to detect movement of the camera when the marker is stationary or vice versa. It is not possible to detect any movement when the marker and the camera are both moving. For this reason marker tracking alone is not suitable for our needs. We need a system that can handle a moving camera *and* a moving marker.

2.3 ARKit

ARKit is a new markerless augmented reality by Apple. The lack of any markers enables new AR experiences

In 2017 Apple introduced ARKit for iOS [Apple]. It is a markerless AR framework which can render virtual objects live into the camera feed. *Markerless* means that ARKit can detect a world environment without the need for specific markers. This opens new possibilities for AR experiences because there is no need for a marker that previously had to be printed out or produced in another way. A flat object like a table can be detected automatically and virtual objects can be placed onto the table. Furthermore, ARKit can track the camera's point of view in relation to a fixed world point. ARKit sets the world origin to the point the camera was at during the initialization process. After that, ARKit will calculate the position relative to this origin. One advantage of ARKit is that the internal coordinate system is metric, so measurements and distance calculation is very easy. To create a fluent AR experience ARKit uses various sensors and fuses this motion data with the camera feed [Apple]. This fusion is called *Visual Inertial Odometry* (VIO) which allows the device to sense how it moves through a real-world coordinate system.

ARKit can do even more than marker frameworks

In addition, ARKit is able to perform *Scene Understanding* and *Light Estimation*. With Scene Understanding, ARKit can detect horizontal and vertical planes in the camera view. A developer can use this information, for example, to place

objects directly on a table and thus create even more immersive AR experiences. Light Estimation is used to render virtual objects with a shadow that fits into the real-world environment. ARKit tries to determine the location of the light source responsible for the shadowing. This location is given to the renderer to be applied to the virtual objects. Sugano et al. proved that shadows help embed virtual objects better in the real environment.

In contrast to marker tracking frameworks, the current version of ARKit is not able to detect any kind of markers. As ARKit is not able to read markers, it is not able to do a high precision tracking of objects in the real-world environment. To provide this functionality, other frameworks have to be used in conjunction with ARKit.

2.4 Personal Fabrication

With personal fabrication, anyone can manufacture products that were previously only available to the mass industry

Personal fabrication is a new social trend to produce products independently that normally are only produced in an industrial environment. With consumer-ready production machines like 3D printers, CNC machines or laser cutters, everyone is now able to manufacture a product. Not only easy-to-use materials like wood and leather, but also materials that need more advanced hardware like metal, acrylic glass or plastic can be used. These materials were formerly only present in a professional production environment, e.g. mass production [Mota, 2011]. At the moment these machines are too expensive for individual users, so they can be found in shared maker hubs. These hubs are present all over the world and offer interested makers hardware and a place to work. For a small fee everyone can use the offered machines. These hubs are often called *FabLabs*, an abbreviation of fabrication laboratory [Mota, 2011].

During this thesis we use the 3D printers offered by the FabLab of the RWTH Aachen University. 3D printers are able to print an arbitrary object of nearly any shape. With new 3D printers it is even possible to print with multiple materials in one object, allowing for the printing of multicolored objects.

Chapter 3

Related work

Two larger research areas that are related to this thesis are immersive modeling and sketching in 3D. Immersive modeling describes the creation of 3D objects in free space, where sketching creates 3D models using 2D graphics. One example for immersive modeling is TiltBrush by Google [b]. With TiltBrush and a VR setting like an Oculus Rift or HTC Vive users can draw 3D artwork directly in the environment. The focus of TiltBrush is much more consumer-focused. The concept of the final product is similar, but the TiltBrush developers themselves did not have to integrate any kind of tracking, as this is part of the used VR hardware. Another approach is CavePainting by Keefe et al. [2001]. This system, however, only works inside of a specially configured VR cave, where many different sensors are available for tracking purposes.

Most immersive modeling approaches were implemented in VR

In AR there are also approaches to immersive modeling, such as HoloSketch [Hol], an app for the HoloLens, which can display 3D objects in augmented reality. However, a user is not able to draw freely in space, only some basic objects can be created.

One example of sketching in 3D is Lift-Off by Jackson and Keefe [2016] who have developed a pen whose special strength is to draw freehand in a VR environment. To make it easy for the users, they first draw two-dimensionally and extrude that drawing into the third dimension. The pen is

tracked by sensors mounted in a VR cave. The user can employ the pen to interact with the VR environment and e.g. has access to virtual menus. Since the user can hold two of these pens in her hands, she has significantly more options to give input than in our solution.

Laviole and Hachet [2012] developed PapARt, which is capable of tracking a sheet of paper and projecting a 3D scene onto it. To the user, it looks like the sheet of paper is a window to the virtual world behind it. To enable this functionality, markers have to be printed at certain locations on the sheet. The AR framework can then detect a user rotating the sheet, as well as touching it or placing objects on it.

Another example of editing an object was developed by Magnenat et al. [2015], working at Disney. Instead of modifying the 3D mesh, they developed a live texturing method of 3D objects to modify their appearance. This system allows the user to paint on a sheet of paper and a mobile device maps the painted area onto a 3D object.

PaintSpaceAR [Pai] is an app that uses ARKit to draw in free space. Instead of creating an input device, they treat the iPhone itself as an input device. The user can create objects using the touchscreen while moving the iOS device through the room. The object is drawn relative to the world space, so it stays where it was created.

Another related project was developed by Wacker et al. [2018], who developed a system that tracks a pen with special hardware. The aim of this paper was to find out to what extent physical or virtual guidance improves users' 3D drawings. The used hardware includes an expensive 3D tracking system named VICON and a HoloLens that must be worn by the user. We are very much inspired by this work, although with a different focus. Thus, we are more interested in using simple hardware to reach a wider target audience than the existing systems do.

We want to create a solution without expensive hardware

In contrast to related work, our goal is to offer a user an opportunity to interact precisely in augmented reality without the need for expensive hardware.

Chapter 4

Design

Our goal was to design and create an input device with which users can make inputs in AR and see the result directly on their mobile phone. A proven input device is the pen, capable of performing particularly detailed movements, but limited to a 2D surface. Inspired by the pen, we wanted to develop a 3D pen that offers the same functions for AR. We also defined our goal as having a button to start and stop the drawing functionality. Besides the pen, we also need an app that tracks the pen, detects the pen tip, communicates with the pen button, and then displays the drawing. The user should be able to move the mobile phone around the drawing so that she can enjoy the full AR experience. Another goal was that the user can use the pen and app anywhere, regardless of their environment. In addition, the app should be easily expandable so that future studies, developments or other research work can also build on this foundation. Another requirement was to export the drawing so that the user can print or save it.

We want to create a pen that enables the user to interact with virtual 3D content

Even if the title of this thesis states "on Mobile Phones", we first put our focus on the iOS platform with ARKit as an AR framework because we have a better knowledge there. Google, the developer of the mobile operating system Android, released their world tracking AR framework "ARCore"[Google, a] two months after ARKit in August 2017. Its features are comparable with ARKit's features. Thus, the problems we had to solve on iOS with the help of ARKit

Since our competences are in the area of iOS, we will first focus on this platform

could be solved on Android with ARCore. Although we built our app for iOS, apart from ARKit, none of what we use is iOS-exclusive.

Thus, the design can be divided into two major parts: the pen that the user holds in her hand, as well as the app, which must communicate with the pen. The first part contains the detailed software design and the second the hardware design.

4.1 Software Design

We want to use ARKit because it works in a very high quality

We already evaluated ARKit and determined that it does a very good job in terms of world tracking. The object positioning is very precise and the object stays exactly in place even during excessive movement. In case ARKit misses the position of the object, it often repositions the object after a short delay. This is due to VIO, which does not only use the motion sensor but also the camera feed as sensory input.

Furthermore, ARKit gives access to the raw feature point cloud it is working on that could be used to track objects in the real-world environment. A feature point is a three-dimensional point in the real world that is located at a solid object's surface. The feature point cloud thus is the set of all detected feature points in the current frame. Using this point cloud allowed us to get a basic 3D view of the environment. While this could be used to track larger objects, in the evaluation it turned out to be too complicated and computationally expensive.

We need a framework that works with ARKit

As already stated ARKit alone did not meet our requirements of a world tracking system with a high precision object or marker tracking. Currently ARKit is the only framework with world tracking in this quality, so we needed to find a solution that works *with* ARKit. We searched for other frameworks that are specialized in the mentioned marker tracking and evaluated them in different categories.

There are multiple frameworks that differ in quality, costs, feature richness and more. Before we decided on a frame-

Name	Tracking	Cost	Marker	OS	iOS
EasyAR	+	free	+	✗	✓
Vuforia	+	500$	+	✗	✓
OpenCV ArUco	o	free	o	✓	✗
ArUco 3	+	free	o	✓	✗
ARToolKit 5	-	free	+	✓	✓
ARToolKit 6	+	free	+	✗[6]	✓

Table 4.1: This table contains the results of our analysis of marker tracking frameworks. In the "Tracking" column we evaluated how we liked the tracking in terms of quality and speed. The "Cost" column indicates how much we have to pay for the use. "Marker" means which marker variations are possible. The more markers are recognizable, the more freedom we have in designing the pen. "OS" means whether the library is open source and we can make changes to the source code. The last column "iOS" means if there is a ready-to-use iOS binary we can use.

work we needed to specify what we expect from the marker tracking framework. Our most important decision factor was quality of tracking as we wanted to create a high precision input device. Other decision factors were cost to use the framework, size and appearance of the markers, detection speed, open-source code base, a ready-to-use iOS binary and some minor factors like a good documentation. We looked at several libraries, namely EasyAR[1], Vuforia[2], arUco as an OpenCV contrib module[3], ARToolKit[4] in version 5 and 6 and ArUco 3[5], which was published as an improved standalone framework.

We evaluated some frameworks and evaluated them in different categories

Table 4.1 presents the results of our evaluation. Although Vuforia is a high-quality AR framework, we unfortunately could not use it due to the high price. ARToolKit 5 did not satisfy our demands in terms of tracking quality. All other frameworks met our requirements and proved to be

[1] https://www.easyar.com

[2] https://www.vuforia.com

[3] https://docs.opencv.org/3.3.0/d5/dae/tutorial_aruco_detection.html

[4] https://artoolkit.org

[5] https://www.uco.es/investiga/grupos/ava/node/26

[6] Because ARToolKit 6 was in beta version the sources are not yet publicly available

very promising during the evaluation. We started to build proof of concept-implementations with EasyAR, ARToolKit 6 and OpenCV ArUco. Very early in development we faced a major problem: ARKit and the chosen frameworks both require exclusive access to the camera. Fortunately it was possible to attach a second framework behind ARKit, because ARKit gives access to the raw image data of the last frame. For this reason the framework we chose had to support a frame-by-frame input instead of a camera stream.

To use a marker framework it must not require exclusive access to the camera

Unfortunately, ARToolKit 6 and EasyAR did not support frame-by-frame input, although EasyAR plans to add support for this mode of input in future versions [Eas]. As both ARToolKit 6 and EasyAR are closed source it was not possible for us to implement this functionality by ourselves. Therefore the only option that remained is arUco, even though its tracking quality is not the best. Furthermore, we had to manually build the iOS binary, as arUco binaries are only available in x64 architecture. Mobile operating systems like iOS or Android on the other hand run on an arm64 architecture. Thus, we had to modify the framework and its dependencies by adjusting the build configuration files in such a way that they are compatible with arm64. This results in an iOS-ready framework.

We decided to use arUco, as it fulfill the most important requirement of providing a frame-by-frame input

After the framework was built, we could grab the frame input from ARKit and give it to the detection class of arUco. We developed a proof of concept application which uses arUco in combination with ARKit. At this point we started to work on the main app. Fortunately, some days after we got OpenCV arUco working, Garrido-Jurado et al. [2014] released the third version of arUco as a standalone version. The update promised improvements in speed and tracking quality and we verified this claim with another proof of concept-implementation. With arUco 3 we finally found the framework that we can use to develop our app.

The developers of arUco released a major update that we use now

Our app should be very simple and easy to use. For the first version it sufficed to have visualization whether the pen is currently being tracked, and what its location is. In addition, it should be possible to draw a path in 3D space. Furthermore, we wanted the user to be able to export her drawing so that she is able to send it to a 3D printer and

The app should be able to draw a path in 3D space

print it there. In a settings area the user should be able to set the length of her pen or connect to the pen via Bluetooth.

4.2 Hardware Design

The hardware design mainly involved designing the pen in our 3D CAD application Autodesk Fusion 360[1]. The challenge with the pen was to ensure that it is large enough to offer enough space for computer readable markers as well as the hardware, while making sure not to neglect ergonomics. The pen should be detectable from any angle, so at least one marker should always be visible. We designed a first prototype that basically looked like a cube with a cylindrical extrusion on top, as can be seen in Figure 4.1. The six sides of this cube offer space for up to five detectable markers and the holder, which is attached on the last side. Furthermore, we named the pen "ARPen".

For this thesis we have to design a pen from scratch

We named our pen "ARPen"

Although we could have worked with this version, we were not completely satisfied and made further adjustments. We have sharpened the pen tip so that the pen looks more like a real pen and it becomes clear that this tip is intended for drawing. Furthermore, we did not make the holder round but triangular in order to better embed buttons in the pen. In addition, we were of the opinion that this resulted in a more comfortable grip. The most significant change, however, was a realignment of the marker cube. This not only further improved the holding sensation, but also enabled six markers to be attached, which significantly increased precision and viewing angle stability.

After a first version we did some major improvements

We also needed a lid so that it is possible to reach the inside of the cube, where our hardware is contained. Closing mechanisms for 3D printed objects are particularly tricky, as they are usually very delicate and exceed the existing print resolution. Since the printed material is not flexible, we had to develop a mechanism that does not require parts of the object to be bent. We printed a small block in each

We designed a closing mechanism for the cube that works well with 3D printing

[1] https://www.autodesk.com/products/fusion-360

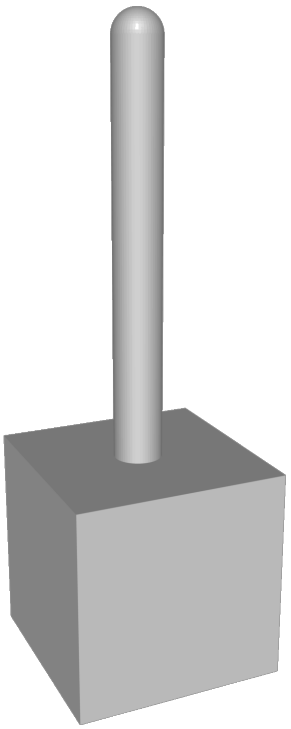


Figure 4.1: For the first version we took a pen and created an area where five markers can be attached.

corner of the lid so that it contacts the wall of the cube when one attaches the lid onto the cube. Since 3D printers print in many layers, tiny grooves are created in the walls of the cube and at the edge of the blocks on the lid. These grooves snap into each other so that the lid holds firmly to the cube. This makes it very easy to attach and detach the lid.

Printing the markers directly into the pen improves the tracking quality significantly

To hold the ARPen in our hands we need to 3D print it. We used the printers kindly offered by RWTH Aachen University's FabLab. Because the ARPen would fall over during the printing process, we needed to print the ARPen in multiple parts. We worked a lot with this improved version but it still had one major issue. The markers are printed on paper and glued around the box and inaccuracies in gluing have extremely degraded the precision. Therefore, we needed to attach the markers in a more stable way with less room for inaccuracies. Fortunately, the FabLab also offers

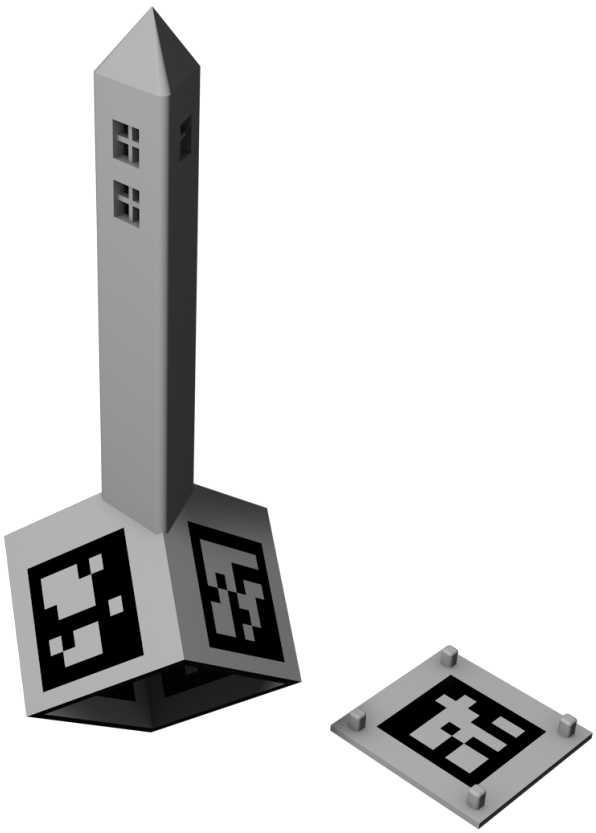


Figure 4.2: The improved version of the ARPen

multi-material 3D printers, so we printed the marker with black material right into the white box. The result of these various changes to our design can be seen in Figure 4.2.

To enable the user to decide whether she wants to draw a line or not, we have attached three buttons to the ARPen. The drawing shows the corresponding openings. These three buttons are connected to a Bluetooth chip which transmits the current button state to the iOS device. The buttons must be soldered to a cable that runs through the holder right into the cube. The cube contains the Bluetooth chip, the cabling and a battery.

We embedded buttons in the pen and connected them to a Bluetooth chip

As a Bluetooth chip we chose a small, low budget and power saving chip. We had no requirements concerning the computing power of the chip as the chip's only job is to

We used a cheap, power saving and low cost Bluetooth chip

transmit the state of the buttons. After some research we decided on the RedBear Nano v2[1], because this chip supports Bluetooth low energy, has 11 digital I/O pins that can be used for buttons or other accessories and works with a common 3.7 V LiPo battery. We used an inexpensive battery with a capacity of 150 mAh, which is enough to power the ARPen for multiple hours.

[1] https://redbear.cc

Chapter 5

Implementation

In this chapter we will describe in detail how we implemented the app, the tracking, the integration of the tracking into ARKit and the implementation of the chip.

5.1 ARKit

We created the app using the boilerplate code that is provided by Xcode. For ARKit there are multiple starter projects that we could build upon. There is one project for every supported rendering engine: Metal, SceneKit and SpriteKit. SpriteKit was not suitable for us because it is a 2D rendering engine. Metal and SceneKit are 3D rendering engines that can be used in conjunction with ARKit. Metal is a low-level hardware-near interface to the graphics processor and SceneKit is a high level engine that is built on top of Metal. Next to the rendering engine we also specified the programming language of the project. The choices were Objective-C[1] and Swift[2], a new and modern programming language.

Setting up ARKit is very easy as Xcode provides ready-to-use example and boilerplate code

[1] https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[2] https://swift.org

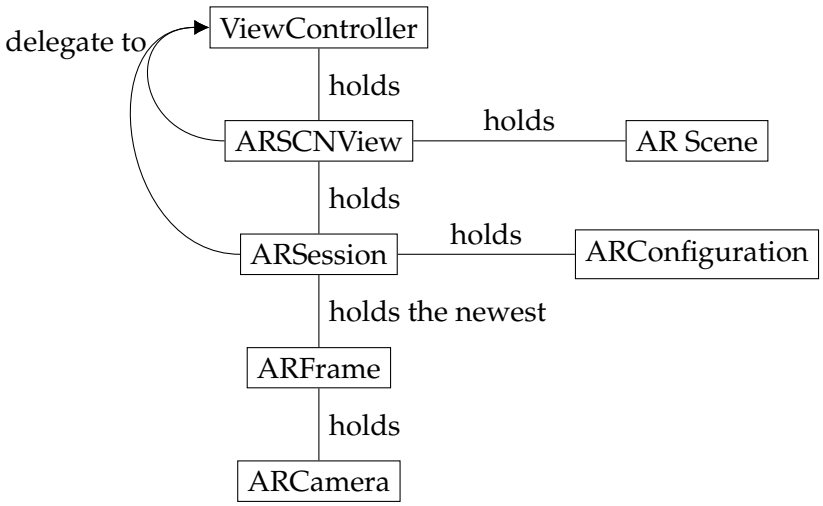


Figure 5.1: A brief architectural overview of the ARKit framework. The `ARSession` is the most important object as it controls the behavior of ARKit and other important information can be accessed. ARKit contains other classes, but since we do not use them, they are not shown in the diagram.

Furthermore, all ARKit-related objects were already initialized and the project contained a simple AR scene that was displayed on app startup.

We chose the SceneKit project as there is no need for the low-level API from Metal. As main programming language we chose Swift. In order to import external C++ frameworks, Objective-C and Objective-C++ must later be used.

To start developing we needed to understand how ARKit is built and how a developer can use it. Figure 5.1 shows an architectural overview of the important parts of ARKit.

All classes with the "AR" prefix are implemented by ARKit so our job was to implement the `ViewController`. In the boilerplate code provided by Xcode this was already done. However, we chose to disable some features we did not need in order to be more performant, e.g. plane detection and light estimation.

Figure 5.2: Three arUco markers. From left to right, the patterns in the markers encode the numbers zero, one and two, respectively.

5.2 arUco

The integration of arUco into an iOS app was not easy at all. First we needed to compile the framework for the arm64 architecture as all ARKit-ready iOS devices run on arm64. For this we also needed to compile arUco's dependencies for arm64. Fortunately, arUco has only two of them: Eigen, a library for linear algebra, and OpenCV, a computer vision library. OpenCV offered a compiled iOS version that we can use but Eigen had to be manually compiled for arm64. It was also important to compile it into a static rather than a dynamic framework as a static one did cause fewer linking problems. As arUco is written in C++ we needed to write an Objective-C++ wrapper around the framework because Swift was not able to link to C++ written binaries directly. To do this we should understand how arUco works.

arUco must be compiled manually, as no binary for iOS is available

We need to use Objective-C++ to link to arUco

To use arUco we only had to initialize the main class `aruco::MarkerDetector`. To do so we had to provide the camera parameters, which could be partly taken from the `ARCamera`'s property `intrinsics`. For the other part we needed to use the camera calibrator, provided by arUco.

Furthermore, we specified the marker dictionary. That dictionary defines which types of markers are used in our program. We used `ARUCO_MIP_36h12` which is recommended by the developers and offers a good tradeoff between robustness and size. These markers have a simple design of 6×6 white or black squares, surrounded by a black line. Moreover, they are never symmetrical to avoid problems

We used a recommended marker dictionary

in calculation [Garrido-Jurado et al., 2016]. An example of an arUco marker can be seen in Figure 5.2.

Detecting a marker from an image was quite easy with arUco

We fed the detector with an image by calling `detector.detect(image, markers, camera Parameters, markerSize)`. In addition, the already mentioned `cameraParameters` and the real-world `markerSize` was needed for the calculation. The resulting markers appeared in the `markers` vector. We then got the ID of all detected markers and their three-dimensional relative position to the camera.

ARKit outputs an image that we used as an input for arUco

To feed the `detector` we needed to get the image from ARKit to arUco. The only point where we could get the raw image is the `session(_:didUpdate:)` method from the `ARSessionDelegate`. This method provided the newest captured camera image that we can use to convert to an arUco friendly format. The type of the image was a `CVPixelBufferRef` that could easily be converted to a `cv::Mat`, the image format from OpenCV that arUco uses for its inputs. Both types were a simple representation of an image. In ARKit each pixel is encoded as an unsigned 8-bit integer with one channel, so we used the `cv::Mat` initializer with the argument `CV_8UC1`, which specifies exactly this pixel encoding. The initializer does not copy the image but uses the same reference to the image, thus this process is very fast.

Using multi-threading increased the performance

We were then able to get the newest image from ARKit to OpenCV. To make the detection even faster we decided to run the detection process in a separate thread. For this we used the `NSOperationQueue` abstraction that made threading significantly easier. Additionally, we ensured that only one detection process runs at once.

5.3 Fusion of ARKit and arUco

Listing 5.1 shows our definition of the wrapper between ARKit and arUco. The wrapper needs to be initialized and the delegate needs to be set. The delegate must conform to the `OpenCVWrapperDelegate` protocol to receive

```
@protocol OpenCVWrapperDelegate
  -(void)markerPosition:(NSArray<NSValue*>*)  pos
                    ids:(NSArray<NSNumber*>*) ids;
  -(void)noMarkerFound;
@end

@interface OpenCVWrapper : NSObject
  @property id<OpenCVWrapperDelegate> delegate;
  -(void)findMarker:(CVPixelBufferRef)pixelBuffer;
@end
```

Listing 5.1: Definition of our arUco wrapper

the result from the calculation in a defined manner. After that, the ARKit delegate can call the `findMarker:` method and after successful computation the wrapper's delegate is called with an array of markers and the respective IDs. In case no marker was detected, the `noMarkerFound` method is called by the wrapper.

We called the wrapper in the already mentioned delegate method `session(_:didUpdate:)` and waited for the delegate method to be called. In the callback method we could display e.g. a red dot at the position were the marker was found. For this we created a `SCNNode` and added it to our AR Scene. We had to take into account that the node was positioned relative to the world origin point, which did not move. However, the positions given by the wrapper were relative to the camera, which moves through the room. Thus, we needed to link the position of the markers in some way to the camera's position and rotation. Fortunately, a camera is also a `SCNNode` which can have children. The transformation, including the position, rotation and scale, is also applied to every child node. The red dot representing the position of the marker could be attached to the camera as a child node and was therefore positioned correctly.

We introduce a red dot indicating the position of the marker in the AR space

When we wanted to draw a path through three-dimensional space with the marker, we had to convert the marker position back into space, otherwise the marker would still have moved with the camera. For this, SceneKit provided the handy method `SCNNode.convertPosition(_ position:`

We converted the marker position to the world space to made it independent of the camera position

```
class MarkerBox : SCNNode {
    init(length: Double)
    func set(position: SCNVector3,
             orientation: SCNVector3,
             forID id: Int)
    func posititonWith(ids: [Int]) -> SCNVector3
}
```

Listing 5.2: The Swift interface of the MarkerBox class that will take care of all calculation related operations

`SCNVector3, to node: SCNNode?)` that is able to convert vectors between local and global space. This could be used to create a path in world space.

Milestone. *We reached our first major milestone as we are able to draw a path with the marker relative to the world space.*

5.4 Marker Box

To calculate the pen tip we had to calculate the orientation of the marker

One of our goal was to draw with the pen tip and not with the marker itself. We had not only one marker but rather six markers placed on the faces of a cube. Therefore, we needed to calculate the pen tip position from the marker box. To do this we needed to know the position and ID of the marker we could detect in the camera image. Furthermore, we also needed to calculate the orientation of the marker, the calculation of which arUco provides a method for. We adjusted the `OpenCVWrapper` and the associated protocol to execute this method and provided the result of the calculation next to the position of the marker, so the `OpenCVWrapperDelegate` then received the ID, the position, and the orientation. To centralize this calculation we introduced another class named `MarkerBox`. The Swift interface of this class is shown in Listing 5.2.

To clarify how the calculation works, Figure 5.3 shows a technical drawing of the ARPen. Line b in the drawing represents the length of the pen which the `MarkerBox` must be initalized with. For each marker we call the `set(position:rotation:id:)` method and with this

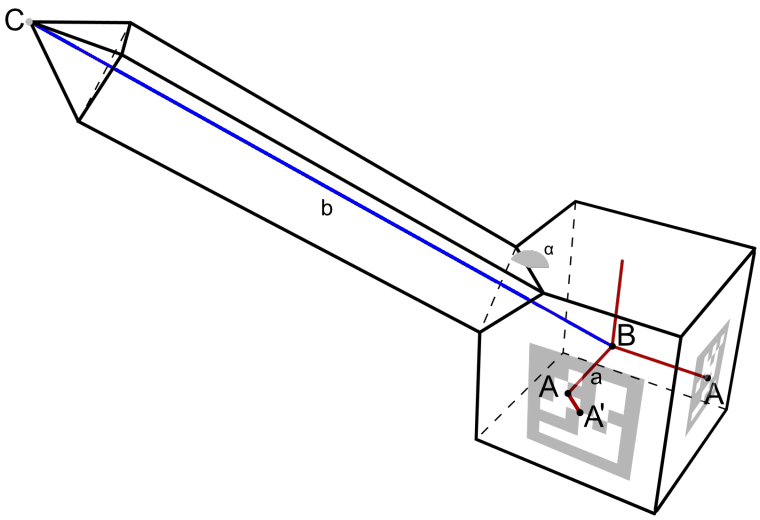


Figure 5.3: A technical drawing of the ARPen that shows how the pen tip is calculated from the markers. A is the middle of the side, A' is the detected position of the marker, B is the center of the box and C is the pen tip. The vector a describes the vector from the marker to the middle point and b from the middle point the pen tip.

information the `MarkerBox` object is able to calculate the pen tip point, C in the drawing. This calculation is run by calling `positionWith(ids:)`.

Some markers are displaced a little bit to support tracking from some view angles

In theory, we have to calculate B from all possible As and then applying the vector b so we result at C. However, not all markers were exactly positioned in the middle of their side. The markers on the faces adjacent to the holder have the special feature of being displaced 5 mm away from the holder. By doing this we ensured that the space between the markers and the holder was large enough to not influence the detection of the markers' edges. The middle point of the marker is A, but the middle point of this face is A'. The marker on the other sides are exactly positioned in the middle, thus $A = A'$.

To simplify the calculation, we worked intensively with the fact that `SCNNode`s inherit the transformation to their child nodes. We created separate nodes for each marker and added a new child node. This child node got a fixed trans-

We took advantage of SceneKits internal node structure to simplify the pen tip calculation

formation, so that it always pointed from the marker to the tip of the pen. Only the position and rotation of the node representing the marker was set. The child node then automatically aligns itself with the pen tip. As we had multiple markers we did this for every marker that was detected and calculate the average point of all pen tips.

To calculate C from A we used the following vector:

$$\vec{b} = \begin{pmatrix} \frac{\cos(180° - \alpha) * b}{\sqrt{2}} \\ \frac{\cos(180° - \alpha) * b}{\sqrt{2}} \\ \sin(35.3° * b) - 0.02 \end{pmatrix}$$

In the case $A \neq A'$, i.e. for the translated markers, we used a modified vector:

$$\vec{b} = \begin{pmatrix} \frac{\cos(180° - \alpha) * b - 0.005}{\sqrt{2}} \\ \frac{\cos(180° - \alpha) * b - 0.005}{\sqrt{2}} \\ \sin(35.3° * b) - 0.02 \end{pmatrix}$$

These vectors were formed by the addition of the vectors $\vec{a}$ and $\vec{b}$ in the drawing. b is the length of the pen that was previously given to the `MarkerBox` and α was measured in our CAD application, where we designed the ARPen, to be $144.7°$.

This calculation is only performed during the initialization. It is only performed again if the length of the ARPen is adjusted in the settings. As already mentioned the child nodes have a fixed position (the result of the calculation above) and only the parent nodes are changing their position and orientation. These characteristics are updated by the `set(position:orientation:id:)` method. After all positions and orientations have been set, the `positionWith(ids:)` method can be called, which calculates a pen tip position for every detected marker. This is done by converting the position $(0, 0, 0)$ in relation to the child node to world space by calling SceneKit's `convertPosition(position: SCNVector3, to node: SCNNode?)` method with `position` as $(0, 0, 0)$ and `node` as `nil`. If `node` is `nil`, SceneKit converts the position $(0, 0, 0)$ of the child node, representing the pen tip, to the world space. All positions

We calculated the pen tip and converted this position to the world space

are taken to calculate the average point, which results in a very good approximation of the pen tip's position in the real world space. Any objects that are placed at that location are not anymore in relation to the camera, like the marker, but in relation to ARKit's world space. When the camera moves, the generated object does not move with the camera, as is always the case with marker tracking frameworks, but remains in place.

Milestone. *We reached our second milestone as we were able to draw with the pen tip. We calculated the pen tip to a world space, so the objects did not move with the camera. Furthermore, we supported an adjustable pen length.*

To draw a path, we created very small cylinders between the positions we calculated at each frame. From a distance, this looked like a coherent path. While this was not the most efficient method of accomplishing our goal, it was sufficient for our purposes.

5.5 Bluetooth Chip

One problem we had is that the pen was drawing a path when it is visible in the viewport and the user could not control whether the pen should draw a path or not. We decided to add a button to the ARPen that controls this behavior. This implicated that we needed a chip that detects the state of the button and is capable of transmitting this information to the iOS device. We already evaluated that a RedBear Nano v2 fit exactly our needs, as it is inexpensive and has low power consumption. The chip could be programmed using the Arduino toolchain and a special programming device which was included with the purchase of the chip. This toolchain includes an IDE, an SDK for Bluetooth communication and a compiler.

As a Bluetooth chip we use the RedBear Nano v2

To read the button state we used the functions provided by the SDK. Furthermore, we had to design a proper cabling. This cabling is shown in Figure 5.4. It should be noted that no further resistors are needed as the chip includes pull-up resistors, which are used. To read the button state we

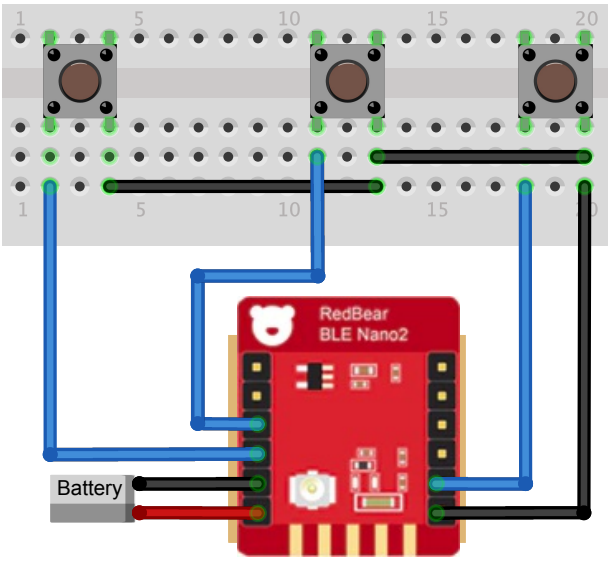


Figure 5.4: The Bluetooth chip and its cabling. Three buttons are connected to the I/O ports of the Bluetooth chip. In addition, we have attached a 3.7V battery to the corresponding pins.

We used the internal pull-up resistors to keep the cabling simple

applied current to the respective line and read the logical value behind the pull-up resistor. When the button is not pressed the current flows out via the pull-up resistor and we a read a logical one. When the button is pressed, the current flows directly to ground through it, and we read a logical zero.

In Bluetooth every device is an peripheral that contains any number of services

A Bluetooth device connected to an iPhone is called peripheral. Each peripheral has a name and a UUID. A peripheral can have any number of services and each service says that this peripheral supports a certain function. Furthermore, a service is also specified by a UUID. For example, if the peripheral could measure heartbeat data, it would offer the ”heartbeat” service. Every device that connects to the peripheral knows which functions the corresponding peripheral supports thanks to the services. For some services, such as heart rate, the UUID is standardized, so all Bluetooth heart rate monitors should have the same service UUID.

A service is divided into several characteristics. Each char-

acteristic is an endpoint at which data can be read and written. In the case of the heartbeat service, the standard provides that there is one characteristic for the heartbeat in beats per minute and another for the position of the sensor on the body.

Services are divided into characteristics

For the ARPen we defined that there is only one service and under it only one characteristic. This characteristic changes whenever one of the 3 buttons is pressed. The iPhone can now say it wants to be informed when this endpoint changes. This allowed us to transfer the states of the buttons.

Our Bluetooth application is based on an example project provided by RedBear as part of their Arduino toolchain. To be able to read out the button states, we switched on the pull-up resistors at the corresponding pins. Furthermore, we created a ticker which excludes the button states every 100ms and changes the value of the Bluetooth characteristic if the state has changed.

If the app now wants to connect to the pen, it searches for all Bluetooth peripherals that support the corresponding service and connects to the first one found. If this is not the correct one, the user can call up a list of all peripherals in the settings and connect specifically to one.

The app connects to the pen by searching for a predefined service UUID

The characteristic we have defined contains only ASCII data. This data is limited to values such as, for example, `B1:UP` or `B2:DOWN`, which indicate that Button 1 has been released or Button 2 has been pressed. The app reads the data and can perform actions depending on the button number and state. In our case a path is drawn by pressing the button `B1` and drawing stops by releasing it.

5.6 App Architecture

The system we developed here could be used for multiple scenarios. Any idea that went in the direction of AR could use the ARPen and build on it. To support this we designed a `PluginManager` and a `Plugin` protocol to make it very

```
protocol Plugin {
    func didUpdateFrame(scene: PenScene,
                        buttons: [Button: Bool])
}
```

Listing 5.3: The Plugin protocol

The app uses a plugin architecture that makes further development easy

easy for future developers, who want to add their own functionality to the app. As long as their code conforms to the `Plugin` protocol, as shown in Listing 5.3, their plugin can be seamlessly integrated into our app, which only requires registering at the `PluginManager`. At every frame, the plugin is called by the `PluginManager`, and is given information such as the current button states as well as the scene object. With the `buttons` object the plugin is able to interact with all buttons and the `scene` object is needed to add or remove visual nodes from the current AR scene and to read the current pen tip position. To make the app even more modular, we have also made the basic functionality of drawing a line with the pen into a plugin. This makes it possible to switch this function on or off as desired. It is also a good example for developers who want to write a new plugin.

The user can export the drawing directly to a 3D printer

Our long-term goal is to develop an app with which the user can create and edit 3D objects. It is therefore particularly important that the 3D object created by the user can also be exported. Therefore, we have written a function which allows to export the whole scene to an STL file. STL is a very common format for 3D printing [Hiller and Lipson, 2009].

Within this goal we have also developed an `ObjectCreation` plugin, which allows the user to create an object in space. With the pen a user can define points in the room and the plugin creates an object between these points.

The length of the pen can be adjusted

Furthermore, the user should have the possibility to edit the length of the ARPen. We have created a settings view where the user can adjust the length of the ARPen, as well as disconnect, connect via Bluetooth and export the scene as mentioned above.

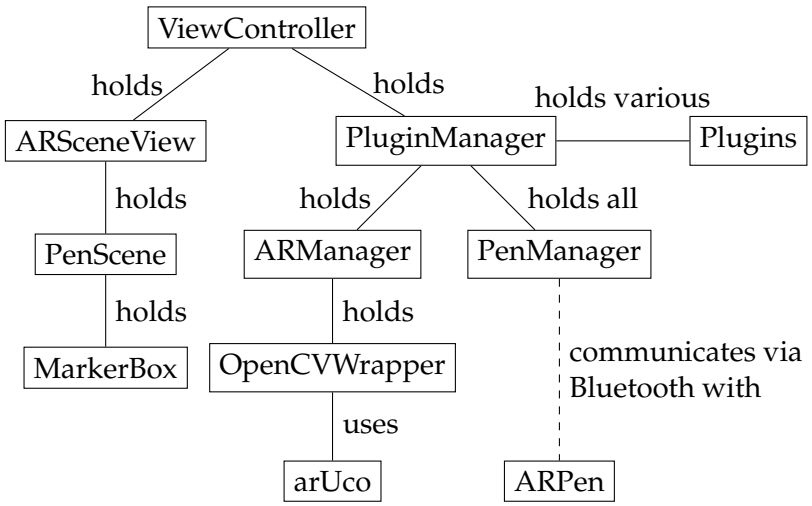


Figure 5.5: A brief architectural overview of our app. Every class has its own responsibility. Furthermore, it is important that every class is held by only one other class to reduce the risk of memory leaks.

To keep the code clean and to ensure that no class has multiple responsibilities, we split the `ViewController` into the `PenManager` class dedicated to the Bluetooth communication and the `ARManager` dedicated to all ARKit related functions. To encapsulate these classes even more from each other and to give the plugins access to information that these two classes hold, they are not held directly by the `ViewController` but by the `PluginManager`. The `PluginManager` also holds all active plugins. However, the `ViewController` holds the `ARSceneView`, which is prominently displayed to the user after the app starts. In order for the `ARManager` to have easy access to the scene that is needed to display the pen position, for example, a pointer to the `PenScene` is given to the `PluginManager` and through it to the `ARManager`. Since the `PenScene` holds the `MarkerBox`, the `ARManager` is able to pass the marker detection results to the `MarkerBox` object. Figure 5.5 visualizes the app architecture.

The architecture of the app is designed so that each class has its own responsibility

Chapter 6

Evaluation

One of the big advantages of our system is that it only needs the ARPen and an iPhone 6s or newer. Unfortunately, Apple does not publish exact iPhone sales figures, therefore the number of ARKit-capable devices can only be estimated. Most estimates currently range from 250 million to 500 million ARKit capable devices [van Dijk] [Boland]. This gives us a very large potential user base.

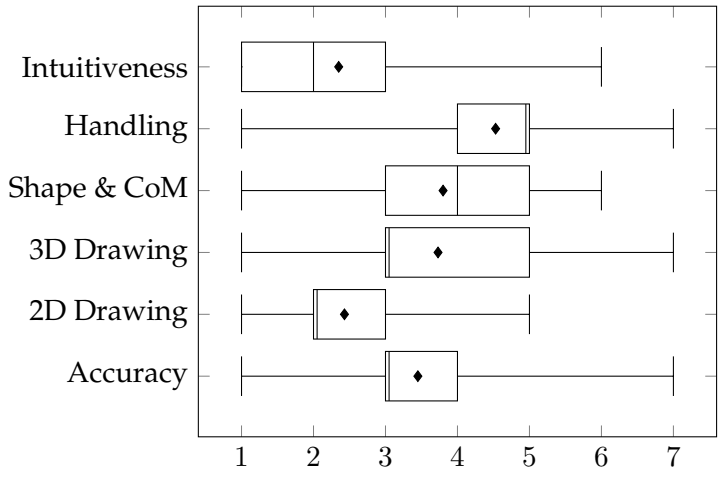
Therefore, we conducted a study in which a user was asked to use the ARPen for some tasks and then was asked to give a short feedback on the ARPen. We interviewed 20 people and asked various questions. Of the participants, 16 were male, two were female and two chose not to provide any information. 19 right-handers and one left-hander participated. On average, our participants were 22.06 years old.

20 people participated in the study

Participation in the study consisted of three phases. In the first phase, participants used the ARPen to draw whatever they like in order to get a general impression of the ARPen. In the second phase the participants are asked to follow a two-dimensional form with the ARPen. In the last phase, they are asked to draw a three-dimensional object. The study took roughly 15 minutes.

Unfortunately, our study also had some limitations. At that time, the implementation was not yet designed to support different orientations, so participants had to use the iPhone

At the time of the study, only landscape mode could be used



Screen size

Too large	Perfect	Too small
0 %	40 %	60 %

3D modeling experience

None	Little	Much
70 %	15 %	15 %

Figure 6.1: Result of the evaluation

in "Landscape Left" mode, so the camera was on the left side. Right-handed people sometimes found this annoying because they wanted to hold the iPhone with their left hand which resulted in their hand covering the camera.

After the study, the participants completed a questionnaire. In this questionnaire, participants were asked to rate 6 questions on a scale of 1 to 7, with 1 being the best result. Additionally, they were given the opportunity to leave a free text. The questionnaire is attached in the appendix A "Questionnaire".

The evaluation of the questionnaire is shown in Figure 6.1. Some strengths and weaknesses can be quickly identified. The pen achieved the best results in 2D drawing and in operability. Since the pen is tracked directly, even without a Bluetooth connection, and this is also indicated by a small red dot at the tip of the pen, every participant quickly

The best result was achieved in the area of intuitiveness

adapted to drawing with the ARPen. In addition, the interaction carried out in the study is kept simple, as only the button responsible for drawing has to be used. These circumstances certainly had an influence on the good rating.

Two-dimensional drawing worked much better than three-dimensional drawing

The good marks in the area of two-dimensional drawing could be traced back to the fact that everyone has existing experience of drawing with "normal" pens [Yang and Cham, 2007] which can easily be transferred to the ARPen. However, very few users had significant experience in 3D modeling. 15% of the participants said they had a little experience in this area and another 15% said they had a lot of experience. The lack of experience could influence the lower grade in three-dimensional drawing. In order to make a statement whether experienced participants evaluate the ARPen better in three-dimensional drawing, we need to gather more data.

The worst result achieved the pen in the area of handling and the position of the center of mass

The worst results were achieved in the categories "Shape & CoM", where "CoM" refers to Center of Mass, and "Handling", where we asked how it feels to hold the iPhone and the ARPen simultaneously. Since the battery and Bluetooth devices are located in the back of the ARPen, the center of mass of the pen is very far back, unlike a conventional pen. This makes the ARPen a little more difficult to hold than a regular pen.

Most participants had constant problems simultaneously holding the ARPen and the iPhone in their hands. It often happened that the participant concentrated too much on the tip of the pen and the marker left the camera image. This stopped the tracking of the ARPen while drawing. Since a certain distance between the camera and the ARPen is needed to be able to use it well, participants with a smaller arm length had bigger problems.

The accuracy of the ARPen was assessed by the participants in the middle range. Things that affect tracking are usually the lighting conditions, the speed with which the participant moves the pen and the texture of the environment. In highly textured environment with good light, the participants had fewer problems with tracking. In addition, smaller things are more difficult to draw with ARPen than

Sometimes ARKit oriented the scene to the ARPen and not to the environment

larger ones due to the limited accuracy of the ARPen. It also happened more often that ARKit was not oriented to the environment but to the pen, which led to particularly disturbing effects. This occurred especially when drawing 2D objects when the participant made particularly slow movements. ARKit assumes that the observed scene is static and does not move, because the observed movements in the scene are used for tracking. When the participant holds her device still while slowly moving the pen, ARKit can no longer distinguish accurately enough whether the iPhone is moving or not. The camera image showing the pen indicates that the iPhone is moving, but the motion sensors do not indicate this. ARKit decides to give more weight to the camera image compared to the motion sensors. This causes the scene to move with the pen, so the participant is no longer able to draw. This problem always occurred when the pen was a large part of the visible area and the pen was moving slowly. Some participants recognized the problem, and were able to make ARKit understand the scene correctly by moving the iPhone simultaneously, by increasing the distance between camera and pen, or by moving the pen faster.

This has caused problems in slow drawing

Many parallels could be found in the free text area

On the back of the questionnaire the participants could comment on the categories "What do you like", "What do you not like" and "What changes would you suggest" regarding the ARPen. In the category "What do you like" we often read that the participants liked the idea and the concept of the pen. The participants also wrote that the triangular shape of the pencil made it particularly pleasant to hold. In the category "What do you not like" we often read that the center of mass and the field of vision needed improvement. Since these participants were dissatisfied with the field of vision, they often wished for an enlargement of the field of vision when making suggestions for improvement. In the field "What changes would you suggest" many participants wished to have the possibility to draw a straight line, for example by smoothing the drawn line. Inaccuracies in the measurement, paired with recalibrations by ARKit lead to jittery lines.

We also asked how the participants felt about the screen size. All participants conducted the study with an iPhone

7. This has a screen diagonal of 4.7 inches. 60% of the participants thought the screen was too small. In connection with the feedback in the free text field we come to the conclusion that most participants desire a larger field-of-view. This was especially prevalent among participants who held the iPhone close to the ARPen.

In addition to the questionnaire, we also observed the participants during the use of the pen and identified similarities between different participants. Especially the three-dimensional drawing was a huge challenge for some participants. Some did not draw the given 3D object three-dimensionally, but rather had their attention fixed on the two-dimensional display of the iPhone while drawing. They drew the object so that it looked fine on the display, but looked at from other perspectives in the three-dimensional space the object was malformed. After the participants moved the iPhone around their object, they noticed their mistake and were surprised. Three-dimensional drawing does not seem to be as intuitive for many participants as two-dimensional drawing.

We learned a lot by observing how the participant use the ARPen

Almost all participants enjoyed participating in the study and were quite satisfied with the idea and its implementation. Many of the suggestions for improvement noted can certainly be addressed in the future in order to further increase the acceptance of ARPen.

Chapter 7

Conclusion

In this chapter we will summarize the content of this work. We will briefly discuss the last chapters and give an outlook on what can still be improved about ARPen and which ideas can be built on it.

7.1 Summary

In this Bachelor's thesis we presented how to build an app that recognizes and tracks a pen in three-dimensional space. First, in the chapter 2 "Background" we built up the necessary previous knowledge on the topics "ARKit", "Marker Tracking Frameworks" and "Personal Fabrication" and briefly described how these technologies work.

Afterwards, in the chapter 3 "Related work" we dealt with other research in the field of Augmented Reality.

In the chapter 4 "Design" we have defined which goals we want to achieve. We decided to build a pen as well as an app that tracks the pen in an augmented reality environment. ARKit is a framework that offers markerless world tracking, but it's not good at tracking a small object like a pen. Therefore, we need another framework besides ARKit to follow the pen in space. We have tried out various frame-

works for this. After our analysis, we chose arUco 3 because it meets our most important requirement of not requiring exclusive access to the camera, as ARKit also requires exclusive access. A proof of concept has shown that the interaction of ARKit and arUco 3 not only works in theory but also in practice.

Now that we know what the app must be able to do, we have described what features the pen should have. This of course includes the look, which differs from a conventional pen only in the large box on which the markers are attached. After we presented a first simple version, we considered further and found some possibilities for improvement which we also implemented. Finally, we designed an improved version which we still use as of this time. It differs fundamentally from the first version in its design; one more marker can now be attached and there is space for buttons.

In the Implementation chapter we described how we developed the app, starting with chapter 5.1 “ARKit”. Fortunately, ARKit is very easy to use and the configuration effort is low.

In the chapter 5.2 “arUco” we built the arUco library for iOS and integrated it into our project. Parts of arUco had to be made compatible with ARKit before they could be used together. From this point on, the app was able to recognize markers.

The connection between ARKit and arUco was then made in the chapter 5.3 “Fusion of ARKit and arUco”. There we transferred the images that ARKit delivers after each processing to arUco which searches for markers in these images. We transferred the position of the markers to world space and were thus able to draw in world space with a marker. However, drawing with the marker directly does not yet fulfill our requirements.

In the chapter 5.4 “Marker Box” it was our goal to draw with the pen tip, for which we had to calculate the position of the tip relative to the marker. SceneKit helped us make the calculation as efficient and simple as possible, because

we made particularly intensive use of SceneKit's structure. In addition, we exploited the way how node translation are applied to children nodes. Now we could paint with the pencil tip, a milestone was reached. To make the painting experience even more realistic we had to integrate a button into the pen with which the user decides when she wants to draw a path.

In 5.5 "Bluetooth Chip" we explained how Bluetooth works and what must be done to create a simple Bluetooth device. The buttons are connected to a Bluetooth chip placed in the box of the pen. The app establishes a connection with the pen and can now recognize a button being pressed and react accordingly.

In chapter 5.6 "App Architecture" we described how we built an architecture that can be extended by anyone to allow other developers, researchers or other interested parties to easily add features to the app without having to understand the entire app and familiarize themselves with the more complicated tracking. Therefore, we have designed a plugin architecture that accepts plugins and provides them with all necessary information to add any features to the app. In case a plugin needs more information than it is given, we have also described the rest of the architecture in detail to make subsequent changes as easy as possible.

In chapter 6 "Evaluation" we presented our user study, which we conducted to test how users handle ARPen and especially how satisfied they are with it. The results were satisfactory. Users found the ARPen very intuitive and were also able to draw in two dimensions. However, the ARPen still has weaknesses in handling and three-dimensional drawing.

7.2 Future work

We have some suggestions that can further improve the ARPen. In the study the user noticed that they want an extended field of view to be able to use the ARPen closer to the camera. To achieve this we would like to try to clamp

a fisheye lens in front of the iPhone lens. Certainly this would affect ARKit's tracking, but we do not yet know how strong these effects are. Another way to increase the field of view is to build a holder that holds the device on its own, for example a holder that holds the device in front of the body, to extend the distance between the camera and the ARPen. Furthermore, the user does not have to use both hands, which allows her to concentrate on the ARPen.

For a better user experience we need to enlarge the field of view

Another problem is the marker box, which often leaves the iPhone's field of view, causing the user to be unable to continue drawing. Enlarging the field of view can help, of course, but we want to find a more effective method. One approach is to place a marker near the tip of the ARPen that allows further tracking even when the marker box has left the field of view. Furthermore, even though this is very complex, the pen could be equipped with a gyroscope and an accelerometer, so that it is able to track its position in space independently.

The new ARKit 1.5 could help us to solve some problems

Shortly before the completion of this thesis Apple has released ARKit 1.5. Some new features are interesting for us, such as the increased resolution of the ARKit. Instead of a 720p video stream at 30 fps, a 1080p at 60 fps stream is now possible. Due to the increased resolution, marker tracking could provide better results. The increased fps can help track the pen while it is moving, but also increases the requirement on arUco which has to recognize the marker in half the time to keep the live view fluid. Another feature of ARKit are so-called "Reference Images". ARKit can use these images to orientate an AR scene to a known image in the real world. Unfortunately ARKit cannot track the reference images in the room, so it is not a replacement for arUco. However, the images could be used to improve the stability of tracking. If a reference image is placed in the real world so that it is captured by the camera from time to time, ARKit could easily calibrate the scene on this anchor image over and over again. ARKit's deviations and recalibrations that the participants of the study experienced could be reduced.

The plugin system makes it particularly easy to extend the software. Other developers have already written plugins to

add more functions to the app. It is possible to implement extensive methods for the creation and editing of 3D objects. With good tracking it also makes sense to create better editing possibilities for 3D objects, such as inserting new objects into the scene or boolean operations between objects. Sharing the created 3D content with other app users is also conceivable.

It would also make sense to offer an alternative to the printed pen. 3D printers are still difficult to reach for many people and should therefore not be a prerequisite for using the pen. It is conceivable to fold the pen purely from paper. All input methods should then be made via the touch screen of the iPhone. With a pen printed on paper, the technology can be made available to an even wider audience.

Appendix A

Questionnaire

The following four pages contain the questionnaire in english and german language that we used to evaluate the ARPen.

Number:

Evaluation ARPen

Gender: ☐ Female ☐ Male ☐ Other

Age: _____

left handed ☐

right handed ☐

Knowledge in 3D modelling ☐ no ☐ some ☐ a lot

How do you rate the accuracy of the ARPen?

very accurate — very unprecise

What is it like to draw 2D objects with the ARPen?

easy — hard

What is it like to draw 3D objects with the ARPen?

easy — hard

How does the ARPen feel in terms of shape and center of gravity in the hand?

very good — very bad

How is it for you to hold the iPhone and ARPen in your hands at the same time?

easy — hard

How do you feel about the operation of the hardware?

intuitive — unintuitive

How do you feel the screen size of the iPhone?

too large ☐ perfect ☐ too small ☐

What do you like about the ARPen?

What do you not like about the ARPen?

What changes would you suggest that would improve the pen?

Thank you for the participation

Nummer:

Evaluation ARPen

Geschlecht: ☐ Weiblich ☐ Männlich ☐ k.A.

Linkshänder ☐

Alter: _____

Rechtshänder ☐

Erfahrung mit 3D-Modellierung ☐ keine ☐ wenig ☐ viel

Wie beurteilst du die Genauigkeit des ARPens?

sehr genau — sehr ungenau

Wie ist es, mit dem ARPen 2D Objekte zu zeichnen?

einfach — schwierig

Wie ist es, mit dem ARPen 3D Objekte zu zeichnen?

einfach — schwierig

Wie fühlt sich der ARPen im Bezug auf Form und Schwerpunkt in der Hand an?

sehr gut — sehr schlecht

Wie ist es für dich das iPhone und den ARPen gleichzeitig in den Händen zu halten?

einfach — schwierig

Wie empfindest du die Bedienung der Hardware?

intuitiv — unintuitiv

Wie empfindest du die Bildschirmgröße des iPhones?

zu groß ☐ genau richtig ☐ zu klein ☐

Was gefällt dir am ARPen?

Was hat dir am ARPen nicht gefallen?

Welche Veränderungen würdest du vorschlagen, die den ARPen verbessern würden?

Vielen Dank für deine Teilnahme

Bibliography

CameraFrameStreamer Class - EasyAR documentation. URL https://www.easyar.com/doc/EasyAR%20SDK/API%20Reference/2.0/CameraFrameStreamer.html. Accessed on 2018-04-02.

Get HoloSketch - Microsoft Store. URL https://www.microsoft.com/en-us/store/p/holosketch/9p3br4t5m4tv. Accessed on 2018-04-02.

Paint Space AR. URL https://www.paintspacear.com/. Accessed on 2018-04-02.

Apple. ARKit - Apple Developer. URL https://developer.apple.com/arkit/. Accessed on 2018-03-10.

ARToolKit. About artoolkit's history and team — artoolkit.org. URL https://artoolkit.org/about-artoolkit. Accessed on 2018-02-26.

Mark Billinghurst, Hirokazu Kato, and Seiko Myojin. Advanced Interaction Techniques for Augmented Reality Applications. In *Lecture Notes in Computer Science*, pages 13–22. Springer Berlin Heidelberg, 2009. URL https://doi.org/10.1007/978-3-642-02771-0_2.

Mike Boland. ARtillry Intelligence: 380 Million iPhones Are Compatible with ARKit ARTILLRY: A PUBLICATION AND INTELLIGENCE FIRM FOR AR & VR. URL http://artillry.co/2017/07/26/artillry-intelligence-arkits-installed-base-is-380-million-iphones/. Accessed on 2018-03-24.

S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, jun 2014. URL https://doi.org/10.1016/j.patcog.2014.01.005.

S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition*, 51:481–491, mar 2016. URL https://doi.org/10.1016/j.patcog.2015.09.023.

Google. ARCore - Google Developer — ARCore — Google Developers, a. URL https://developers.google.com/ar/. Accessed on 2018-03-10.

Google. Tilt Brush by Google, b. URL https://www.tiltbrush.com/. Accessed on 2018-03-18.

Jonathan D Hiller and Hod Lipson. STL 2.0: a proposal for a universal multi-material Additive Manufacturing File format. In *Proceedings of the Solid Freeform Fabrication Symposium*, number 1, pages 266–278. Citeseer, 2009.

Bret Jackson and Daniel F. Keefe. Lift-Off: Using Reference Imagery and Freehand Sketching to Create 3D Models in VR. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1442–1451, apr 2016. URL https://doi.org/10.1109/tvcg.2016.2518099.

Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola. CavePainting. In *Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01*. ACM Press, 2001. URL https://doi.org/10.1145/364338.364370.

J. Laviole and M. Hachet. PapARt: Interactive 3D graphics and multi-touch augmented paper for artistic creation. In *2012 IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE, mar 2012. doi: 10.1109/3dui.2012.6184167.

Stephane Magnenat, Dat Tien Ngo, Fabio Zund, Mattia Ryffel, Gioacchino Noris, Gerhard Rothlin, Alessia Marra, Maurizio Nitti, Pascal Fua, Markus Gross, and Robert W. Sumner. Live texturing of augmented reality characters

from colored drawings. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1201–1210, nov 2015. URL https://doi.org/10.1109/tvcg.2015.2459871.

Eray Molla and Vincent Lepetit. Augmented reality for board games. In *2010 IEEE International Symposium on Mixed and Augmented Reality*. IEEE, oct 2010. URL https://doi.org/10.1109/ismar.2010.5643593.

Catarina Mota. The rise of personal fabrication. In *Proceedings of the 8th ACM conference on Creativity and cognition - C&C '11*. ACM Press, 2011. URL https://doi.org/10.1145/2069618.2069665.

Ohan Oda, Levi J. Lister, Sean White, and Steven Feiner. Developing an Augmented Reality Racing Game. In *Proceedings of the 2nd International Conference on INtelligent TEchnologies for interactive enterTAINment*. ICST, 2008. URL https://doi.org/10.4108/icst.intetain2008.2472.

Louis B. Rosenberg. Virtual fixtures as tools to enhance operator performance in telepresence environments. In Won S. Kim, editor, *Telemanipulator Technology and Space Telerobotics*. SPIE, dec 1993. URL https://doi.org/10.1117/12.164901.

Patrick Schueffel. *The Concise Fintech Compendium*. School of Management Fribourg, 2017.

Gábor Sörös, Hartmut Seichter, Peter Rautek, and Eduard Gröller. Augmented visualization with natural feature tracking. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia - MUM '11*. ACM Press, 2011. URL https://doi.org/10.1145/2107596.2107597.

N. Sugano, H. Kato, and K. Tachibana. The effects of shadow representation of virtual objects in augmented reality. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* IEEE Comput. Soc. URL https://doi.org/10.1109/ismar.2003.1240690.

Owen van Dijk. How many devices can run ARKit? A lot. Owen van Dijk Medium. URL

https://medium.com/@ohwhen/how-many-devices-can-run-arkit-a-lot-f49f2f9675c8. Accessed on 2018-03-24.

Philipp Wacker, Adrian Wagner, Simon Voelker, and Jan Borchers. Physical Guides: An Analysis of 3D Sketching Performance on Physical Objects in Augmented Reality. In *To appear in CHI '18 EA: Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, April 2018. ACM. URL https://doi.org/10.1145/3170427.3188493.

Maria C. Yang and Jorge G. Cham. An Analysis of Sketching Skill and Its Role in Early Stage Engineering Design. *Journal of Mechanical Design*, 129(5):476, 2007. URL https://doi.org/10.1115/1.2712214.

Index

Typeset April 6, 2018